Modélisation et rejeu basés sur des règles

Maxime Gaide

THESE

Pour l'obtention du Grade de
Docteur de l'École Nationale Supérieure de Mécanique et d'Aérotechnique
(Diplôme National - Arrêté du 25 mai 2016 Modifié par l'Arrêté du 26 Août 2022)

Ecole Doctorale :
MIMME Mathématiques Informatique Matériaux Mécanique Énergétique

Secteur de Recherche : Informatique et Applications

Présentée par :

Maxime GAIDE

****

Modélisation et rejeu basés sur des règles

****

Directeur de thèse : **Stéphane JEAN**
Co-encadrants : **David MARCHEIX** et **Xavier SKAPIN**

****

Soutenue le 11 décembre 2024

devant la Commission d'Examen

****

JURY

Rapporteurs :

David CAZIER	Professeur des Universités, Université de Strasbourg
Guillaume DAMIAND	Directeur de recherche, Université Claude Bernard

Examinateurs :

Agnès ARNOULD	Maître de Conférences, Université de Poitiers
Stéphane JEAN	Professeur des Universités, Université de Poitiers
Pascale LE GALL	Professeur des Universités, CentraleSupélec
David MARCHEIX	Maître de Conférences, Université de Poitiers
Jean-Luc MARI	Professeur des Universités, Aix-Marseille Université
Xavier SKAPIN	Maître de Conférences, Université de Poitiers

Remerciements

Ces trois années ont été une aventure sur beaucoup de plans et tout ceci aurait été impossible sans un nombre remarquable de personnes, qu'elles se soient impliquées ou non dans ces travaux, que je les connaisse depuis longtemps ou que l'on se soit simplement croisée. Je souhaite vous remercier.

Pour commencer, je remercie mon directeur de thèse, Stéphane Jean, pour ses conseils et sa bienveillance. Merci d'avoir accepté de diriger cette thèse, pour tes conseils et d'avoir trouvé les mots pour me rassurer quand j'en ai eu besoin. Je remercie mes encadrants, David Marcheix, Xavier Skapin et Agnès Arnould pour leur investissement dans cette thèse et la confiance qu'ils m'ont accordée. Grâce à vous, j'ai progressé bien plus que je n'aurais jamais osé l'imaginer. Je remercie Hakim Belhaouari qui a été un soutien précieux pendant ces trois ans, et je ne parle pas que de Jerboa. J'ai apprécié chacune de nos conversations qui ont systématiquement piqué ma curiosité même si elles étaient, pour la plupart, très courtes.

Je remercie les rapporteurs et les membres du jury de cette thèse. Un grand merci à David Cazier et Guillaume Damiand d'avoir accepté d'être rapporteurs de cette thèse. Merci à Pascale Le Gall d'avoir accepté de présider ce jury. Merci à Jean-Luc Mari, qui suit mon parcours depuis le master, d'avoir accepté de faire partie du jury. Encore une fois, j'en suis très heureux. Un grand merci pour vos retours, vos questions et les pistes qu'elles soulèvent.

Je remercie tous les membres du LIAS pour leur accueil chaleureux, leur bonne humeur et leur aide. Un grand merci à Bénédicte pour sa gentillesse et sa prévenance. Merci à Mickaël pour son aide sur les aspects techniques et les relectures de dossiers. Merci à Brice pour son aide lors des soirées jeux (c'est clairement toi l'animateur de ces soirées) et pour la décoration de nos portes. Et un grand merci aux collègues doctorant·e·s pour les encouragements mutuels, ça aide les jours où on préfèrerait rester au lit. Merci à Ali, Luc et Quentin, le temps défile à toute vitesse à chaque fois que je passe vous voir, c'est terrible. Merci à Thomas, très heureux de partager le même bureau que toi et j'adore nos énigmes au tableau.

J'ai aussi eu le plaisir de travailler avec les membres du XLIM et plus particulièrement avec l'équipe Informatique Graphique que je remercie pour son accueil. Je remercie mes collègues doctorant·e·s qui ont remis l'ADIIS sur pied, ça me fait toujours plaisir de vous voir que ce soit pour un workshop ou une soirée.

Je remercie mes amis, Cédric, Frank, Gabriel, Jorge, Matis, Mehdi et Thibault pour leur soutien, on ne discute pas assez, mais ça ne m'empêche pas de penser à vous tous.

Je remercie aussi Louise, Matheus et Richard, je suis tellement heureux de vous avoir rencontré, merci d'être incroyables. Je ne pourrais jamais trop insister sur le fait que sans vous ça n'aurait pas pu être aussi bien. Merci pour les midis jeux de cartes, les soirées souvent improvisées, l'usine satisfactory trop tôt tombé dans l'oubli, les eoguessr, le bureau que j'ai piqué à la première occasion et j'en passe.

À toute ma famille, Gauthier, Timotee, Anton, mes parents, Philippe, mes grands-parents, Bruno et Evelyne, merci de m'avoir soutenu quand j'en ai eu besoin. Gauthier, saches que t'es encouragements m'ont vraiment fait chaud au cœur et que ça compte beaucoup pour moi.

Enfin, je remercie ma compagne qui est à mes côtés depuis près de sept ans et qui compte plus que tout pour moi. Julia, merci de m'avoir supporté quand j'étais fatigué, angoissé, de mauvaise humeur. Merci pour ton entêtement, de me forcer à dormir quand j'essaie de faire des nuits blanches. Avec toi, je sais que je peux tout surmonter.

Table des matières

Introduction

Contexte

Depuis le début des années 60, les logiciels de modélisation géométrique ne cessent de gagner en popularité, à tel point que ceux-ci sont présents dans tous les aspects du quotidien. Tout d'abord développés pour la conception et la fabrication de pièces industrielles [Autb ; Dasa ; Sie ; Dasb], les logiciels de modélisation géométrique se sont étendus vers de nombreux autres domaines. Nous pouvons identifier, notamment, les domaines de la créativité et du divertissement au travers du cinéma et des jeux vidéos [Auta ; Autc ; Ble], mais aussi ceux de l'architecture [Autd] et de l'aménagement urbain [ESR]. Avec ces logiciels, nous pouvons observer que diverses façons de créer et manipuler des objets virtuels, des *modèles*, ont été développées. Nous retrouvons, entre autres, la génération procédurale et la modélisation paramétrique.

La génération procédurale est une méthode de création de modèles basée sur un corpus de règles qui peuvent être des grammaires formelles [Lin74] ou des instructions composées d'opérations de modélisation. Ces règles peuvent alors être paramétrées afin de générer des familles d'objets comme des végétaux [Ter+09 ; BTG15], des bâtiments [Mül+06 ; HMV09 ; QB15] ou des villes [ESR], voire des mondes entiers pour le cinéma et le jeu vidéo [Auta ; God ; Autc ; Uni ; Epi].

La modélisation paramétrique est une autre approche de création où un modèle est construit par applications manuelles successives d'opérations, regroupées sous le nom de *processus constructif*. À chaque étape de construction, l'opération appliquée est enregistrée dans une *spécification paramétrique* et une nouvelle représentation géométrique (aussi appelée *instance courante*) est générée. Cette approche de création est particulièrement répandue dans la conception de pièces industrielles, mais aussi pour la conception de mobiliers ou encore dans la fabrication de prothèses médicales. Pour le grand public, ces outils [Ble ; Fre] offrent également la possibilité de concevoir des pièces de rechanges afin de les fabriquer à l'aide d'imprimantes 3D.

Au fil des années, la nécessité de modéliser des objets toujours plus complexes est allée de pair avec les avancées théoriques et technologiques. Dans certains cas, cette nécessité se concrétise par le besoin de modéliser des objets de dimension quelconques. Des modèles de représentations comme les cartes combinatoires [Vin83] ou les cartes généralisées [Lie91 ; DL14] ont alors été élaborées pour permettre la construction de modèles n-dimensionnels [ALS15], c'est-à-dire génériques en dimensions. Dans cette logique, des librairies telles que CGoGN [Kra+14] et CGAL [Dam24a ; Dam24b ; Dam24c] proposent des modules permettant de traiter ces modèles. Parallèlement, d'autres approches permettant la manipulation des cartes généralisées ont été proposées.

Ainsi, des approches formelles sont apparues avec l'utilisation de grammaires [PL90 ; WS05 ;

BB10 ; BTG15] pour la génération procédurale de végétaux, ou encore des règles de transformation de graphes [Pou+08]. Ces règles de transformation de graphe ont, par la suite, été étendues au langage de règles Jerboa [Bel+14 ; Bel+17] puis au langage de script Jerboa [Gau19]. En particulier, les règles Jerboa suivent un ensemble de contraintes garantissant que l'application d'une règle sur une carte généralisée de dimension n préserve à la fois sa cohérence topologique et géométrique.

Aujourd'hui, des modeleurs utilisant ces formalismes, tels que les cartes généralisées ou les règles Jerboa, permettent la simulation physique [Ben+17] et la reconstruction de réservoirs géologiques [Gau+16 ; Gau19]. Ils sont également intégrés dans des modeleurs paramétriques pour implanter des mécanismes dits de *réévaluation* (ou *rejeu*) de modèles [Bab10 ; Car19].

Problématiques

La modélisation d'un objet est une tâche fastidieuse impliquant des cycles successifs d'essais et de corrections. En effet, la modification d'un objet nécessite souvent de reprendre manuellement l'intégralité du processus de modélisation. Pour alléger cette tâche, les logiciels de modélisation paramétrique proposent d'enregistrer chacune des étapes de modélisation dans une spécification paramétrique. Il est alors possible d'éditer les paramètres des opérations contenues dans cette spécification paramétrique puis de la réévaluer, c'est-à-dire réappliquer une à une les opérations en prenant en compte les éventuelles modifications apportées. Un mécanisme de réévaluation doit, ainsi, permettre la génération automatique d'une nouvelle représentation géométrique à partir d'une spécification paramétrique éditée.

Depuis les années 90, la réévaluation de spécifications paramétriques fait l'objet de nombreuses études pour la génération de variantes de modèles 3D. En effet, la mise un œuvre de ce mécanisme se heurte au problème de l'évolution des entités topologiques lors de la réévaluation. Afin de permettre l'appariement de ces entités, il est alors indispensable de mettre en place un mécanisme d'identification dit persistant. Un tel mécanisme doit permettre une identification à la fois unique et sans ambiguïté de chacune des entités topologiques enregistrées dans une spécification paramétrique. Ce problème est celui de la *nomination persistante*.

Pour résoudre ce problème, la plupart des approches consistent en la construction d'un nom référençant le voisinage topologique d'une entité [Che95 ; Kri95 ; CCH96 ; AMP00]. Dans certains cas ambigus, c'est-à-dire lorsque la topologie seule ne suffit pas à identifier une unique entité topologique, des informations géométriques sont ajoutées au nom persistant [Wu+01 ; BNB05 ; WN05].

Pour la nomination basée sur le voisinage topologique, il est nécessaire d'effectuer le suivi des évolutions des entités topologiques au cours de l'évolution du modèle. Ce suivi permet, au rejeu, de mettre en œuvre une phase d'*appariement* où les entités du modèle initial sont mises en correspondance avec des entités équivalentes dans le modèle rejoué. Aujourd'hui, deux approches existent pour suivre les évolutions d'entités topologiques : soit en réalisant un parcours exhaustif ou localisé du modèle initial et en le comparant au modèle réévalué ; soit en codant directement et explicitement les changements topologiques (ou *évènements*) dans le code des opérations de modélisation. Un parcours total en vue de comparer deux versions d'un même modèle permet ainsi de détecter l'ensemble des évènements. Néanmoins, une telle approche est coûteuse en temps de calcul en fonction de la complexité du modèle. Au contraire, le codage des

évènements à détecter directement dans les opérations est instantané, mais des erreurs humaines peuvent survenir. Le risque d'introduire de telles erreurs et ainsi conduire à des mauvaises détections est important. Plutôt qu'une intégration directe de la détection des évènements dans le code des opérations, nous pouvons imaginer qu'une analyse automatique du code, *a posteriori*, permettrait de détecter automatiquement les évènements, mais cela serait certainement particulièrement difficile.

Au delà des difficultés liées à la détection et au suivi des évènements, les schémas de nomination persistante de la littérature posent un autre problème. Ces schémas sont majoritairement basés sur un type d'entité topologique spécifique, le plus souvent des cellules de type face, rendant difficile la généralisation des méthodes liées à la réévaluation. Dans le cas des faces par exemple, cela rend particulièrement difficile la caractérisation et le suivi des volumes et autres entités de plus hautes dimensions. De fait, aucune des solutions proposées pour la nomination ne définit de schéma de nomination réellement générique. Ainsi, à notre connaissance, chaque modeleur ne permet la construction et la réévaluation que pour un ensemble défini de dimensions.

Enfin, concernant l'édition des spécifications paramétriques, la grande majorité des approches proposées restreignent l'étude à la modification des paramètres géométriques. Or, l'ajout, la suppression ou encore le déplacement d'opérations impactent fortement le processus de réévaluation et les mécanismes de nomination persistante. Il est donc fondamental d'étudier cette problématique de façon plus globale.

Finalement, aucun de ces systèmes n'offre de garantie sur la cohérence topologique des objets créés. Ainsi, sans ce type de garantie et avec les variations induites par l'édition d'une spécification paramétrique, les modèles réévalués doivent être analysés puis réparés, le cas échéant, afin d'être utilisables [Hor+09 ; Gon+17].

Démarche et contributions

La nomination persistante, dans le cadre de la réévaluation de modèles paramétriques, est un problème étudié depuis plusieurs décennies. La réévaluation est, à ce jour, devenu une fonctionnalité incontournable dans les modeleurs paramétriques tels que Revit [Autd], SolidWorks [Dasb] et bien d'autres. Cependant, la réévaluation propose, en fonction des modeleurs, des résultats variables ne respectant pas toujours les intentions de conceptions des utilisateurs.

Pour aborder ces problématiques, nous nous basons sur les travaux de Cardot [Car19]. Ses travaux proposent en effet d'utiliser le formalisme des cartes généralisées et des règles Jerboa pour garantir la cohérence topologique des modèles ainsi que la généricité des procédures pour la réévaluation. Ainsi, nous présentons dans ce manuscrit un modeleur paramétrique répondant à ces besoins et pour lequel nous avons :

- formalisé la détection de évènements à partir de l'analyse syntaxique des règles Jerboa ;
- implanté un mécanisme de nomination persistante à deux niveaux :
  - le premier niveau identifie une entité topologique par les opérations qui impactent son évolution,
  - le second niveau réutilise la formalisation des changements topologiques pour reconstituer, à l'aide de graphes orientés acycliques (DAG), un historique enrichi de ses évolutions.

— implanté une procédure d'appariement des entités topologiques pour le rejeu ;
— proposé des stratégies permettant de configurer la réévaluation en fonction des préférences des utilisateurs ;
— proposé une extension des structures de données existantes pour intégrer le support de scripts à notre mécanisme de réévaluation ;
— implanté une procédure d'appariement d'entités topologiques entre deux règles alternatives pour le rejeu.

Organisation du manuscrit

Ce manuscrit détaille en cinq chapitres les différents aspects de cette thèse.

Le premier chapitre concerne les outils formels pour la modélisation à base de règles. Nous commençons par des rappels sur les graphes et leurs morphismes pour ensuite aborder leurs transformations. Nous y présentons ensuite deux outils de formalisation qui sont les cartes généralisées et le langage de règles Jerboa. Avec les cartes généralisées pour représenter les objets et le langage de règle Jerboa pour formaliser les opérations, nous présentons de manière formelle l'approche de modélisation à base de règles sur laquelle nous nous appuyons.

Le deuxième chapitre concerne les concepts et notions liés à la modélisation paramétrique. Dans ce chapitre, nous commençons par distinguer les approches de modélisation paramétrique dites *variationnelles* et *fonctionnelles* pour la réévaluation. Nous abordons ensuite le problème de la nomination persistante dans le cadre des systèmes paramétriques fonctionnels. Nous proposons également une classification des différentes méthodes de nomination persistante. Nous passons en revue un ensemble de solutions pour la réévaluation de modèles paramétriques en mettant en exergue leurs avantages et inconvénients. Enfin, nous positionnons nos travaux par rapport à la littérature.

Dans le troisième chapitre, nous présentons notre première contribution sur la formalisation des évènements. Nous proposons tout d'abord de caractériser ces évènements lors de la transformation de cartes généralisées. Nous formalisons ensuite ces évènements dans les règles de transformation de graphes. Nous y définissons notamment la notion d'origine permettant d'enrichir le suivi des évènements. Nous étendons cette formalisation des évènements aux règles Jerboa qui nous servent à représenter les opérations de modélisation.

Dans le quatrième chapitre, nous présentons notre seconde contribution sur la réévaluation dans un système de modélisation à base de règles. Nous décrivons tout d'abord deux cas d'étude qui nous servent d'exemples tout au long du chapitre. À l'aide de ces cas d'étude, nous représentons un système de nomination en deux couches nous permettant de reconstituer l'historique des évolutions d'une entité topologique. Nous y détaillons ensuite notre mécanisme de réévaluation intégré dans un système de modélisation à base de règles. Enfin, nous présentons des stratégies configurables par les utilisateurs afin d'adapter certains mécanismes de réévaluation selon leurs préférences.

Dans le cinquième chapitre, nous étendons notre mécanisme de réévaluation aux scripts de règles. Nous présentons brièvement le langage de scripts Jerboa permettant d'organiser des règles à l'aide des structures de contrôles (itération, alternatives, *etc.*). L'extension en elle-même se concentre sur trois des structures de contrôles usuelles : la séquence, l'itération et l'alternative. Nous présentons ensuite les structures de données requises pour supporter la réévaluation de

scripts présentant des séquences de règles, des itérations sur les entités topologiques, ainsi que des alternatives. En particulier, nous définissons une nouvelle procédure permettant d'apparier les entités topologiques définies dans deux règles alternatives. Enfin, nous présentons l'exemple de la réévaluation d'un script combinant ces différentes structures de contrôle.

Une conclusion générale vient achever la présentation des travaux effectués tout au long de cette thèse. Nous y proposons un récapitulatif des contributions ainsi que des perspectives à court, moyen et long terme.

Publications

[Gai+23a] Maxime Gaide, David Marcheix, Agnès Arnould, Xavier Skapin, Hakim Belhaouari et Stéphane Jean. « Automatic Detection of Topological Changes in Geometric Modeling Operations ». In : *Computer Graphics and Visual Computing*. 2023, p. 9-18.

[Gai+23b] Maxime Gaide, David Marcheix, Agnès Arnould, Xavier Skapin, Hakim Belhaouari et Stéphane Jean. « Model Reevaluation Based on Graph Transformation Rules ». In : *Computer Graphics and Visual Computing*. 2023, p. 61-63.

[Gai+24] Maxime Gaide, David Marcheix, Agnès Arnould, Xavier Skapin, Hakim Belhaouari et Stéphane Jean. « Reevaluation in Rule-Based Graph Transformation Modeling Systems ». In : *WSCG*. 2024, p. 117-128.

CHAPITRE 1

Outils pour la modélisation à base de règles

Sommaire

1 Introduction

En informatique, de nombreuses applications utilisent la création et la restitution d'objets virtuels, notamment dans le domaine de la conception assistée par ordinateur (CAO) et la génération procédurale [Dasa ; Sie ; Ope ; Dasb]. Dans le cas de la CAO, un objet est construit en appliquant une succession de contraintes ou d'opérations de modélisation géométrique, ou bien par édition directe, comme avec la fonction sculpture de Blender [Ble]. En revanche, pour la génération procédurale, un objet est construit suivant un ensemble de règles. Celles-ci peuvent être définies par une grammaire formelle, comme les L-Systems [Lin74], ou par des instructions fondées sur des opérations. Les systèmes de génération procédurale sont utilisés dans diverses applications, notamment dans les jeux vidéo et le cinéma [Auta ; Autc ; God ; Uni ; Epi], ainsi que pour la génération de structures végétales [Ter+09 ; BTG15] et la création d'espaces urbains [ESR]. Nos travaux se concentrent sur la conception d'un modeleur CAO dans lequel les opérations sont définies par un langage de règles de transformation de graphe.

Dans ce chapitre, nous commençons par quelques rappels et définitions concernant les graphes, en particulier les graphes dits « étiquetés », qui occupent une place centrale dans nos travaux (section 2). Ensuite, nous présentons, dans la section 3, le modèle de représentation des cartes généralisées [Lie94 ; DL14], qui permet de formaliser non seulement un objet géométrique, mais aussi ses entités topologiques à travers l'usage de graphes. Enfin, dans la section 4, nous introduisons le langage *Jerboa* [Bel+14], conçu pour formaliser les opérations de modélisation géométrique au moyen de règles de transformation de graphes. Avec ces formalismes, nous disposerons des outils nécessaires pour introduire les définitions et concepts fondamentaux qui sous-tendent nos travaux tout au long de ce manuscrit.

2 Graphes et transformations de graphes

Afin de formaliser la représentation et la transformation d'objets issus d'un processus de modélisation géométrique, nous nous intéressons à leur représentation sous la forme de *graphes topologiques*. Les graphes dits topologiques appartiennent à la catégorie des *graphes étiquetés*, c'est-à-dire des graphes dont chaque nœud est porteur d'une information de dimension. .

Définition 1 (Graphe étiqueté)
Soient deux ensembles d'étiquetage, $\mathcal{C}_V$ pour les nœuds et $\mathcal{C}_E$ pour les arcs.
Un graphe étiqueté $G = (V, E, s, t, l_V, l_E)$ *est défini par :*
- *un ensemble V de* nœuds *;*
- *un ensemble E d'*arcs *;*
- *deux fonctions* source $s : E \to V$ *et* cible $t : E \to V$ *qui lient les arcs aux nœuds ;*
- *deux* fonctions d'étiquetage $l_V : V \to \mathcal{C}_V$ *et* $l_E : E \to \mathcal{C}_E$ *qui étiquettent respectivement les nœuds et les arcs.*

Soient un nœud $v \in V$ et un arc $e \in E$; $s(e)$, $t(e)$ sont appelés respectivement le nœud source et le nœud cible de e et $l_V(v)$ et $l_E(e)$ sont respectivement les étiquettes de v et e.

Prenons, par exemple, le graphe étiqueté représenté dans la figure 1.1. Pour ce graphe, les ensembles d'étiquettes pour les nœuds $\mathcal{C}_V$ et pour les arcs $\mathcal{C}_E$ sont définis sur le même ensemble de couleurs $\{orange, vert, bleu, violet\}$. L'ensemble V des nœuds est constitué des nœuds a, b, c et d et l'ensemble E des arcs est constitué des arcs e_1, e_2, e_3 et e_4. Ce graphe est construit de

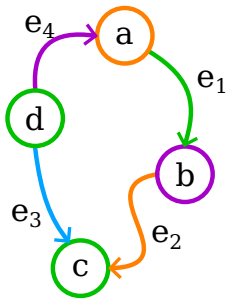


FIGURE 1.1 – Graphe dont les nœuds et arêtes sont étiquetés par des couleurs

telle sorte que e_1 a pour source et cible les nœuds a et b, respectivement. Enfin, les nœuds a et b sont respectivement étiquetés par les couleurs *orange* et *violet* et l'arc e_1 est étiqueté par la couleur *vert*.

Définition 2 (Morphisme)
Soient $G = (V, E, s, t, l_V, l_E)$ *et* $G' = (V', E', s', t', l'_V, l'_E)$ *deux graphes étiquetés.*
Un morphisme $m : G \to G'$ *est un couple de fonctions* $m_V : V \to V'$, $m_E : E \to E'$ *préservant les sources, les cibles et les étiquettes de G dans G', c'est-à-dire pour tout arc* $e \in E$, $m_V(s(e)) = s'(m_E(e))$, $m_V(t(e)) = t'(m_E(e))$, $l_V(v) = l'_V(m_V(v))$ *et* $l_E(e) = l'_E(m_E(e))$.
m *est :*

- injectif *si ses fonctions* m_V *et* m_E *sont injectives* [1],
- *un* isomorphisme *si ses fonctions* m_V *et* m_E *sont des bijections* [2],
- *une* inclusion, *noté* $\iota : G \hookrightarrow G'$ *ou simplement* $G \hookrightarrow G'$, *si* ι_V *et* ι_E *sont les applications identités.*

La figure 1.2 illustre chacun des morphismes présentés dans la définition 2 :

Morphisme dans la figure 1.2a par la fonction ma_V, pour les nœuds issus de G, nous avons $a \mapsto e$, $b \mapsto f$, $c \mapsto j$ et $d \mapsto j$ dans G'. Par la fonction ma_E, pour les arcs issus de G, nous avons $e_1 \mapsto e_5$, $e_2 \mapsto e_6$, $e_3 \mapsto e_7$ et $e_4 \mapsto e_8$ dans G'.

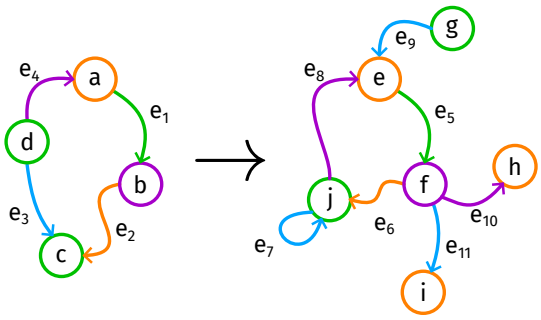
Morphisme injectif Dans la figure 1.2b, les nœuds e, f, g, h sont respectivement les images de a, b, c, d et tout arc (respectivement nœud) de G' est l'image d'un arc (respectivement nœud) de G.

Isomorphisme Dans la figure 1.2c, les arcs et nœuds de G sont identiques à ceux de G' au renommage près.

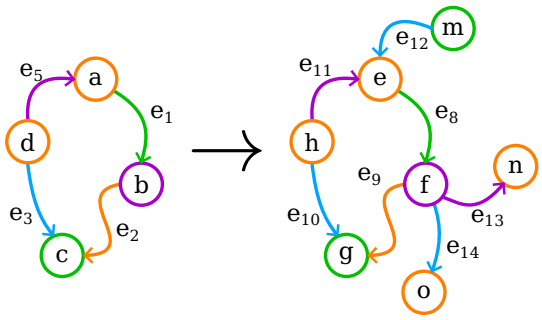
Morphisme d'inclusion dans la figure 1.2d, les applications sont des identités sans renommage.

À l'aide de la notion de morphisme, il est possible d'étendre les principales notions ensemblistes aux graphes. Ces notions ensemblistes sont, entre autres, l'union, l'intersection ou encore la différence entre deux graphes.

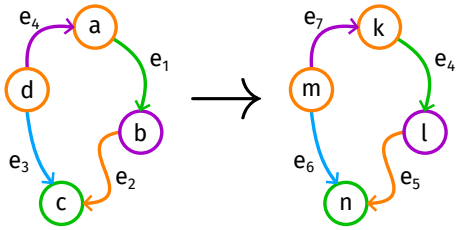
1. Pour rappel, une fonction $f : A \to B$ est injective si et seulement si tout élément $b \in B$ a au plus un antécédent par f, c'est-à-dire $\forall a, a' \in A^2$, si $f(a) = f(a')$ alors $a = a'$.
2. Pour rappel, une fonction $f : A \to B$ est bijective si et seulement si tout élément de $b \in B$ a exactement un antécédent par f, c'est-à-dire $\forall b \in B$, $\exists!\ a \in A$ tel que $f(a) = b$.



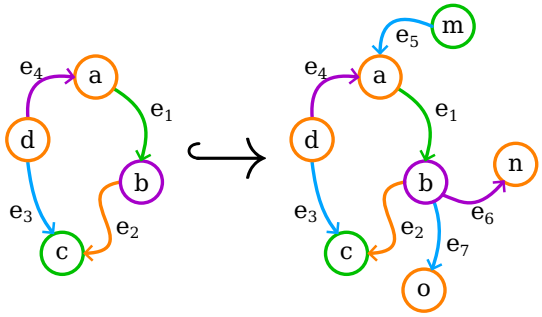
(a) Morphisme $ma : G \to G'$



(b) Morphisme injectif $mb : G \to G'$



(c) Isomorphisme $mc : G \to G'$



(d) Morphisme d'inclusion $\iota : G \to G'$

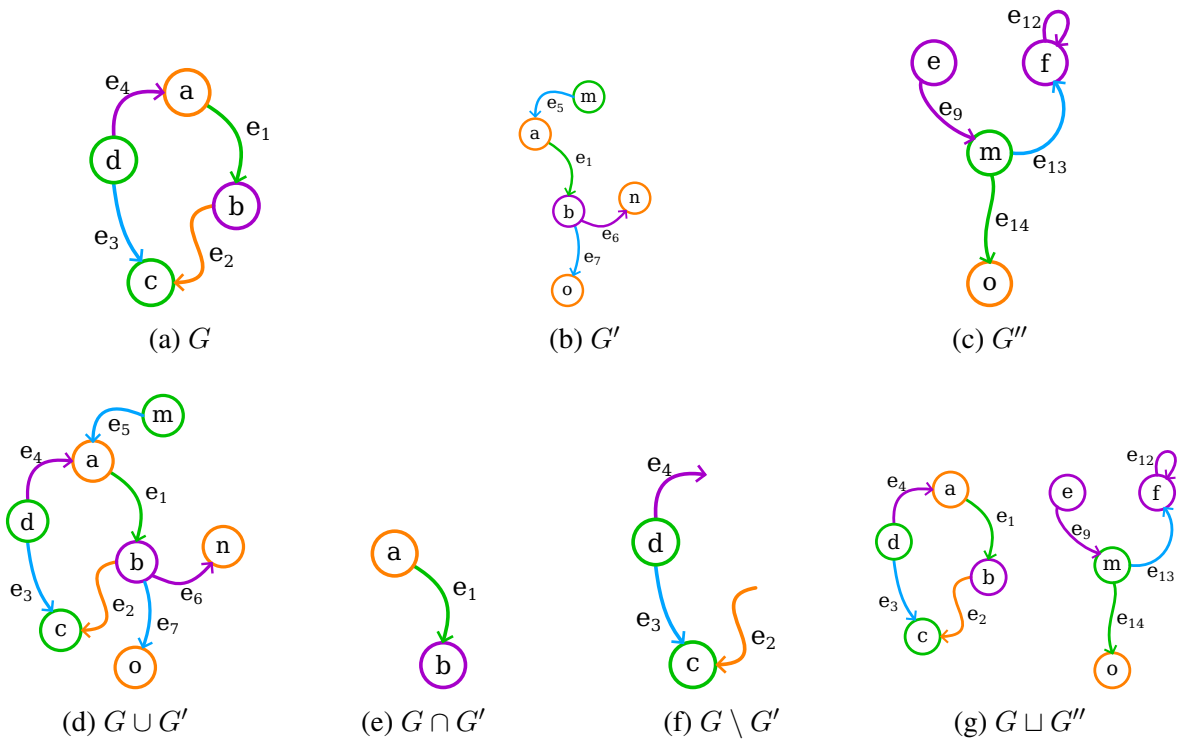
Figure 1.2 – Morphismes $G \to G'$

Définition 3 (Inclusion, union (disjointe), intersection, différence)
Soient $G = (V, E, s, t, l_V, l_E)$ et $G' = (V', E', s', t', l'_V, l'_E)$ deux graphes étiquetés.

- *G est* inclus *dans G', si et seulement si il existe un morphisme d'inclusion $G \hookrightarrow G'$.*
- *L'*intersection *$G \cap G'$, quand elle existe, est le graphe K dont les nœuds sont $V \cap V'$ et les arcs sont $E \cap E'$, tel que $G \hookleftarrow K \hookrightarrow G'$.*
- *L'*union *$G \cup G'$, quand elle existe, est le graphe J sont les nœuds sont $V \cup V'$ et les arcs sont $E \cup E'$, tel que $G \hookrightarrow J \hookleftarrow G'$.*
- *L'*union disjointe *$G \sqcup G'$ est l'union de deux graphes disjoints G et G' (*c'est-à-dire *tels que $V \cap V' = \emptyset$ et $E \cap E' = \emptyset$).*
- *La* différence *$G \setminus G'$ est la structure $(V \setminus V', E \setminus E', s_{|E \setminus E'}, t_{|E \setminus E'}, l_{V_{|V \setminus V'}}, l_{E_{|E \setminus E'}})$.*
- *Soit $\emptyset$ le* graphe vide*, c'est-à-dire l'unique graphe ayant un ensemble vide de nœuds et d'arcs.*

De tels graphes, issus de l'union ou de l'intersection de deux graphes distincts sont présentés dans la figure 1.3. Dans le cas de l'union de G avec G' (figure 1.3d), les nœuds a et b de G sont associés respectivement aux nœuds a et b de G' et les nœuds m, n, o de G', sont ajoutés avec leurs arcs e_5, e_6 et e_7 dans $G \cup G'$. Le graphe $G \cap G'$ (figure 1.3e) est constitué uniquement des nœuds a, b et de l'arc e_1 de G, car ils sont associés, par leurs étiquettes, aux nœuds a, b et l'arc e_1, respectivement, de G'.

La différence entre deux graphes peut produire une structure qui n'est pas un graphe. Par exemple, la structure illustrée dans la figure 1.3f présente deux arcs e_2 et e_4 dits *pendants*, c'est-à-dire des arcs auxquels il manque au moins un nœud source et/ou cible. Le graphe $G \sqcup G''$ (figure 1.3g) est constitué de l'union G et G' privée de leur intersection. Dans ce cas-ci, G et G'' sont tous deux entièrement compris dans l'union disjointe puisque leur intersection est vide. Ces propriétés permettent, à leur tour, de définir la notion de sous-graphe.



(a) G (b) G' (c) G''

(d) $G \cup G'$ (e) $G \cap G'$ (f) $G \setminus G'$ (g) $G \sqcup G''$

FIGURE 1.3 – Opérations ensemblistes sur les graphes

Définition 4 (Sous-graphe)
Soient deux graphes $G = (V, E, s, t, l_V, l_E)$ *et* $G' = (V', E', s', t', l'_V, l'_E)$, ω *un mot sur* l_E *et* v *un nœud de* V.
G' *est un* sous-graphe *de* G *si et seulement si,* $G' \subset G$. *On note* $G\langle\omega\rangle(v)$ *le plus petit sous-graphe qui contient le nœud* v, *tous les nœuds atteignables par des arcs étiquetés par des lettres de* ω *et ces arcs eux-mêmes.*

(a) Graphe G (b) G'

FIGURE 1.4 – Génération du sous-graphe $G' = G\langle vert, orange\rangle(a)$

Dans la figure 1.4, nous définissons G' comme un sous-graphe de G généré à partir du nœud a et des étiquettes d'arcs $\langle vert, orange\rangle$. Le sous-graphe illustré dans la figure 1.4b est généré à partir du nœud a. Les nœuds b et c sont atteints au travers des arcs e_1 et e_2 respectivement à travers $vert$ et $orange$. En fin de compte, G' comporte les nœuds a, b, c et les arcs e_1, e_2.

Un dernier concept essentiel pour l'utilisation des graphes et celui de chemin. Intuitivement, un chemin est une suite d'arcs consécutifs.

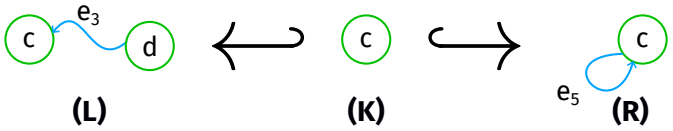
Définition 5 (Chemin et cycle)
Un chemin $e_1 e_2 \dots e_k$ *dans un graphe* G *est une suite d'arcs tels que pour* $1 \leq i < k$, $t(e_i) = s(e_{i+1})$. *La source d'un tel chemin est* $s(e_1)$, *sa cible est* $t(e_k)$ *et son étiquette est le mot* $\alpha(e_1)\alpha(e_2)\dots\alpha(e_k)$.
Un cycle *est un chemin* $e_1 e_2 \dots e_k$ *pour lequel la source est également la cible, i.e.* $s(e_1) = t(e_k)$.

(a) Graphe G avec deux chemins depuis d vers c (b) Graphe G' avec un cycle

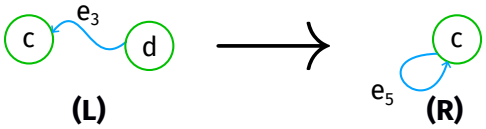
Figure 1.5 – Chemin et cycle dans un graphe

Dans la figure 1.5, nous illustrons à partir de G (figure 1.5a) et G' (figure 1.5b) ce que sont un chemin et un cycle. Le graphe G présente plusieurs chemins, par exemple e_1, e_2, dans lequel les nœuds a et b sont la source et la cible, respectivement. Ce chemin est lui-même compris dans le chemin e_4, e_1, e_2 dont le nœud d est la source. Le graphe G' contient la boucle e_1, e_2, e_3, e_4 dont le nœud a est à la fois la source et la cible.

Définition 6 (Règle de transformation de graphe)
Soient deux graphes étiquetés L *et* R *tels que l'intersection* $L \cap R = K$ *existe.*
$r : L \longrightarrow R$ *est la* règle de transformation de graphe *définie par le double morphisme d'inclusion* $L \hookleftarrow K \hookrightarrow R$. L *est appelé le membre gauche, ou motif gauche,* R *le membre droit, ou motif droit, et* K *l'interface.*



(a) Règle avec interface explicite $r : L \hookleftarrow K \hookrightarrow R$



(b) Règle $r : L \longrightarrow R$

Figure 1.6 – Règle de transformation de graphe r

Une telle règle est illustrée dans la figure 1.6 où le membre gauche (L) filtre tout ou partie d'un graphe et le membre droit (R) transforme l'élément filtré par le membre gauche. Une manière simple d'assurer que l'intersection $L \cap R$ (ou K) est un graphe est que L et R n'aient pas d'arête commune.

L'application d'une règle de transformation de graphe peut être définie formellement dans la théorie des catégories [Ehr+06]. Dans ce manuscrit, nous en donnons une définition ensembliste équivalente. L'application de la règle $r : L \longrightarrow R$ sur un graphe G le long d'un morphisme de

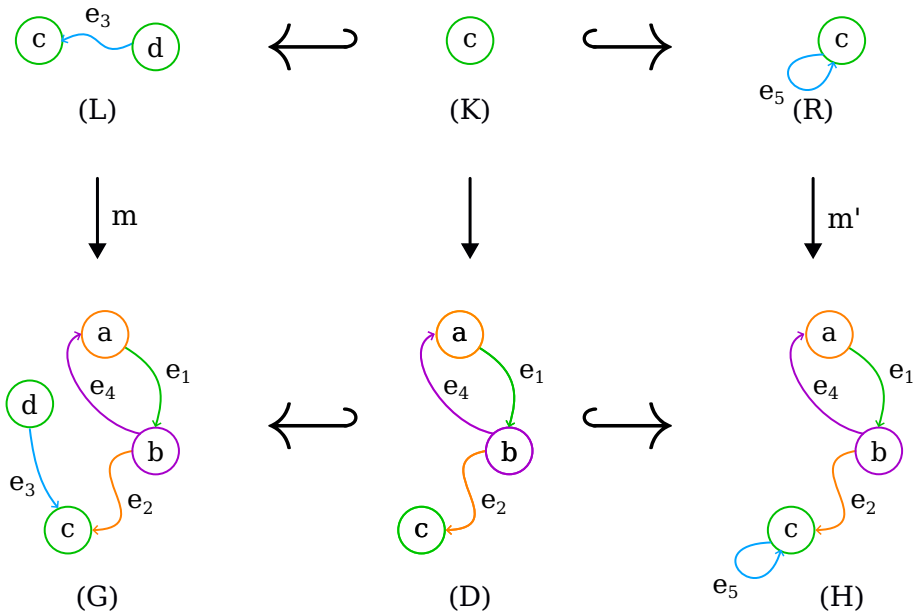


FIGURE 1.7 – Application d'une règle de transformation de graphe

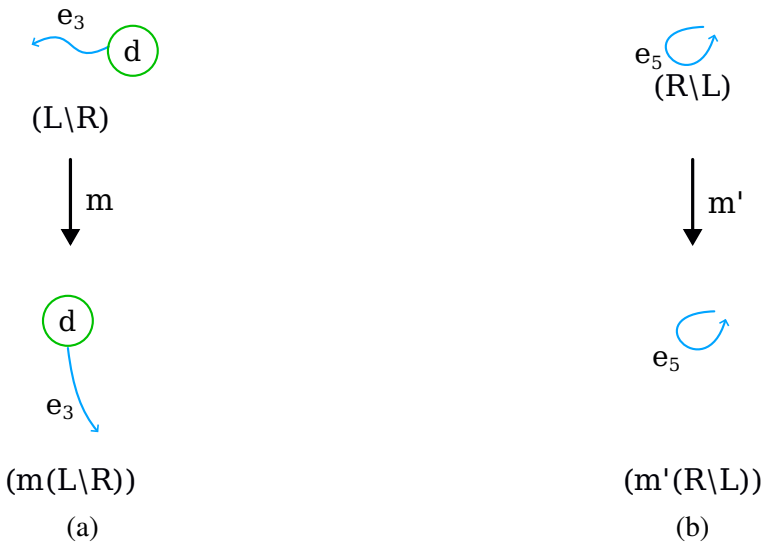


FIGURE 1.8 – Morphismes : (a) $m(L \setminus R)$ et (b) $m'(R \setminus L)$

filtrage $m : L \to G$ produit un graphe transformé H, où m' est un morphisme injectif qui étend m sur R.

Définition 7 (Application d'une règle de transformation de graphe)
Soient $r : L \longrightarrow R$ une règle d'interface K et $m : L \to G$ un morphisme injectif appelé morphisme de filtrage.

*L'*application $G \xrightarrow{(r,m)} H$ *de r sur G le long de m, lorsqu'elle existe, est définie par le double morphisme d'inclusion $G \hookleftarrow D \hookrightarrow H$, où $D = (G \setminus m(L \setminus K))$ et $H = D \sqcup m'(R \setminus K)$, avec m' une restriction de m sur K, puis son extension sur R telle que m' soit injective. m', et donc H, est unique à isomorphisme près. H est la transformation directe de G par la règle r le long du morphisme de filtrage m.*

Intuitivement, $m(L \setminus R)$ est supprimé dans G, puis $m'(R \setminus L)$ est ajouté pour produire la

transformation directe H. Prenons l'exemple de l'application illustrée figure 1.7 ainsi que les morphismes présentés sur les figures 1.8a et 1.8b. La différence de G et $m(L \setminus R)$ donne le graphe D, puis l'union de D avec $m'(R \setminus L)$ donne le graphe H. Le morphisme m' est construit de telle manière que la partie ajoutée est disjointe de G, *c'est-à-dire* que les nouveaux nœuds et arcs ont des noms « frais » (disjoints de ceux de G). Donc, m' et H sont définis à isomorphisme près.

Remarque : *La transformation directe H existe à la condition que D existe, c'est-à-dire que pour tout arc e de G, si e relie un nœud v filtré (qui appartient à $m(L)$) et un nœud v' non filtré (qui n'appartient pas $m(L)$), alors v est filtré par l'interface (qui appartient à $m(K)$). Dans la littérature, cette condition s'appelle la condition d'arc pendant [Ehr+06] et est illustrée figure 1.3f. Or, les travaux présentés dans ce manuscrit s'appuient sur des formalismes ne permettant pas de travailler sur des structures présentant des arcs pendants.*

Dans la figure 1.7 qui illustre l'application d'une règle de transformation de graphe, les morphismes m et m' sont des inclusions, mais en pratique ce n'est pas le cas et la partie ajoutée dans H est ajoutée avec des noms frais. Dans la transformation directe présentée, le nœud d et l'arc e_3 sont supprimés et remplacés par la boucle e_5.

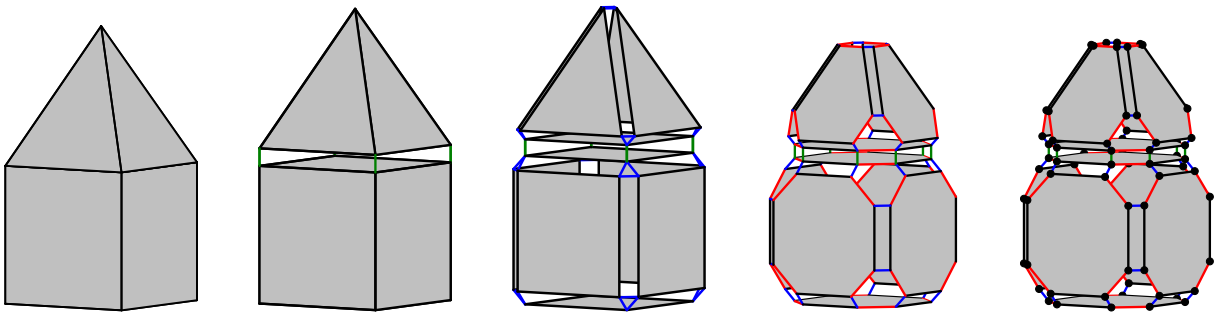
3 Les cartes généralisées

Les *cartes généralisées* (ou « G-cartes ») [Lie91 ; DL14] permettent la représentation de variétés géométriques (avec ou sans bord) basées sur des structures topologiques cellulaires de dimension quelconque.

Définition 8 (G-carte)
Un graphe topologique *de dimension $n \in \mathbb{N}$ est un graphe étiqueté $G = (V, E, s, t, \alpha)$, où les nœuds sont appelés* brins*, les arcs sont appelés* liaisons et $\alpha : E \to [0, n]$ est la fonction d'étiquetage des liaisons.
Une *G-carte* est un graphe topologique qui vérifie les contraintes suivantes :

- *non-orientation* : pour tout arc $e \in E$, il existe un arc inverse $e' \in E$ tel que $s(e') = t(e)$, $t(e') = s(e)$ et $\alpha(e') = \alpha(e)$;
- *arcs incidents* : pour toute dimension $i \in [0, n]$ et pour tout nœud $v \in V$, il existe exactement un arc $e \in E$ d'étiquette $\alpha(e) = i$ et de source $s(e) = v$;
- *cycle* : pour toutes dimensions $i, j \in [0, n]^2$ telles que $i + 2 \leq j$, et pour tout nœud $v \in V$, il existe un cycle, étiqueté par $ijij$, de source v.

Les G-cartes permettent de représenter les objets quasi-variétés par leurs décompositions successives selon les dimensions, comme illustré dans la figure 1.9. Un objet (figure 1.9a) peut être décomposé en volumes connectés par des *3-liaisons*, en vert (figure 1.9b). Ces volumes peuvent se décomposer en faces, connectées par des *2-liaisons*, en bleu (figure 1.9c). Les faces se décomposent en arêtes connectées par des *1-liaisons*, en rouge (figure 1.9d). Enfin, ces arêtes se décomposent en sommets connectés par des *0-liaisons*, en noir (figure 1.9e). Le graphe ainsi obtenu figure 1.9e est une G-carte de dimension 3. Notons que, en bordure de l'objet, les liaisons bouclent sur elles-mêmes. Par soucis de visibilité, nous masquons ces boucles dans les illustrations.



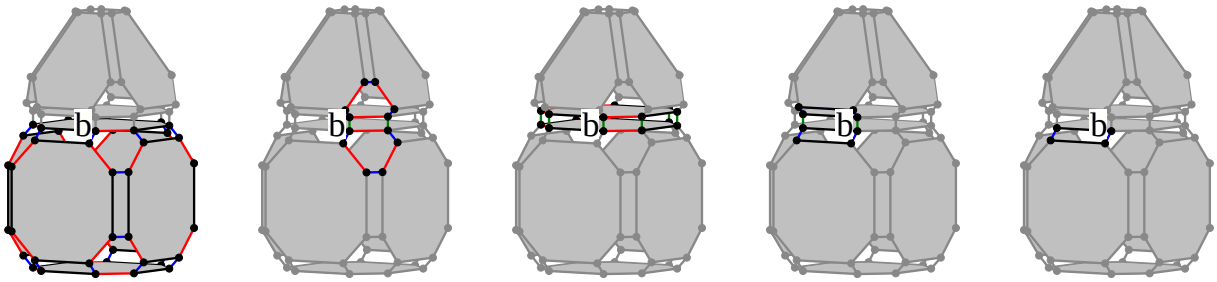
(a) Objet (b) 3-liaisons (—) (c) 2-liaisons (—) (d) 1-liaisons (—) (e) 0-liaisons (—)

FIGURE 1.9 – Décompositions d'un objet par dimensions successives

Remarque : *On a défini ici la structure topologique, sans étiqueter les brins. Mais en pratique, les objets sont représentés par des structures topologiques plongées où les étiquettes des brins représentent leurs propriétés géométriques et physiques.*

Les G-cartes représentent les cellules et plus généralement les orbites à l'aide de sous-graphes.

Définition 9 (Orbite)
Soient G une G-carte de dimension $n \in \mathbb{N}$, $\langle\omega\rangle$ un mot de $[0, n]$ et b un brin de G.
*L'*orbite *incidente à b et de type $\langle\omega\rangle$, où $\langle\omega\rangle$-orbite, est le sous-graphe de $G\langle\omega\rangle(b)$. Lorsqu'il n'y a pas d'ambiguïté sur le graphe G considéré, on notera une telle orbite $\langle\omega\rangle(b)$.*



(a) Volume (b) Sommet (c) Face (d) Arête (e) Arête de volume

FIGURE 1.10 – Orbites dans une G-carte

Par exemple, la figure 1.10 illustre un panel de cellules présentes dans une G-carte de dimension 3. On retrouve, dans la figure 1.10a, le volume incident au brin b, noté $\langle 0, 1, 2\rangle(b)$, contenant le brin b, les brins accessibles à partir de b par les liaisons $0, 1$ et 2 ainsi que ces liaisons. Le sommet (figure 1.10b) est noté $\langle 0, 1, 2\rangle(b)$, la face (figure 1.10c) $\langle 0, 1, 3\rangle(b)$ et l'arête (figure 1.10d) $\langle 0, 2, 3\rangle(b)$. Notons qu'il est également possible de représenter des entités topologiques plus générales que des cellules. Par exemple, l'arête de volume dans la figure 1.10e contient les brins accessibles par des liaisons 0 et 2.

4 Le langage Jerboa

Dans la section précédente, nous avons défini les cartes généralisées comme des graphes étiquetés vérifiant certaines contraintes de cohérence. Les opérations sur ces graphes peuvent être définies par des règles de transformation de graphes. Dans cette section, nous abordons la formalisation des opérations de modélisation sous la forme de règles de transformation de graphe. Tout d'abord, nous présentons les règles de transformation de graphe et les concepts nécessaires à leur mise en œuvre. Ensuite, nous étendons ces concepts aux règles de transformations de graphes Jerboa [Bel+14 ; Arn+22 ; Pas+22].

4.1 Règles de transformation de graphe

Les règles de transformation de graphe (définition 6) permettent de construire des objets représentés par des G-cartes. Des conditions syntaxiques sur les règles définies dans [Pou09 ; Pas+22] garantissent la préservation des G-cartes.

Théorème 1 (Préservation de la cohérence topologique)
Soit $r : L \longrightarrow R$ une règle de transformation de graphe topologique de dimension $n \in \mathbb{N}$. Si r vérifie les propriétés syntaxiques suivantes, alors pour tout morphisme de filtrage $m : L \to G$ d'une G-carte G, la transformation directe $G \xrightarrow{r,m} H$ produit une G-carte H :

- Condition de non-orientation *: L et R sont* non-orientés.
- Condition d'arcs incidents *: pour toute dimension $i \in [0, n]$,*
  - *un nœud préservé de $L \cap R$ est la source d'un arc étiqueté i dans L si et seulement s'il est la source d'un arc étiqueté i dans R ;*
  - *tout nœud supprimé de $L \setminus R$ et tout nœud créé de $R \setminus L$ est la source d'un unique arc étiqueté i.*
- Conditions de cycles *: pour tout couple (i, j) tel que $0 \leq i \leq i + 2 \leq j \leq n$,*
  - *un nœud créé de $R \setminus L$ est la source d'un cycle étiqueté $ijij$;*
  - *si un nœud préservé de $L \cap R$ est source d'un cycle étiqueté $ijij$ dans R, il est source d'un cycle étiqueté $ijij$ dans L ;*
  - *si un nœud préservé de $L \cap R$ n'est pas la source d'un cycle étiqueté $ijij$ dans L, alors les arcs d'étiquettes i et j, dont il est la source, sont préservés dans R.*

Remarque : *Dans la suite, nous utiliserons uniquement des règles qui vérifient ces conditions de préservation des G-cartes. Ainsi, lorsque G est une G-carte, sa transformation directe H l'est aussi. Notons que d'autres conditions permettent de préserver la cohérence géométrique [BAL11 ; Arn+22].*

En pratique, les morphismes de filtrage ne sont pas donnés *in extenso*, mais à partir de l'association d'un sous-ensemble de brins de L appelés brins d'accroche (*hooks* en anglais) à leur image dans G. Par exemple, dans la figure 1.11, le nœud b de L est son brin d'accroche (en surbrillance en orange). Son association au brin b de G définit un unique morphisme de filtrage m grâce à la condition d'arcs incidents. Pour cela, chaque règle comporte un brin d'accroche par composante connexe de L. Grâce à la condition d'arcs incidents, nous pouvons déterminer si une orbite de la règle filtre entièrement, ou non, l'orbite correspondante dans l'objet et ce, indépendamment de l'objet.

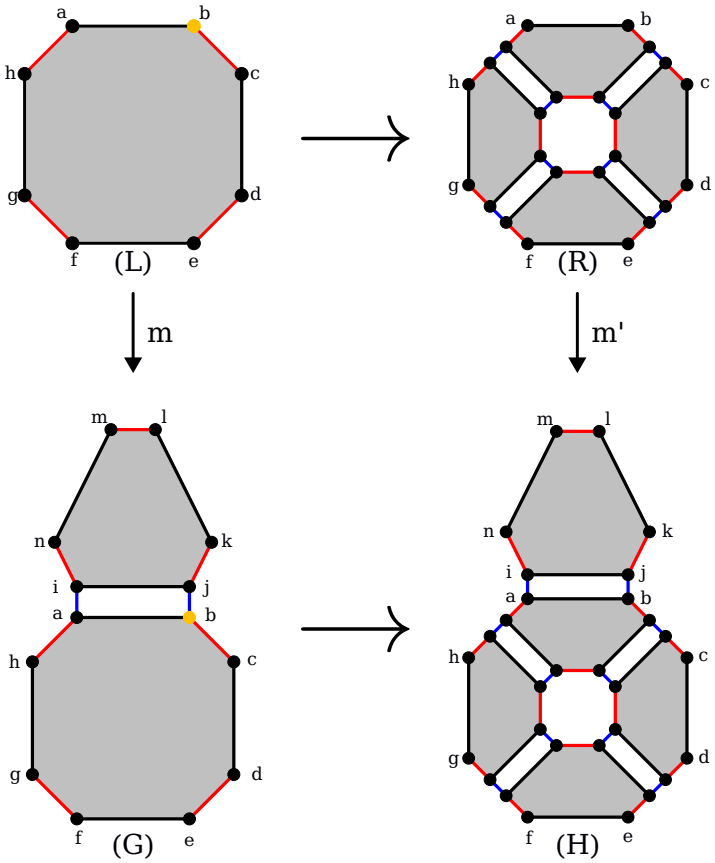


FIGURE 1.11 – Application de la règle de triangulation d'une face carrée

Définition 10 (Orbite complète et entièrement filtrée)
Soient G une G-carte, $m : L \to G$ un morphisme de filtrage, n un nœud de L et $v = m(n)$ le brin filtré par n dans G.
L'orbite $G\langle\omega\rangle(v)$ est entièrement filtrée *par m si $G\langle\omega\rangle(v) \subset m(L)$,* non filtrée *par m si $G\langle\omega\rangle(v) \cap m(L) = \emptyset$, ou* partiellement filtrée *par m dans les autres cas.*
L'orbite $L\langle\omega\rangle(n)$ est complète *si pour tout nœud n' de $L\langle\omega\rangle(n)$ et toute étiquette i de $\langle\omega\rangle$, n' est la source d'un i-arc.*

Remarque : *Grâce à la condition d'arc incident, l'orbite $L\langle\omega\rangle(n)$ est complète si et seulement si $G\langle\omega\rangle(m(n))$ est entièrement filtrée.*

4.2 Règles Jerboa

Comme pour les règles de base, les conditions syntaxiques suivantes sur les règles permettent de garantir la préservation des contraintes des G-cartes lors des transformations directes [Pas+22].

Théorème 2 (Préservation de la cohérence topologique des règles Jerboa)
Soit une règle $r : L \longrightarrow R$, G une G-carte et $\underline{m} : \underline{L} \to G$ un morphisme de filtrage.
La transformation directe $G \xrightarrow{r,\underline{m}} H$ produit une G-carte si les conditions de préservation de la cohérence suivantes sont vérifiées :
— condition de non orientation *: L et R sont non orientés ;*

— condition d'arcs incidents *: pour toute dimension $i \in [0, n]$, la condition d'arcs incidents est celle des règles de base en considérant simultanément les arcs implicites et explicites :*
  — *un nœud préservé de $L \cap R$ est source d'un i-arc dans L (implicite ou explicite) si et seulement s'il est source d'un i-arc dans R (implicite ou explicite) ;*
  — *tout nœud supprimé de $L \setminus R$ et tout nœud ajouté de $R \setminus L$ est la source d'exactement un i-arc (implicite ou explicite) ;*

  où pour toute dimension i, un nœud v est source d'un arc implicite d'étiquette i si l'étiquette d'orbite de v comprend i ;

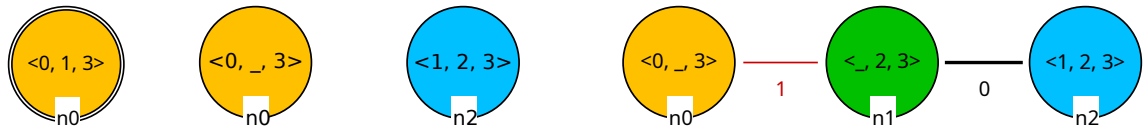
— condition de cycle *: pour toutes dimensions i et j telles que $i + 2 \geq j$, la condition de cycle est celle des règles de base en considérant les cycles ayant des arcs implicites ou explicites :*
  — *un nœud créé v de $R \setminus L$ est la source d'un $ijij$-cycle ;*
  — *si un nœud préservé de $L \cap R$ est la source d'un $ijij$-cycle dans L, alors il est la source d'un $ijij$-cycle dans R ;*
  — *si un nœud préservé de $L \cap R$ n'est pas la source d'un $ijij$-cycle dans L, alors les i-arcs et j-arcs dont il est la source dans L sont préservés dans R.*

  où un nœud v est la source d'un $ijij$-cycle avec $i + 2 \geq j$ si :
  — *v est un nœud de L qui contient i et j dans son étiquette d'orbite ;*
  — *v est un nœud de R qui contient i et j dans son étiquette d'orbite et que ces dernières renomment un cycle de L, c'est-à-dire telles qu'il existe un nœud v' de L avec $l(v) \mapsto l(v')(i) + 2 \geq l(v) \mapsto l(v')(j)$ ou $l(v) \mapsto l(v')(j) + 2 \geq l(v) \mapsto l(v')(i)$;*
  — *v est la source d'un i-arc explicite de cible v' (dans L ou R) tel que les deux nœuds v et v' ont j non renommé dans leurs étiquettes d'orbite, c'est-à-dire tel que $l(v) \mapsto l(v')(j) = j$;*
  — *inversement, v est la source d'un j-arc de cible v' tel que les deux nœuds v et v' ont i non renommé dans leurs étiquettes d'orbites ;*

La règle que nous avons vue plus tôt (figure 1.11) est dite *instanciée*, c'est-à-dire que ses membres représentent des parties transformées des objets. Ainsi, les règles instanciées déterminent complètement la topologie des parties modifiées. Le langage de règles Jerboa [Bel+14], avec la paramétrisation des motifs par des orbites, permet de concevoir des règles plus générales permettant, par exemple, de filtrer puis transformer des faces indépendamment de leur nombre d'arêtes.

Définition 11 (Motif avec orbite)
Un motif avec orbite *est un graphe $M = (V, E, s, t, l, \alpha)$ tel que sa* structure $\underline{M} = (V, E, s, t, \alpha)$ *est un graphe topologique de dimension $n \in \mathbb{N}$ et $l : V \to ([0, n] \cup \{\_\})^*$ est une fonction d'étiquetage des nœuds par des mots sur les dimensions topologiques et le symbole « _ », telle que toutes les étiquettes des nœuds du motif soient des mots de la même longueur.*

Pour décrire ce qu'est un motif avec orbite, nous allons travailler avec les motifs illustrés dans la figure 1.12 où tous les nœuds sont étiquetés avec des orbites de longueur 3. Commençons avec le motif $M1$ (figure 1.12a). Ce motif comporte un unique nœud $n0$ étiqueté par l'orbite face $\langle 0, 1, 3\rangle$, il désigne donc une face. Le motif $M2$ (figure 1.12b), toujours composé du nœud $n0$ mais étiqueté par le mot $\langle 0, \_, 3\rangle$, permet de désigner une face dont les 1-liaisons sont



(a) Face $M1$ (b) Arêtes $M2$ (c) Sommet dual $M3$ (d) Face triangulée $M4$

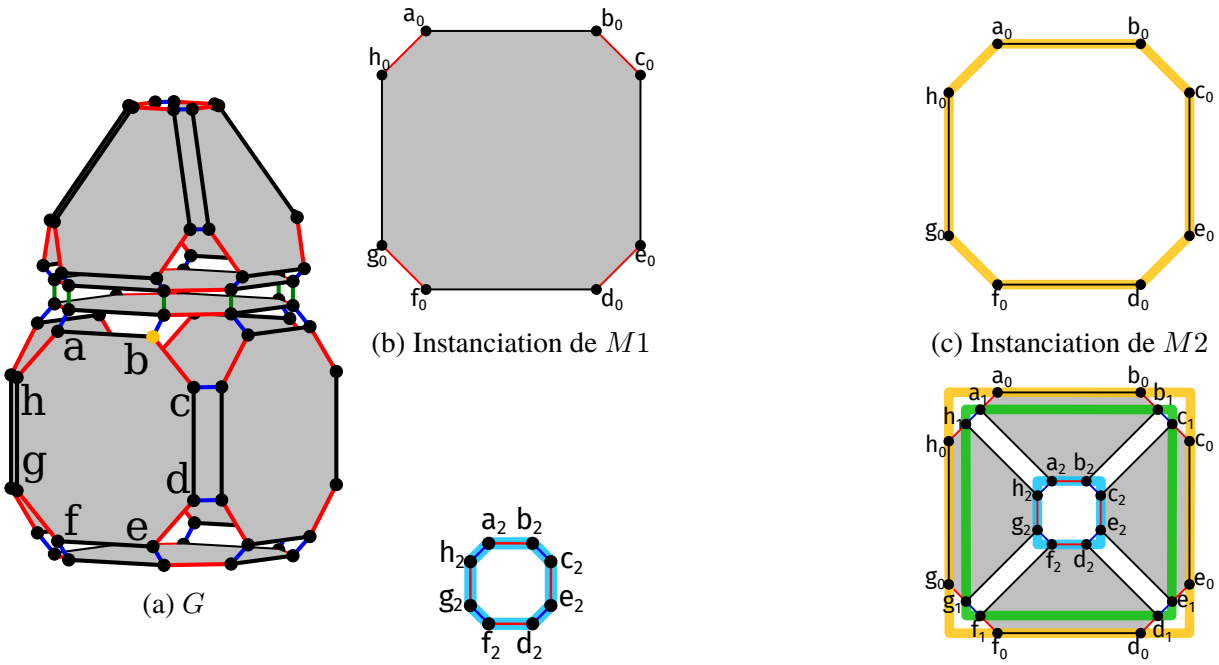
FIGURE 1.12 – Motifs

supprimées, c'est-à-dire les arêtes déconnectées de la face. Passons maintenant au motif $M3$ (figure 1.12c) composé du nœud $n2$ et étiqueté par l'orbite sommet $\langle 1, 2, 3\rangle$. Ce motif permet de construire le sommet dual d'une face. Enfin, le motif $M4$ (figure 1.12d) contient trois nœuds dont $n0$ et $n2$ que nous venons de considérer.

Définition 12 (Renommage)
Soient $n \in \mathbb{N}$ une dimension topologique, ω et ω' deux mots sur $([0, n] \cup \{\_\})^*$.
$\langle\omega\rangle$ est dite une orbite pleine *lorsque ω est un mot sur $[0, n]^*$, c'est-à-dire sans suppression.*
Une fonction de renommage *$\langle\omega\rangle \mapsto \langle\omega'\rangle$, d'une orbite pleine $\langle\omega_1\omega_2...\omega_k\rangle$, vers $\langle\omega'_1\omega'_2 \dots \omega'_k\rangle$ définit le renommage des étiquettes topologiques $\omega_1 \mapsto \omega'_1, \omega_2 \mapsto \omega'_2 \dots \omega_k \mapsto \omega'_k$.*
Le renommage *$(\langle\omega\rangle \mapsto \langle\omega'\rangle)(G\langle\omega\rangle(v))$ d'une $\langle\omega\rangle$-orbite d'un graphe topologique $G = (V, E, s, t, \alpha)$ de dimension $n \in \mathbb{N}$ est le graphe (V, E', s', t', α') défini par :*
- *E' est le sous-ensemble des arcs de E qui n'ont pas été supprimés, c'est-à-dire $E' = \{e \in E \mid (\langle\omega\rangle \mapsto \langle\omega'\rangle)(\alpha(e)) \neq \_\}$,*
- *s' et t' sont les restrictions respectives de s et t sur E',*
- *α' est définie sur E' pour toute arête $e \in E'$ par $\alpha'(e) = (\langle\omega\rangle \mapsto \langle\omega'\rangle)(\alpha(e))$.*

Étudions les motifs illustrés dans la figure 1.12. Par l'observation des étiquettes de $M1$ et $M2$, nous pouvons en déduire que la fonction de renommage $\langle 0, 1, 3\rangle \mapsto \langle 0, \_, 3\rangle$ est définie entre $M1$ et $M2$. Cette fonction permet de supprimer toutes les liaisons 1 (notées par le caractère « _ » en deuxième position) et laisse les liaisons 0 et 3 (respectivement en première et troisième positions) inchangées. De même, l'observation des étiquettes de $M1$ et $M3$ nous permet de déduire que la fonction de renommage $\langle 0, 1, 3\rangle \mapsto \langle 1, 2, 3\rangle$ est définie entre $M1$ et $M3$. Elle permet de renommer les liaisons 0 et 1 respectivement en 1 et 2 (en première et deuxième positions) et laisse inchangées les liaisons 3 (en troisième position).

Définition 13 (Instanciation d'un motif)
Soient G un graphe topologique de dimension $n \in \mathbb{N}$, v un brin de G et $\langle\omega\rangle$ un type d'orbite de dimension n.
*L'*instanciation *d'un motif $M = (V, E, s, t, l, \alpha)$, dont toutes les étiquettes d'orbites ont la même longueur que $\langle\omega\rangle$, par l'orbite $G\langle\omega\rangle(v) = (V', E', s', t', \alpha')$ est définie par le graphe topologique $M(G\langle\omega\rangle(v)) = (V'', E'', s'', t'', \alpha'')$:*
- *$V'' = V \times V'$,*
- *$E'' = E''_i \cup E''_e$ comprend les* arcs implicites *issus de l'instanciation des nœuds du motif $E''_i = \{(v, e') \in V \times E' \mid (\langle\omega\rangle \mapsto \langle\alpha(v)\rangle)(\alpha'(e')) \neq \_\}$ et les* arcs explicites *issus du motif $E''_e = \{(e, v') \in E \times V'\}$,*
- *tout arc implicite $(v, e') \in E''_i$ a pour source $s''((v, e')) = (v, s'(e'))$, pour cible $t''((v, e')) = (v, t'(e'))$ et pour étiquette $\alpha''((v, e')) = (\langle\omega\rangle \mapsto \alpha(v))(\alpha'(e'))$,*
  et tout arc explicite $(e, v') \in E''_e$, a pour source $s''((e, v')) = ((s(e), v')$, pour cible $t''((e, v')) = ((t(e), v')$, et pour étiquette $\alpha''((e, v')) = \alpha(e)$.



(a) G

(b) Instanciation de $M1$

(c) Instanciation de $M2$

(d) Instanciation de $M3$

(e) Instanciation de $M4$

Figure 1.13 – Instanciation des motifs de la figure 1.12 par l'orbite $G\langle 0,1,3\rangle(a)$

La figure 1.13 illustre l'instanciation des motifs de la figure 1.12 par la face incidente au brin b de la G-carte figure 1.13a (la face $G\langle 0,1,3\rangle(b)$). L'instanciation de $M1$ (figure 1.13b) est isomorphe à la face d'instanciation, car le renommage est l'identité. Pour rappel, les liaisons 3 sont toutes des boucles car la face d'instanciation est un bord de la G-carte. Elles ne sont pas représentées dans les figures.

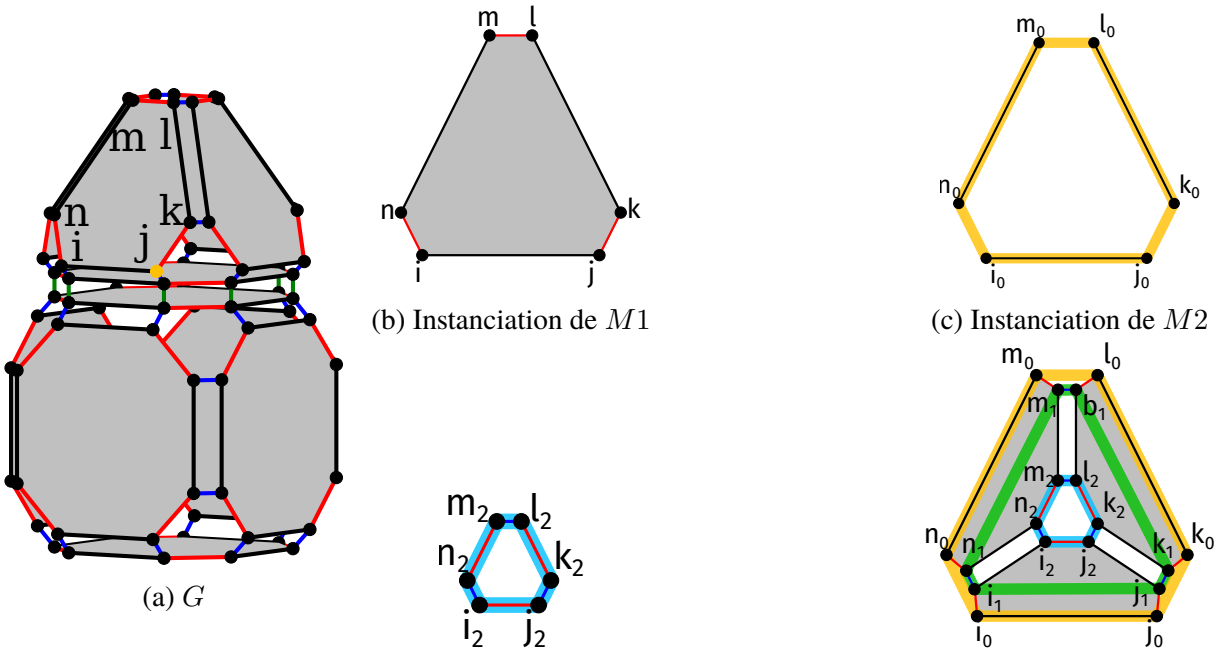
L'instanciation de $M2$ (figure 1.13c) est isomorphe au quatre arêtes de la face d'instanciation, car le renommage supprime les liaisons 1 qui relient les arêtes deux-à-deux. L'instanciation de $M3$ (figure 1.13d) est un sommet dual de la face d'instanciation, car les liaisons 0 et 1 sont renommées respectivement en 1 et 2.

L'instanciation de $M4$ (figure 1.13e) est une triangulation de la face d'instanciation, où les brins $a_0, b_0 \ldots$ en jaune sont les instanciations du nœud $n0$, les brins $a_1, b_1, \ldots$ en vert sont les instanciations du nœud $n1$ et les brins $a_2, b_2, \ldots$ en bleu sont les instanciations du nœud $n2$. L'arc explicite entre $n0$ et $n1$ dans le motif $M4$ produit les 1-liaisons entre les brins jaune et vert et l'arc explicite entre $n1$ et $n2$ produit les 0-liaisons entre les brins vert et bleu.

Les instanciations des motifs $M1, M2, M3$ et $M4$ (figure 1.12) pour la face triangulaire de G incidente au brin j présentées dans la figure 1.14 se déroulent de la même manière pour la face carrée (figure 1.13).

Définition 14 (Règle transformation de graphe Jerboa)
Une règle *Jerboa* $r : L \longrightarrow R$ *est définie par un couple de motifs avec orbites* L *et* R *dont toutes les étiquettes d'orbites ont la même longueur, telles que* $\underline{L} \longrightarrow \underline{R}$ *est une règle de transformation de graphe, dont un sous-ensemble des nœuds de* L *sont appelés nœuds d'accroche (*hooks *en anglais) et portent des étiquettes d'orbites pleines, c'est-à-dire sans « _ ».*

La figure 1.15 représente la règle Jerboa de triangulation qui reprend les motifs de la figure 1.12. Le nœud $n0$, dans le motif gauche, est le nœud d'accroche représenté graphiquement par un double cercle.



(a) G

(b) Instanciation de $M1$

(c) Instanciation de $M2$

(d) Instanciation de $M3$

(e) Instanciation de $M4$

FIGURE 1.14 – Instanciation des motifs de la figure 1.12 par l'orbite $G\langle 0,1,3\rangle(a)$

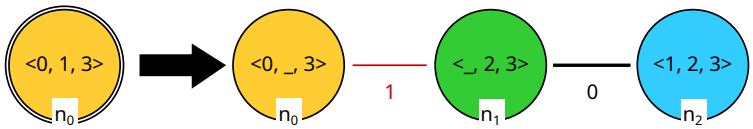


FIGURE 1.15 – Règle de triangulation

L'application d'une règle Jerboa est l'application de son instanciation, qui est une règle instanciée. Cela nécessite tout d'abord de rechercher dans l'objet à transformer l'orbite d'instanciation de la règle, d'instancier la règle, de construire le morphisme de filtrage entre le motif gauche instancié et l'objet, puis finalement d'appliquer la règle instanciée sur l'objet pour obtenir la transformation directe.

Définition 15 (Orbite d'instanciation)
Soit $\underline{m} : \underline{L} \to G$ un morphisme de filtrage sur une G-carte G, c'est-à-dire un morphisme injectif tel que les nœuds d'accroche désignent des orbites isomorphes à renommage près. Plus précisément, pour un nœud d'accroche h de L, $O = G\langle l(h)\rangle(\underline{m}(h))$ est l'orbite d'instanciation. *Si la règle r possède un autre nœud d'accroche h', alors $O' = G\langle l(h')\rangle(\underline{m}(h'))$ est isomorphe à O à renommage près, c'est-à-dire que $(l(h') \mapsto l(h))(O')$ est isomorphe à O.*

Par exemple, le morphisme de filtrage qui associe l'unique nœud $n0$ du motif gauche de la règle de triangulation (figure 1.15) au brin b de la G-carte G (figure 1.13a) définit l'orbite d'instanciation de la figure 1.13b. Le morphisme de filtrage qui associe le même nœud $n0$ au brin j de la G-carte G définit l'orbite d'instanciation de la figure 1.14b.

Un autre exemple est présenté avec la règle illustrée dans la figure 1.16. Celle-ci permet de coudre deux faces par la connexion de leurs liaisons 3. Nous pourrions souhaiter connecter deux faces quelconques. Ce n'est pourtant pas toujours possible. Prenons l'exemple d'un cube et d'une pyramide. Connecter une face du cube à une face triangulaire de la pyramide implique

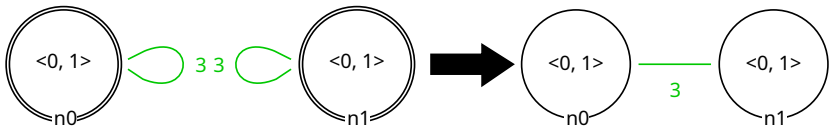
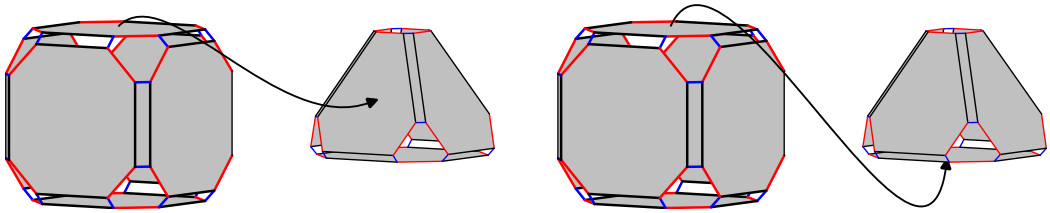


Figure 1.16 – Règle de coutures de faces



(a) Faces non isomorphes (b) Faces isomorphes

Figure 1.17 – Tentatives de couture de faces

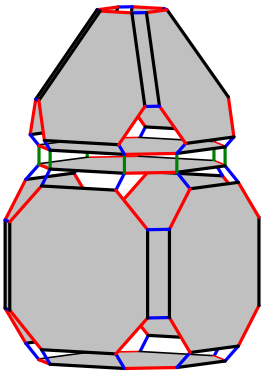


Figure 1.18 – Résultat de la couture figure 1.17b

de connecter un-à-un chacun des brins des deux faces, or celles-ci ne sont pas isomorphes. Réaliser cette opération sur ces deux faces sans transgresser la condition d'arcs incidents est donc impossible. Au contraire, si les faces à connecter sont isomorphes, alors il est possible de les coudre ensemble, comme dans la figure 1.18 où nous connectons la face du dessus du cube à la base de la pyramide.

Définition 16 (Filtrage de l'orbite d'instanciation)
Le morphisme de filtrage $m : L(O) \to G$*, s'il existe, est le morphisme qui vérifie les propriétés suivantes, pour* $L = (V, E, s, t, l, \alpha)$ *et* $O = (V', E', s', t', \alpha')$ *:*

- *pour tout nœud* $v \in V$ *de* L *et un brin* h *de* G*,* $m_V((v, \underline{m}(h))) = \underline{m}(v)$*;*
- *pour tout arc implicite* (v, e')*, avec* $v \in V$ *un nœud de* L *et* $e' \in E'$ *un arc de l'orbite* O *non supprimé (c'est-à-dire tel que* $\langle l(h)\rangle \mapsto \langle l(v)\rangle(\alpha'(e')) \neq \_$*) :* $m_E((v, e'))$ *est l'unique arc de* G *de source* $m_V((v, s'(e')))$ *et d'étiquette* $\langle l(h)\rangle \mapsto \langle l(v)\rangle(\alpha'(e'))$*; et* $m_V(t((v, e'))) = t'(m_E((v, e')))$*;*
- *pour tout arc explicite* (e, v')*, avec* $e \in E$ *un arc de* L *et* $v' \in V'$ *un nœud de l'orbite* O *:* $m_E((e, v'))$ *est l'unique arc de source* $m_V((s(e), v'))$ *et d'étiquette* $\alpha(e)$*; et* $m_V(t(e, v')) = t'(m_E(e, v'))$*.*

Remarque : *Si le morphisme de filtrage* m *existe, il est unique en raison de la contrainte d'arc incident de la G-carte* G *(définition 8) et de la condition d'arc incident du motif* L *(voir le théorème 2).*

Pour montrer qu'un filtrage est unique, prenons une face carrée de l'objet figure 1.13a comme exemple. Lorsque nous appliquons la règle de triangulation, conformément à la contrainte d'arc incident, pour tout brin appartenant à une même $\langle 0, 1, 3\rangle$-orbite, la face filtrée sera toujours la même. Par exemple, pour tout brin de $G\langle 0, 1, 3\rangle(a)$, la face filtrée est incidente au brin a (figure 1.13b) et nous obtenons le morphisme $n0 \mapsto a_0, b_0 \dots h_0$. Ainsi, si nous appliquons la règle de triangulation sur l'orbite $G\langle 0, 1, 3\rangle(e)$, $n0$ filtre la même face et produit le même morphisme $n0 \mapsto a_0, b_0 \dots h_0$.

Définition 17 (Instanciation d'une règle Jerboa)
*L'*instanciation *de la règle $r : L \longrightarrow R$ le long d'un morphisme de filtrage $\underline{m} : \underline{L} \to G$ sur une G-carte G, $r(\underline{m})$ est définie par l'instanciation de ses motifs $r(\underline{m}) : L(O) \longrightarrow R(O)$, où $O = G\langle l(h)\rangle(\underline{m}(h))$ est l'orbite d'instanciation d'un nœud d'accroche h de L.*

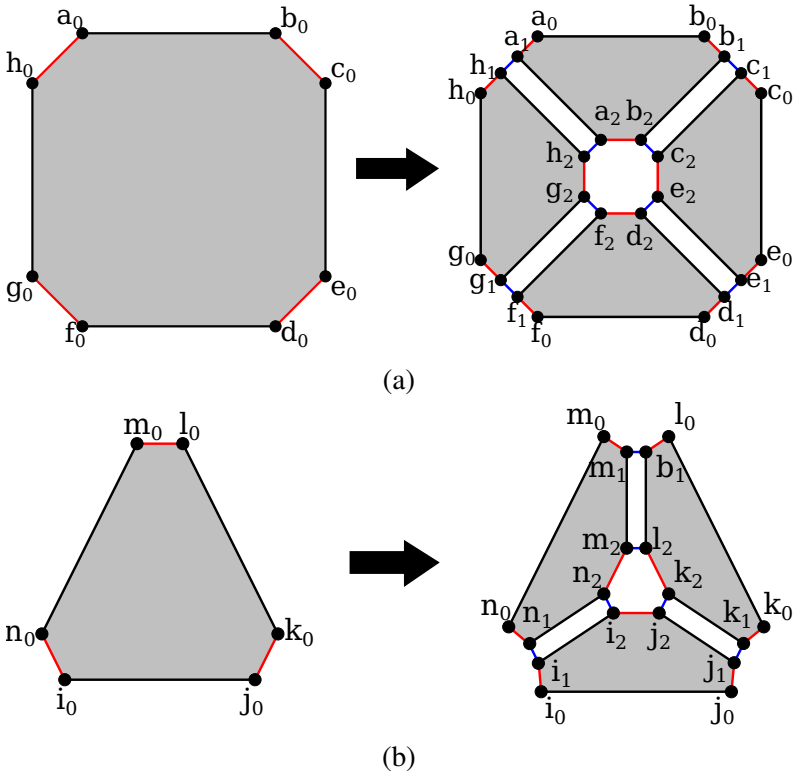
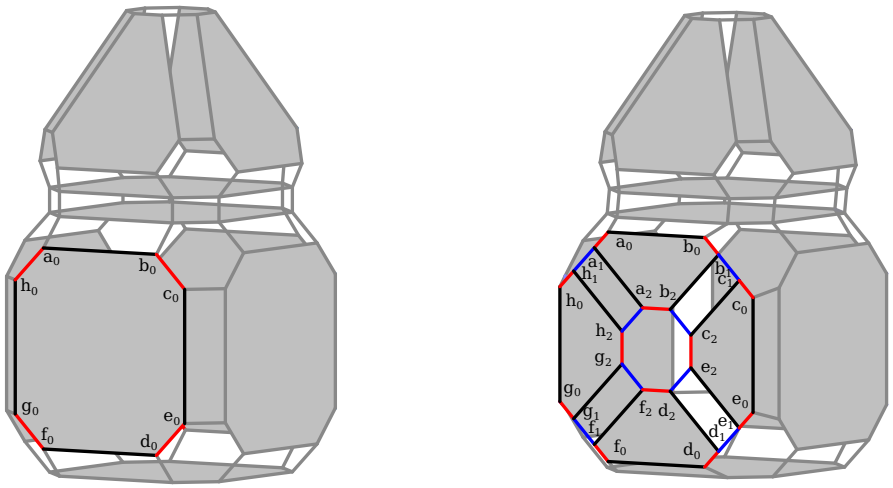


FIGURE 1.19 – Instanciation de la règle de triangulation sur une face carrée (a) et triangulaire (b)

Reprenons les instanciations des motifs $M1$ et $M4$ sur une face carrée (figure 1.13a) et sur une face triangulaire (figure 1.14a). Ces deux motifs sont respectivement les motifs L et R de la règle de triangulation. Ainsi, pour l'instanciation de la règle de triangulation sur une face carrée, nous obtenons la règle instanciée illustrée dans la figure 1.19a. Pour son instanciation sur une face triangulaire, nous obtenons la règle instanciée illustrée dans la figure 1.19b.

Remarque : *Les règles instanciées sont aussi des règles Jerboa dont les nœuds sont étiquetés par l'orbite vide $\langle\rangle$.*

L'*application* de la règle $r : L \longrightarrow R$ le long du morphisme de filtrage $\underline{m} : \underline{L} \to G$, est l'application de la règle instanciée $r(\underline{m}) : L(O) \longrightarrow R(O)$ sur le morphisme de filtrage instancié $m : L(O) \to G$, où $O = G\langle l(h)\rangle(\underline{m}(h))$ est l'orbite d'instanciation désignée par le nœud



(a) G (b) H

Figure 1.20 – Triangulation d'une face

d'accroche h. Autrement dit, la transformation directe $G \xrightarrow{r,m} H$ est la transformation directe $G \xrightarrow{r(\underline{m}),m} H$. Par exemple, l'application de la règle de triangulation nous donne la transformation directe de G (figure 1.20a) vers H (figure 1.20b) où $G\langle 0,1,3\rangle(a_0)$ est la face filtrée par la règle qui est instanciée dans la partie gauche de l'instanciation figure 1.19a ; $H\langle 0,1,3\rangle(a_0)$, $H\langle 0,1,3\rangle(c_0)$, $H\langle 0,1,3\rangle(e_0)$ et $H\langle 0,1,3\rangle(g_0)$ sont les faces issues de la triangulation.

Enfin, la définition 10 s'étend naturellement sur les règles Jerboa.

Définition 18 (Orbite complète et entièrement filtrée)
Soient G une G-carte, $\underline{m} : \underline{L} \to G$ un morphisme de filtrage Jerboa, O une orbite d'instanciation, v un nœud de L, v' un brin de O et $b = m(v,v')$ un brin de G filtré par v par le morphisme instancié $m : L(O) \to G$.
L'orbite $G\langle\omega\rangle(b)$ est entièrement filtrée *par $\underline{m}$ si $G\langle\omega\rangle(b) \subset m(L(O)$,* non filtrée *par $\underline{m}$ si $G\langle\omega\rangle(b) \cap m(L(O) = \emptyset$, ou* partiellement filtrée *par $\underline{m}$ dans les autres cas.*
L'orbite $L\langle\omega\rangle(v)$ est complète *si pour tout nœud w de $L\langle\omega\rangle(v)$ et toute étiquette i de $\langle\omega\rangle$, w est la source d'un i-arc implicite ou explicite.*

Remarque : *Grâce à la condition d'arc incident, l'orbite $L\langle\omega\rangle(v)$ est complète si et seulement si $G\langle\omega\rangle(b)$ est entièrement filtrée.*

5 Conclusion

Dans ce chapitre, nous avons présenté les outils et des concepts nécessaires à la compréhension des formalismes que nous utilisons tout au long de cette thèse. Ces formalismes nous permettent de représenter et manipuler les objets et les opérations de modélisation géométrique dans le contexte d'un système de modélisation à base de règles.

Nous avons commencé avec des rappels et des définitions générales sur une catégorie de graphes dits « étiquetés» grâce auxquels nous représentons les dimensions dans les objets. Nous avons aussi présenté divers outils permettant de manipuler ces graphes et ainsi préparer la définition des outils permettant de les transformer.

Afin de proposer des méthodes généralisables en toutes dimensions, nous avons présenté un modèle appelé *carte généralisée* qui formalise la représentation des objets géométriques de dimension n. Avec cet outil, nous présentons notamment la définition d'*orbite* que nous utilisons pour représenter l'ensemble des entités topologiques dans un objet. Cela nous permet également de définir des opérations sous la forme de *règles de transformation de graphe*.

Enfin, nous avons présenté le langage de règles Jerboa qui permet de construire des opérations à l'aide de ces règles : Tout d'abord, sous la forme de règles qui représentent explicitement les sous-graphes à transformer ; ensuite, sous la forme de motifs d'orbites qui permettent de filtrer et de transformer des orbites dans les cartes généralisées. Ce langage Jerboa nous permet alors de dégager des propriétés sur la topologie des objets transformés et leurs transformations et d'exploiter pleinement le formalisme de cartes généralisées.

CHAPITRE 2

État de l'art : modélisation paramétrique

Sommaire

1 Introduction

Le domaine de la CAO repose sur divers outils de modélisation, parmi lesquels les systèmes paramétriques qui permettent de générer des variations d'un modèle à l'aide d'un mécanisme de *rejeu*, ou de *réévaluation* [Dasa ; Dasb ; Autb ; Autd ; Sie]. Ces systèmes enregistrent le processus de construction, c'est-à-dire une liste d'opérations ou de contraintes appliquées à des entités géométriques (points, droites, caractéristiques de formes, etc.) permettant de modéliser un objet. L'utilisateur peut ensuite modifier certains paramètres de la construction ou ajouter, déplacer, voire supprimer des opérations et des contraintes. Le processus de construction est alors automatiquement rejoué ou réévalué, générant une version modifiée de l'objet initial, en fonction des modifications apportées.

Dans ce chapitre, nous commençons par définir plus précisément la notion de *système paramétrique*, en détaillant sa structure et les mécanismes de réévaluation des modèles. Nous verrons que la réévaluation peut échouer si le système ne gère pas correctement les références associées aux entités topologiques ; et que, dans ce cas, il est nécessaire d'intégrer un mécanisme de *nomination persistante* pour garantir la robustesse du processus.

2 Approches de modélisation paramétriques

Un système paramétrique peut être considéré comme une structure duale composée, d'une part, d'une représentation explicite, appelée *couche géométrique*, qui décrit la forme visible d'un objet, et d'autre part, d'une représentation implicite, nommée *couche de spécification paramétrique* (ou simplement *spécification*). Cette couche implicite contient la liste des contraintes et des opérations appliquées à l'objet, ainsi que leurs paramètres associés. La spécification peut être modifiée et réévaluée, ce qui permet de générer de nouvelles représentations explicites en fonction des modifications effectuées.

Nous distinguons deux principales approches dans le cadre de la modélisation paramétrique, chacune avec son type de spécification. La première est l'approche *équationnelle* (ou *variational*). La seconde est l'approche *fonctionnelle* (ou *history-based*). Ces deux approches sont décrites dans les sections suivantes.

2.1 Approche équationnelle (ou *variational*)

Dans l'approche équationnelle, la représentation implicite (ou spécification paramétrique) d'un modèle est définie par un ensemble de contraintes géométriques (telles que des angles, distances, longueurs, parallélismes, tangences, etc.) appliquées à des entités géométriques. Un solveur est ensuite utilisé pour résoudre ce système de contraintes (lorsqu'une solution existe), afin de calculer une représentation explicite du modèle, c'est-à-dire un objet. Une définition plus formelle est présentée ci-dessous [Pie+96] :

Définition 19 (Modèle paramétrique équationnel)
Soit P l'ensemble des paramètres définis sur un domaine $\mathcal{D}$, où $\mathcal{D}$ est souvent égal à $\mathbb{R}^n$. Soit V l'ensemble des variables (points, courbes, valeurs numériques, etc.) qui doivent être évaluées pour décrire une instance explicite, où V appartient à $\mathcal{V}$ l'ensemble de toutes les

formes possibles. Un modèle paramétrique équationnel *est une équation de la forme :*

$$A(P, V) = 0$$

où A est un opérateur qui n'est, généralement, ni linéaire ni convexe.

Dans le cas des logiciels d'esquisse 2D (*2D sketchers*), cette approche présente plusieurs avantages tels que la simplicité de concevoir un objet « à la main », c'est-à-dire en dessinant d'abord une esquisse « à main levée » avant d'assigner des contraintes géométriques une à une.

Cependant, cette approche présente plusieurs limitations :
- le nombre de solutions possibles pour un système est généralement exponentiel, en raison du nombre de contraintes non linéaires ;
- il est nécessaire de définir une approximation initiale pour les calculs de convergence et des heuristiques pour discriminer plusieurs solutions ;
- l'usage exclusif de contraintes non orientées empêche la réalisation d'actions orientées telles que l'extrusion, les opérations booléennes, *etc.*
- les contraintes pouvant être introduites librement, il est difficile pour l'utilisateur de savoir quand et pourquoi une entité est sous ou surcontrainte, et pour l'ordinateur, de fournir une solution dans le premier cas, ou de montrer des conflits de contraintes dans le second ;
- les solutions sont très souvent calculées de manière approximative par des méthodes de résolution numérique.

2.2 Approche fonctionnelle (ou *history-based*)

Dans l'approche fonctionnelle, la spécification paramétrique d'un objet est un processus constructif (ou historique), autrement dit, un enregistrement des opérations de modélisation et de leurs paramètres, appliquées successivement tout au long du processus de modélisation. Les paramètres des opérations peuvent être des valeurs géométriques ou des références à des entités topologiques existantes. La représentation explicite d'un objet, quant à elle, est entièrement générée au fur et à mesure de l'application du processus constructif, ce qui la rend déterministe.

L'approche fonctionnelle est définie comme suit [Pie+96] :

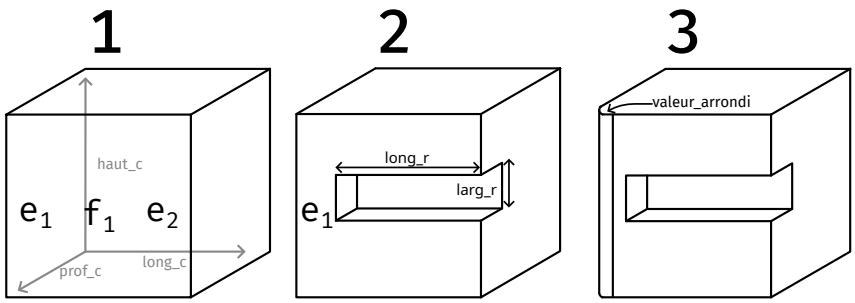
Définition 20 (Modèle paramétrique fonctionnel)
Soit P l'ensemble des paramètres définis sur un domaine $\mathcal{D}$, et V une instance explicite, où V appartient à $\mathcal{V}$ l'ensemble de toutes les formes possibles. Un modèle paramétrique est une fonction :

$$F : \mathcal{D} \mapsto \mathcal{V}$$

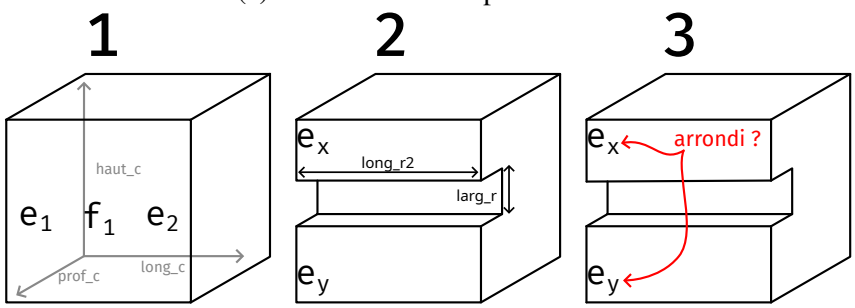
avec $V = F(P)$, où F est définie comme une composition de fonctions f_i appelées fonctions paramétriques *:*

$$F = f_n \circ f_{n-1} \ldots \circ f_1$$

Représenter un modèle paramétrique sous forme de fonction offre plusieurs avantages : le système est déterministe, garantissant ainsi un résultat unique ; cette approche s'applique aussi bien en 2D qu'en 3D pour les modèles basés sur la *CSG* (*Constructive Solid Geometry*) et ceux basés sur la représentation par frontières (*Boundary Representation, B-Rep*) ; ; enfin, des opérations de haut niveau sont supportées telles que les opérations booléennes, l'extrusion, le chanfrein, ou encore la rainure.



(a) Évaluation de la spécification



(b) Réévaluation de la spécification

Figure 2.1 – Représentation explicite : Évaluation et réévaluation d'un modèle

Prenons l'exemple d'une spécification paramétrique (représentation implicite) :
— `1-Cube(long_c, haut_c, prof_c)`
— `2-Rainure(`f_1`,long_r, larg_r)`
— `3-Arrondi(`e_1`,valeur_arrondi)`

Le résultat (représentation explicite) est illustré sur la figure 2.1a. La première opération crée un cube paramétré par les trois valeurs `long_c`, `haut_c` et `prof_c`. Nous nous intéressons, en particulier, à trois entités topologiques nommées f_1, e_1 et e_2, qui sont respectivement la face avant du cube et ses arêtes verticales gauche (e_1) et droite (e_2). La deuxième opération, appliquée sur f_1, creuse une rainure dans cette face et coupe l'arête e_2 en deux. Enfin, la troisième opération crée un arrondi sur l'arête e_1.

La spécification paramétrique ci-dessus (et sa représentation explicite figure 2.1a) illustre plusieurs aspects de la rigidité de l'approche fonctionnelle : (1) l'utilisateur est limité à un ensemble d'opérations prédéfinies dans le modeleur ; (2) l'approche est séquentielle, ce qui implique que toute entité doit exister à l'étape courante de la modélisation pour pouvoir être référencée, c'est-à-dire utilisée comme paramètre d'entrée de l'opération en cours. Par exemple, suite à l'application de la fonction `rainure` à l'étape 3, l'arête e_2 de la face avant du cube initial n'existe plus : elle a été « remplacée » par une succession de nouvelles arêtes. Il devient alors impossible d'appliquer une fonction sur la référence e_2 à partir d'une éventuelle étape 4.

Une autre difficulté associée à l'approche fonctionnelle réside dans le référencement d'entités qui peuvent ne plus exister ou avoir été modifiées lors de la réévaluation. Considérons, par exemple, le rejeu de la spécification paramétrique précédente, légèrement modifiée (figure 2.1b) :
— `1-Cube(long_c, haut_c, prof_c)`
— `2-Rainure(`f_1`, long_r2, larg_r)`
— `3-Arrondi(`e_1`,rayon)`

Dans cette nouvelle spécification, la valeur `long_r` de la deuxième opération est remplacée par `long_r2`. Lors de l'évaluation initiale, la rainure ne traversait pas la face f_1 de part en part et l'arête e_1 était donc identifiée sans difficulté. Or, à la réévaluation, la rainure traverse f_1 de part en part et l'arête e_1 est scindée en e_x et e_y. L'arête e_1 n'existant plus, il n'est alors plus possible d'appliquer l'opération d'arrondi sur la référence e_1.

Cette difficulté illustre une limitation majeure de l'approche fonctionnelle. Lors de la réévaluation, si une entité référencée dans la spécification paramétrique cesse d'exister, il devient alors impossible d'utiliser sa référence, ce qui empêche de produire une nouvelle représentation explicite du modèle sans un mécanisme de nomination plus robuste.

3 Nomination persistante

L'exemple présenté dans la section précédente met en évidence la nécessité d'un mécanisme de nomination persistante : celui-ci doit permettre la régénération d'un modèle malgré la présence de références invalides. Pour cela, il est essentiel que chaque entité topologique d'un modèle paramétrique soit identifiable à l'aide d'un identifiant unique et robuste, appelé *nom persistant*. De plus, le nom attribué à chaque entité doit faciliter leur *appariement* dans le modèle paramétrique régénéré, c'est-à-dire permettre de retrouver, dans le nouveau modèle, les entités qui avaient été initialement désignées pour appliquer les opérations de modélisation.

3.1 Nomination des entités

Dans le contexte de la modélisation paramétrique sans réévaluation, une entité peut être désignée à l'aide d'un identifiant simple, comme un numéro d'apparition dans le modèle ou un pointeur. Cependant, dans le cadre de la modélisation paramétrique avec réévaluation, ces méthodes de référencement deviennent inapplicables. En effet, de telles références sont particulièrement sensibles aux changements topologiques subis par le modèle lors de la réévaluation de la spécification. Par exemple, un numéro d'ordre d'apparition pourrait désigner une entité topologique différente, incompatible avec l'opération qui la référence. De même, si l'entité initialement référencée par un pointeur disparaît, le programme échouera en raison d'une erreur de pointeur nul.

Reprenons l'exemple précédent. Lors de l'évaluation initiale (figure 2.1a), e_1 désigne l'arête qui doit être arrondie à la troisième étape de la modélisation. Suite à la réévaluation (figure 2.1b), la rainure, dont la longueur a changé, scinde e_1 en deux et n'existe plus telle quelle. Par conséquent, utiliser le nom e_1 ou un identifiant correspondant à e_1 comme référence n'est pas approprié, car il reste sensible aux modifications du modèle.

La nomination persistante doit donc être un mécanisme d'identification robuste, c'est-à-dire un système dont les références restent valides au cours de la réévaluation d'une spécification paramétrique, malgré les modifications géométriques et topologiques. Ce mécanisme s'intègre alors comme une troisième couche dans les systèmes de modélisation paramétrique, entre la spécification paramétrique et la représentation explicite.

3.2 Appariement des entités

Comme nous l'avons vu, lorsque la spécification paramétrique est modifiée et réévaluée, les entités référencées sont susceptibles d'être modifiées. L'appariement des entités découle alors directement du mécanisme de nomination persistante.

Dans le cas le plus simple, lorsqu'une entité est préservée, il faut pouvoir mettre cette entité du modèle initial en correspondance avec son équivalent dans le modèle régénéré. En revanche, dans d'autres configurations, les changements provoqués peuvent conduire à plus ou moins d'options pour la mise en correspondance.

Reprenons une nouvelle fois l'exemple de la figure 2.1. Suite à la réévaluation, la scission de l'arête e_1 produit les arêtes e_x et e_y. Avec simplement e_1 comme référence, il est impossible de poursuivre la réévaluation avec l'application de l'opération d'arrondi sur l'une et/ou l'autre des deux arêtes scindées. Dans cette situation, avec un nom permettant d'identifier l'arête e_1 malgré sa scission, un mécanisme d'appariement doit permettre de mettre e_1 en correspondance avec e_x et e_y et d'appliquer l'opération d'arrondi sur l'une ou l'autre de ces arêtes, voire sur les deux.

4 Classification des méthodes de nomination persistante

La nomination persistante, ainsi que l'identification et l'appariement des entités topologiques, constitue un enjeu majeur pour la réévaluation d'un modèle paramétrique. Au cours des dernières années, de nombreuses méthodes ont été proposées pour mettre en œuvre un système permettant de réévaluer des modèles paramétriques. Dans cette section, nous commençons par identifier des concepts communs aux différentes approches. Ensuite, nous passons en revue de manière plus détaillée les systèmes les plus significatifs proposés pour résoudre le problème de la nomination persistante. Enfin, nous proposons une synthèse de ces méthodes.

4.1 Concepts communs de la nomination persistante

Les différents systèmes de nomination persistante reposent, explicitement ou implicitement, sur un ensemble de concepts. Ces concepts définissent principalement des propriétés des entités topologiques, permettant de les classer en catégories distinctes. Cela facilite la mise en œuvre de mécanismes spécifiques pour la nomination et l'appariement.

Distinction des entités invariantes et contingentes

Le premier concept que nous abordons est celui qui propose de distinguer les entités dites *invariantes* de celles dites *contingentes* [AMP99].

> **Définition 21 (Entité invariante)**
> *Une* entité invariante *est une entité géométrique ou topologique qui peut être, complètement et sans ambiguïté, caractérisée par la structure d'une opération de modélisation et ses paramètres d'entrée, indépendamment des valeurs impliquées.*

Pour illustrer cela, observons la figure 2.2. Nous pouvons voir, lors des étapes 1 et 2 de la modélisation, un ensemble d'entités topologiques dites invariantes générées lors de la création

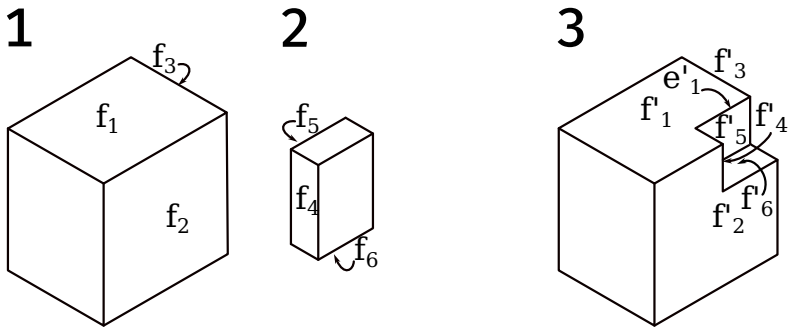


FIGURE 2.2 – Opération booléenne (différence) entre deux objets

des deux blocs. Quels que soient les paramètres des opérations créant ces deux blocs, nous savons que l'ensemble de ces entités seront systématiquement générées, car elles sont caractéristiques de l'opération. Ainsi, l'ensemble des entités topologiques des blocs, telles que les faces f_1 et f_5, sont invariantes.

Définition 22 (Entité contingente)
Une entité contingente *est une entité topologique ou géométrique qui résulte de l'interaction entre le modèle géométrique pré-existant et les entités invariantes issues d'un geste constructif particulier.*

Dans l'étape 3 de la modélisation, la différence entre les deux blocs génère un ensemble de nouvelles entités topologiques dites contingentes. Ces entités dépendent des paramètres des opérations (ici la taille ou la position des deux blocs) et ne sont donc pas systématiquement créées. Parmi ces entités, on trouve par exemple les faces f'_1, f'_2, f'_3, f'_4, f'_5 et f'_6 ou bien l'arête e'_1. En effet, la création et la modification de ces entités dépendent des paramètres des opérations. Si, par exemple, nous déplaçons le second bloc de façon à ne créer aucune intersection entre les deux objets, ou si nous modifions sa taille de façon à ce que la face f_1 se retrouve scindée en deux, alors certaines entités contingentes ne seront plus créées ou au contraire d'autres apparaîtront. De même si l'on reprend l'exemple sur la figure 2.1, au rejeu, le changement de la longueur de la rainure scinde la face f_1 et crée deux nouvelles faces contingentes.

Entités issues d'extrusion ou de révolution

Le second concept commun que nous abordons est un mécanisme de nomination des entités invariantes générées lors d'opérations d'extrusion ou de révolution. Ces opérations génèrent des volumes à partir d'entités topologiques 2D. Ces entités 2D sont les paramètres des opérations d'extrusion et de révolution. Dans le cas d'une extrusion, par exemple, l'opération nécessite deux paramètres : un *profil* à extruder et une *trajectoire* (ou *guide*) d'extrusion. Dans le cas d'une révolution, un profil peut être extrudé autour d'un axe selon un angle de rotation. Un pré-requis à ces opérations est que chacune des structures 2D dispose d'un nom robuste, c'est-à-dire qui persiste indépendamment des changements de la topologie. Différentes méthodes ont été proposées pour procéder à la nomination persistante des entités générées lors d'une extrusion [Che95 ; Wu+01 ; Agb+03]. Cependant, ces méthodes suivent toutes, dans l'ensemble, le même procédé : les noms des entités générées sont dérivés de ceux des entités invariantes impliquées dans leur construction.

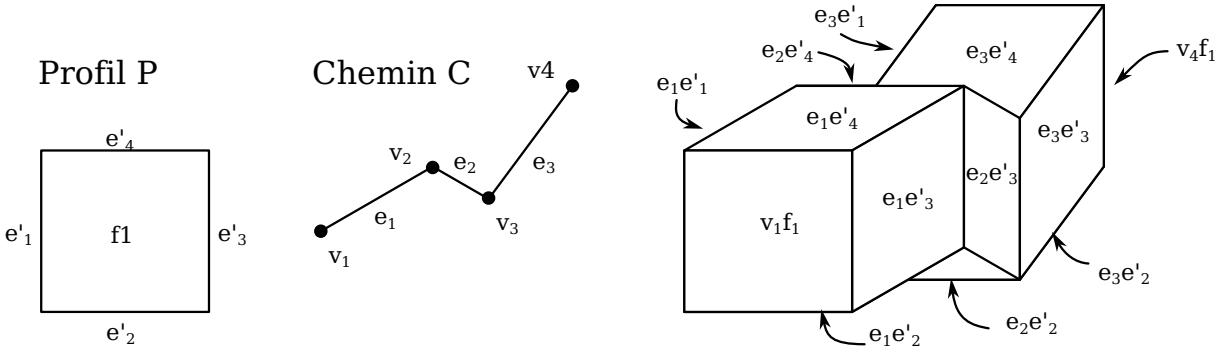
Prenons l'exemple des travaux de Wu et al. [Wu+01]. Ces auteurs proposent une identifi-

cation des entités topologiques générées par extrusion (ou révolution), basée sur le schéma de nomination suivant :

$$[ID_{operation}, ID_{profil}, ID_{element}, ID_{chemin}, ID_{trajectoire}]$$

où $ID_{operation}$ est l'identifiant de l'opération, ID_{profil} et ID_{chemin} sont respectivement les identifiants du profil et de son chemin d'extrusion, $ID_{element}$ est l'identifiant d'une entité du profil à extruder en suivant la direction d'une arête identifiée par $ID_{trajectoire}$.

Dans le cadre d'une extrusion d'une face le long de sa normale, ID_{chemin} et $ID_{trajectoire}$ sont initialisés à 0. En prenant l'exemple présenté dans la figure 2.3, l'identifiant de la face $e_1e'_3$ a pour identifiant $[1, 4, 3, 3, 1]$ où ID_{profil} est le profil d'identifiant 4, $ID_{element}$ est l'arête e'_3 d'identifiant 3, ID_{chemin} est le chemin d'extrusion identifié par 3, et où $ID_{trajectoire}$ est l'arête e_1 d'identifiant 1.



Entité	P	C	e_1	e_2	e_3	v_1	v_2	v_3	v_4	e'_1	e'_2	e'_3	e'_4
Identifiant	4	3	1	2	3	4	5	6	7	1	2	3	4

Figure 2.3 – Exemple de nomination des entités lors d'une extrusion, Baba-Ali

Entités avec et sans sources

Le troisième concept commun présent dans la plupart des approches, bien qu'il ne soit pas toujours défini de manière explicite, consiste à distinguer les entités selon qu'elles possèdent une source ou non. Les entités avec *source* sont définies de la manière suivante [Wu+01].

Définition 23 (Entité avec source)
Une entité avec source est une entité topologique résultant d'une autre entité topologique de même dimension, et ayant une partie du plongement en commun avec cette dernière, suite à l'application d'une opération de modélisation.

D'après Wu et al. [Wu+01], une entité topologique E issue d'une opération booléenne entre deux objets O_1 et O_2, est une entité avec source s'il existe une entité topologique E', de même dimension, appartenant à O_1 ou O_2 telle que E et E' ont au moins une partie de leur plongement géométrique en commun et dans ce cas, E' et la source de E.

Par exemple, dans la figure 2.2 , la face f'_1, modifiée lors de l'opération booléenne, partage une partie de son plongement géométrique avec f_1 qui est donc sa source. De la même manière,

f'_5 a pour source f_5. L'arête e_1 résultant de l'intersection de deux faces et ne partageant le plongement géométrique d'aucune autre entité de même dimension est en revanche une entité sans source.

Nous pouvons observer que lorsque l'on travaille avec un modeleur en dimension n, les entités contingentes de dimension $n-1$ sont toujours des entités avec sources. Ainsi, en 3D les faces contingentes sont toutes avec source. À l'inverse, ce n'est pas forcément le cas pour les arêtes et sommets qui peuvent être créés par l'intersection de plusieurs entités de dimension supérieure. Une conséquence de cela, bien qu'elle ne soit pas toujours exprimée de manière explicite, est que la plupart des modeleurs 3D d'aujourd'hui s'appuient sur les faces pour leurs mécanismes de nomination. En effet, le fait qu'elles soient invariantes ou qu'elles aient systématiquement une source les rend plus facilement identifiables, ce qui permet une nomination plus robuste des autres entités.

Utilisation d'un historique de noms

Le quatrième concept commun dans la littérature concerne l'utilisation d'un historique permettant de *tracer l'évolution* des ancêtres invariants. Un tel système permet alors d'identifier et d'apparier de façon plus robuste les entités par rapport à leurs ancêtres lors de la réévaluation.

La littérature présente deux principales approches pour construire un historique pour la nomination persistante. La première approche consiste à construire un graphe orienté explicite [Kri95 ; AMP00 ; Bab+07 ; Car+19] où chaque nœud est une entité topologique précédée de ses ancêtres et suivie de ses évolutions. La seconde approche consiste en un mécanisme implicite de propagation des noms des ancêtres invariants sur les entités descendantes [CCH96 ; Wu+01].

Un exemple de construction explicite de graphe d'historique est fourni par Kripac [Kri95]. Il propose une historisation des faces d'un objet à l'aide d'un graphe orienté acyclique (*DAG* en anglais) pour représenter l'ensemble des changements topologiques intervenant lors de la modélisation. Dans ce DAG, un nœud, appelé un *FaceIdNode*, représente une face étiquetée $f_{m.n}$ où m est l'étape de modélisation et n l'index de la face à l'étape m.

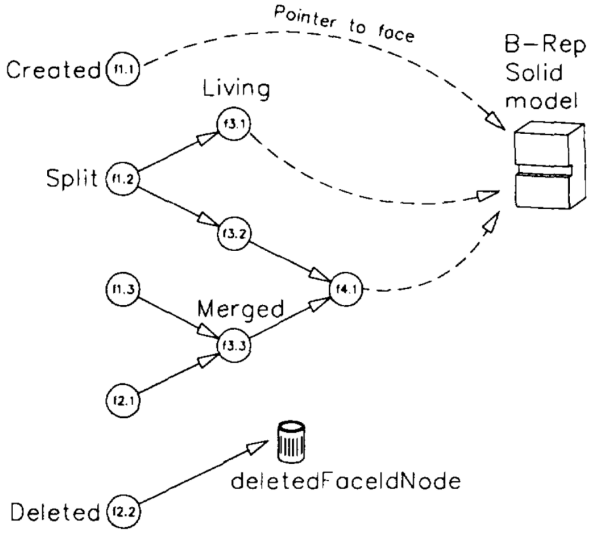
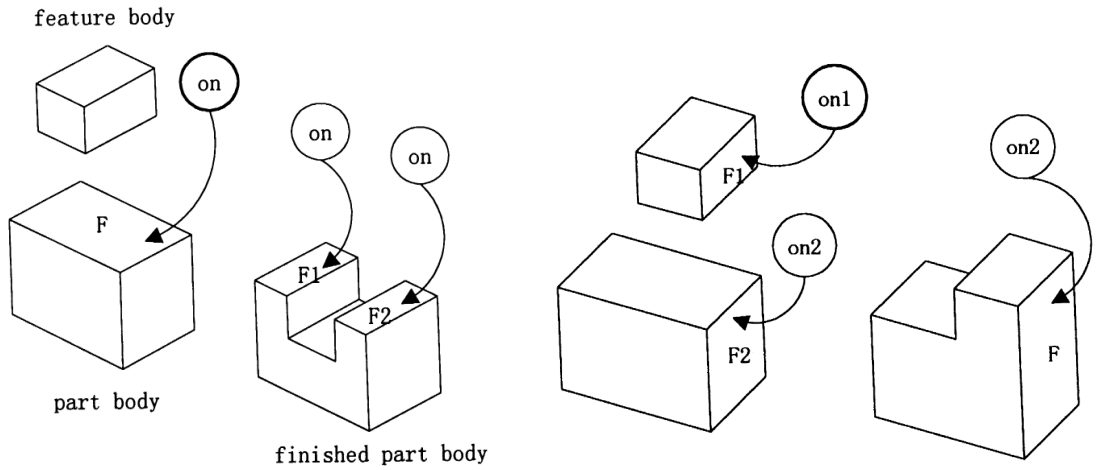


FIGURE 2.4 – Exemple de DAG de suivi de face dans un modèle, Kripac

Le DAG, illustré dans la figure 2.4, représente l'évolution des faces d'un bloc dont la face frontale a été creusée par une rainure. Ce DAG montre que les faces $f_{1.1}$, $f_{1.2}$ et $f_{1.3}$ ont été créées avec une première opération et les faces $f_{2.1}$ et $f_{2.2}$ avec l'opération suivante. Au cours des opérations suivantes, $f_{1.2}$ est scindée en deux faces $f_{3.1}$ et $f_{3.2}$ et $f_{1.3}$ et $f_{2.1}$ sont fusionnées en une face $f_{3.3}$. La face $f_{2.2}$ est supprimée puis $f_{3.2}$ et $f_{3.3}$ sont fusionnées en $f_{4.1}$. Au terme de la modélisation et parmi les faces du DAG, seules les faces $f_{1.1}$, $f_{3.1}$ et $f_{4.1}$ sont présentes dans le modèle.



(a) Scission (b) Fusion

Figure 2.5 – Exemple de propagation de nom persistant, Wu

Un exemple de construction implicite d'un historique par héritage des noms des ancêtres invariants est celui développé par Wu et al. [Wu+01]. Leur système propose différents schémas de nomination, notamment pour les entités extrudées, où chaque entité topologique générée reçoit un nom unique appelé *original name* (on). Une fois les on attribués, ce système met en œuvre un mécanisme de propagation des noms persistants (figure 2.5) qui, lors d'une scission par exemple, propage le nom d'une entité à chacune des deux entités scindées, autrement dit aux entités descendantes de même dimension.

C'est cette mécanique de propagation de nom qui, implicitement, construit un historique de noms. Un tel exemple est illustré dans la figure 2.5a. Suite à la différence entre les deux blocs, une pièce (*part body*) et une caractéristique (*feature body*), la face désignée par F est scindée en deux faces $F1$ et $F2$. F porte un nom on qui est propagé aux faces $F1$ et $F2$ qui ont alors le même nom persistant. Pour la fusion, illustrée dans la figure 2.5b, les faces $F1$ et $F2$, nommées $on1$ et $on2$ respectivement, sont fusionnées. Dans cette approche, puisque $on2$ est associé à une face de la *part body*, c'est ce nom qui est propagé à la pièce finale.

Nomination hétérogène des entités

Le cinquième concept commun concerne l'hétérogénéité des schémas de nomination dans les systèmes de réévaluation. La plupart des systèmes utilisent les entités topologiques de dimension $n-1$ (où n est la dimension du modèle) pour caractériser l'ensemble des autres entités topologiques, et proposent des schémas de nomination différents pour chaque dimension. Ce

choix est expliqué par le fait que les autres entités topologiques peuvent être entièrement définies comme des intersections d'entités de dimensions $n-1$. En 3D par exemple, une arête est une intersection de deux faces, un sommet une intersection de trois faces.

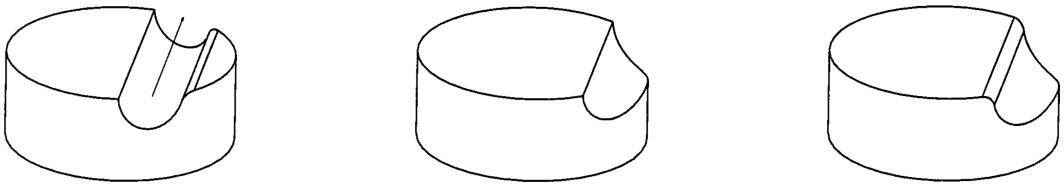
Un tel choix mène à nommer les entités topologiques en fonction de leur voisinage [Che95 ; Kri95 ; CCH96 ; AMP00], voire à ajouter des informations géométriques [Wu+01 ; BNB05 ; WN05] (espace paramétrique, orientation, gradient, etc.) lorsque la topologie seule ne suffit plus à lever les ambiguïtés. Par exemple, cela est nécessaire lorsque deux arêtes issues d'une opération de différence partagent un voisinage identique. L'inconvénient de définir un schéma de nomination de cette manière est que chaque système est conçu pour une dimension de modélisation spécifique, rendant la généralisation vers d'autres dimensions complexe et nécessitant une redéfinition complète des mécanismes de nomination.

Dans ses travaux, Baba-Ali [Bab10] met en évidence une conséquence liée à la nomination hétérogène des entités topologiques. Dans la littérature, peu de méthodes traitent la nomination d'entités de dimensions supérieures aux faces ou d'agrégats d'entités, tels que les agrégats de faces aussi appelées coques dans les systèmes de modélisation 3D. Rappelons que la plupart des systèmes de nomination persistante reposent sur les faces pour définir les autres entités, généralement les arêtes et les sommets, qui sont considérés comme des intersections de faces. De plus, ces systèmes utilisent des mécanismes de nomination hétérogènes en fonction de la dimension. Par essence, ces systèmes sont donc inadaptés à une extension naturelle des mécanismes de nomination vers de nouvelles entités. Il existe ainsi, dans les systèmes de modélisation paramétrique, un manque à combler en ce qui concerne la nomination des volumes, agrégats et autres entités de dimension supérieures aux faces. Permettre leur nomination offrirait aux utilisateurs de logiciels CAO une plus grande marge de manœuvre dans leurs processus constructifs.

Nomination des entités issues d'intersections non-planaires

Le sixième concept concerne la mise en œuvre d'un mécanisme capable de lever l'ambiguïté pour les entités topologiques issues d'intersections de surfaces non-planaires. Les intersections d'entités non-planaires sont souvent sources d'ambiguïtés et compliquent la nomination des entités topologiques issues de ces intersections. Les systèmes de nomination utilisent la plupart du temps d'abord la topologie pour caractériser les entités. En effet, dans le cas d'intersections planaires, les voisinages plus ou moins étendus [Che95] permettent systématiquement de distinguer les entités. À l'inverse, lors de l'intersection avec une ou plusieurs entités non-planaires, l'information topologique seule ne suffit plus toujours à distinguer les entités générées. En effet, les entités issues d'une telle intersection peuvent partager exactement les mêmes voisinages topologiques et il devient alors nécessaire d'utiliser des informations géométriques permettant de lever ces ambiguïtés.

Prenons l'exemple illustré dans la figure 2.6. Dans l'évaluation initiale (figure 2.6a), une première opération creuse une rainure cylindrique dans un cylindre. Cette opération crée deux arêtes le long de la rainure dont les voisinages topologiques contiennent les mêmes faces. La deuxième opération arrondie ensuite l'une de ces deux arêtes. Dans la réévaluation (figure 2.6b), l'intersection est déplacée sur un bord du cylindre et ne crée plus qu'une seule des deux arêtes. Le problème présenté dans cet exemple et le suivant : puisque les deux arêtes de l'évaluation initiale ont exactement le même voisinage topologique, dans la réévaluation il est possible que l'appariement désigne la mauvaise arête pour ré-appliquer l'opération d'arrondi (figure 2.6c).



(a) Évaluation initiale (b) Réévaluation (c) Réévaluation

Figure 2.6 – Exemple d'intersection de cylindres, Chen

Ici, la topologie n'est pas suffisante pour discriminer les deux arêtes et faire le bon appariement lors de la réévaluation.

Ainsi, la plupart des systèmes proposent d'ajouter des informations géométriques aux noms persistants pour permettre l'identification des entités issues de telles intersections [Wu+01 ; BNB05 ; WN05]. Wu et al. [Wu+01], par exemple, proposent de représenter les entités topologiquement ambiguës dans un espace paramétrique (u, v) et de les ordonnancer pour les distinguer les unes des autres. Bidarra et al. [BNB05] proposent de découper l'espace en deux domaines, positif et négatif, à l'aide d'un plan de référence. L'appartenance d'une entité à un domaine ou un autre par rapport au plan de coupe permet alors de distinguer deux entités symétriques. Baba-Ali et al. [BMS09] prennent en considération la courbure d'une surface, à savoir ses extremums locaux (des bosses et des creux), pour la partitionner. L'information géométrique obtenue pour chaque sous-surface permet alors d'améliorer la caractérisation et l'appariement des arêtes.

Structure des plateformes qui intègrent la nomination persistante

Le dernier des concepts communs que nous présentons concerne l'*extension* de l'architecture logicielle des modeleurs paramétriques avec l'ajout d'une couche pour la nomination et l'appariement des entités topologiques. Cette extension, issues des différentes contributions pour la réévaluation, produit l'architecture à trois couches suivante :

- la couche de spécification paramétrique qui contient l'ensemble des opérations de modélisations, issues du processus constructif, définissant un objet ;
- la couche de nomination qui permet une identification sans ambiguïté des entités topologiques et leur appariement lors de la réévaluation ;
- la couche géométrique, qui contient la représentation géométrique d'un objet, c'est-à-dire ses entités géométriques et topologiques, après évaluation de la spécification paramétrique.

4.2 Systèmes de nomination persistante

Dans cette section, nous décrivons les différents systèmes de nomination persistante publiés, en présentant leurs avantages et limitations dans le contexte de la réévaluation.

Kripac (1995)

Dans ses travaux, Kripac propose un système pour répondre au problème de la nomination persistante et permettre la réévaluation de modèles 3D [Kri95]. Les modèles sont présentés

comme des « volumes 3D délimités par des faces » et les arêtes et les sommets sont considérés comme des intersections de faces. Ainsi, les faces sont utilisées comme éléments principaux de la nomination persistante.

Le nom d'une face, $faceId$, est défini par :

$$FaceId(f) = [stepId, faceIndex, surfaceType]$$

où $stepId$ est le numéro d'application de l'opération qui crée f, $faceIndex$ l'indice de f et $surfaceType$ le type de surface de f (plan, cylindre, ...).

Le nom d'une arête, $edgeId$, est défini par :

$$EdgeId(e) = [adjFaceIds, endFaceIds_{0,1}, edgeIntersCode]$$

où $adjFaceIds$ est une liste des faces incidentes à e, cette liste est ordonnée selon un parcours de voisinage ; $endFaceIds_0$ (respectivement $endFaceIds_1$) est l'ensemble des faces à l'extrémité 0 (respectivement 1) de e ; et $edgeIntersCode$ un code d'intersection basé sur la position relative aux surfaces des faces incidentes à e, ce qui rend le code insensible aux transformations telles que le déplacement et le changement d'échelle.

Le nom d'un sommet, $vertexId$, est défini par :

$$VertexId(v) = [adjFaceIds, vertexIntersCodes]$$

où $adjFaceIds$ est une liste des faces incidentes à v, cette liste est ordonnée selon un parcours de voisinage ; et $vertexIntersCodes$ est l'ensemble des codes d'intersections qui indiquent la position de v par rapport à chacune des faces de son voisinage.

Le suivi des faces est réalisé à l'aide d'un DAG, présenté plus tôt (voir figure 2.4). Ce DAG permet de tracer l'évolution de chacune des faces, c'est-à-dire leur création, suppression, scission ou encore leurs fusions, dans le modèle initial. Lors de la réévaluation, ce graphe est réutilisé et mis à jour pour effectuer l'appariement des entités topologiques entre le modèle initial et le modèle réévalué. L'appariement des entités est effectué en cherchant d'abord une correspondance exacte des identifiants. Pour les faces, lorsqu'il n'existe pas de correspondance exacte, la procédure d'appariement remonte le DAG du modèle initial jusqu'à trouver une face qui existe aussi dans le DAG du modèle réévalué. Lorsque qu'une correspondance est trouvée, le DAG du modèle réévalué est parcouru depuis la face trouvée jusqu'aux dernières faces. Si le parcours se termine avec plusieurs faces, alors l'ensemble de ces faces est utilisé pour l'appariement. Pour les arêtes et les sommets, lorsqu'un appariement exact ne peut être trouvé, alors une recherche de similarité est effectuée entre les entités du modèle initial et celles du modèle réévaluée. Cette procédure consiste en une estimation de ressemblance basée sur les faces du voisinage et les codes d'intersections, entre les deux modèles. Dans cette logique, Kripac propose un système de requêtes pour définir le degré de similarité devant être atteint pour effectuer l'appariement entre plusieurs entités topologiques.

Synthèse

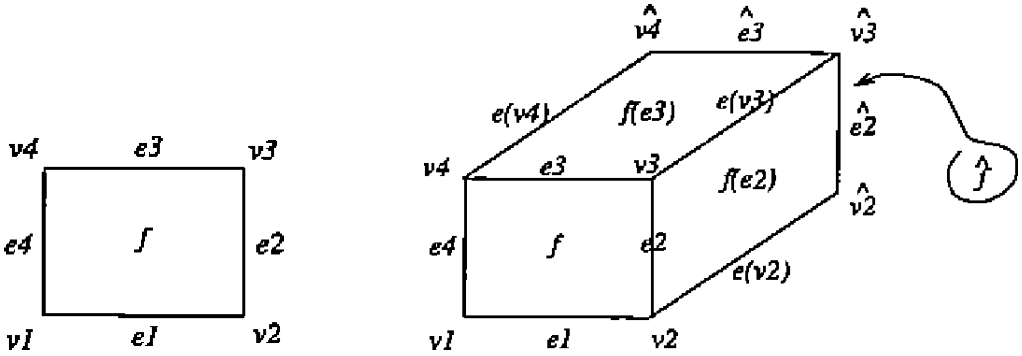
Les travaux de Kripac sont parmi les premiers à présenter une solution à la réévaluation de modèle paramétrique 3D. Il propose un système de nomination persistante, un DAG pour suivre l'évolution des faces et un mécanisme d'appariement des entités. Cependant, ce système présente des inconvénients. D'une part, chaque type d'entité topologique est associé à un schéma

de nomination spécifique, ce qui limite la généralisabilité du système à toutes les dimensions. D'autre part, l'appariement repose en partie sur le voisinage (des faces) pour estimer la similitude entre deux entités et sur un certain nombre d'heuristiques rendant le résultat parfois imprévisible. En particulier, lorsque les correspondances entre les faces du graphe ne sont pas exactes (ce qui est souvent le cas) aucune indication n'est fournie sur la façon dont ces correspondances sont représentées et gérées dans le graphe, ni sur l'impact que cela pourrait avoir sur les futurs appariements.

Capoyleas et al. (1996)

Les travaux de Capoyleas, complétés par Chen et intégrés dans le système *Erep* [HJ92 ; Che95 ; CCH96], proposent plusieurs schémas de nomination des entités topologiques basés sur les opérations :

— d'extrusion, de balayage et de révolution ;
— booléennes ;
— de modification d'entités (chanfrein, arrondi, etc.).



(a) Profil (b) Pièce extrudée

Figure 2.7 – Exemple de nomination d'entités topologique, Capoyleas

Ce système propose que les entités issues d'une extrusion soient nommées par rapport à leur ancêtre invariant dans le profil. Prenons, l'exemple illustré dans la figure 2.7 et plus particulièrement le profil (figure 2.7a) et son extrusion (figure 2.7b). Les entités du profil sont nommées par type d'entité et de manière incrémentale. Prenons, par exemple, l'arête e_1 et le sommet v_3. Suite à l'extrusion, la face opposée à f et notée $\hat{f}$ est la face d'arrivée de l'extrusion, la face de côté générée à partir de e_3 est nommée $f(e_3)$ et l'arête générée à partir de v_2 est nommée $e(v_2)$.

Capoyleas et al. distinguent deux schémas de nomination selon qu'une entité contingente, créée à l'issue d'une opération booléenne, possède une source ou non. Prenons l'exemple de la différence entre un bloc et un cylindre illustré dans la figure 2.8. Si une entité est créée avec source, cela signifie qu'elle a déjà un nom qu'elle conserve dans l'objet produit par l'opération. Ce cas est illustré par les arêtes e_5, $\hat{e_5}$ et la face $f(e_5)$ issues du cylindre et qui partagent, au moins en partie, les plongements géométriques des arêtes e_3, $\hat{e_3}$ et $f(e_3)$, respectivement. Autrement, si une entité est créée sans source (des sommets et des arêtes, en 3D), alors elles sont nommées avec les noms (non uniques) I_v pour les sommets et I_e pour les arêtes.

Enfin, si les entités sont modifiées par des opérations telles que le chanfrein ou l'arrondi, celles-ci sont nommées d'après l'opération. Par exemple, $c(e)$ et $c(l_e)$ sont les entités créées par

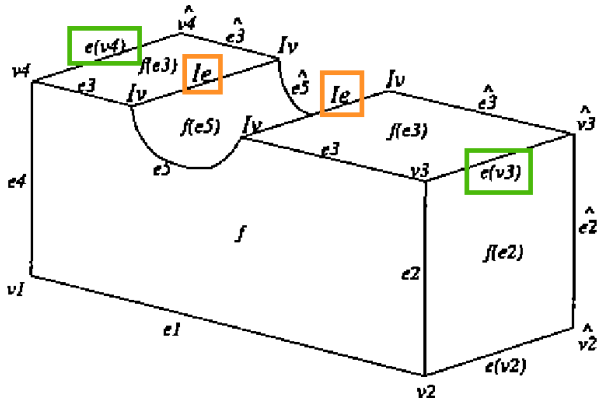


FIGURE 2.8 – Résultat d'une différence entre deux objets, Capoyleas

l'opération de chanfrein où e est une arête et l_e une liste d'arêtes. De la même manière, $a(e)$ est une entité créée par l'opération d'arrondi.

Pour résoudre les ambiguïtés dans les noms, les auteurs proposent de caractériser chacune des entités qui partagent un même nom à l'aide d'un contexte topologique, c'est-à-dire une structure décrivant le voisinage d'une entité r. Un contexte contient un ensemble de classes d'équivalences, notées d'après le schéma suivant :

$$[distance, nomEntite, nbOccurences]$$

Précisons que $nbOccurence$ représente le nombre d'entités qui partagent le même nom et la même distance dans le voisinage. Le contexte est étendu de cette manière jusqu'à trouver une classe d'équivalence unique qui permette alors de discriminer une entité dont le nom est partagé. Prenons, par exemple, les deux arêtes I_e encadrées en **orange** dans la figure 2.8. À terme, pour la première entité I_e, nous obtenons :

$$\begin{aligned} &I_e : \\ &\quad [0, I_e, 1], \\ &\quad [1, I_v, 2], [1, f(e_3), 1], [1, f(e_5), 1], \\ &\quad [2, \mathbf{e(v_4)}, 1], [2, e_3, 1], [2, e_5, 1], [2, I_e, 1], [2, \hat{e_5}, 1], [2, \hat{e_3}, 1] \end{aligned}$$

et pour la seconde :

$$\begin{aligned} &I_e : \\ &\quad [0, I_e, 1], \\ &\quad [1, I_v, 2], [1, f(e_3), 1], [1, f(e_5), 1], \\ &\quad [2, \mathbf{e(v_3)}, 1], [2, e_3, 1], [2, e_5, 1], [2, I_e, 1], [2, \hat{e_5}, 1], [2, \hat{e_3}, 1] \end{aligned}$$

Ainsi, la première arête I_e est caractérisée par l'arête $e(v_4)$ et la seconde est caractérisée par l'arête $e(v_3)$ encadrées en **vert** dans la figure 2.8.

Cependant, cette approche basée sur le voisinage topologique peut ne plus suffire dans le cas d'intersections de surfaces non-planaires. Pour répondre à cette problématique, les auteurs proposent un schéma de nomination persistante des sommets et arêtes d'après leurs faces adjacentes, leur orientation locale (dans leur voisinage) et leur orientation globale (dans l'objet).

Le nom persistant d'une arête d'intersection est :

$$E_a = [FacesIncidentes_a, OrientationLocale_a, OrientationGlobale_a]$$

où $FacesIncidentes_a$ est la liste (cyclique) des faces incidentes à E_a et où $OrientationLocale_a$ et $OrientationGlobale_a$ sont des informations géométriques qui décrivent l'orientation de E_a par rapport à ses faces adjacentes dans l'objet.

Le nom persistant d'un sommet d'intersection est :

$$E_s = [FacesIncidentes_s, OrientationLocale_s, OrientationGlobale]$$

où $FacesIncidentes_s$ est la liste (cyclique) des faces incidentes à E_s et où $OrientationLocale_s$ et $OrientationGlobale_s$ sont des informations géométriques qui décrivent l'orientation de E_s dans l'objet à condition que celui-ci soit orientable.

Synthèse

Le système de nomination persistante proposé par Capoyleas et al. est un système basé sur les faces puisque celles-ci ont nécessairement un ancêtre invariant. L'inconvénient d'un tel système, comme évoqué précédemment, est qu'il est difficile, voire impossible à généraliser pour toutes les dimensions. En dimension supérieure, les faces peuvent être des entités sans source ce qui impliquerait de définir un autre type d'entité comme base de la nomination persistante. De plus, la nomination des entités n'est pas homogène puisque les faces, contrairement aux arêtes et sommets, sont uniquement caractérisées d'après le nom de leur ancêtre, quand elles en ont un. Finalement, l'utilisation de voisinages étendus pour caractériser les entités peut avoir du sens lorsqu'il s'agit de voisins relativement proches, mais il est difficile d'imaginer le sens et l'impact que pourrait avoir l'utilisation de voisins plus éloignés sur le rejeu.

Wu et al. (2001)

Wu et al. [Wu+01] s'intéressent à la nomination des entités topologiques et plus particulièrement à leur suivi, dans le cadre de la modélisation à base d'opérations booléennes. Ces auteurs proposent un système dans lequel les noms persistants sont constitués de deux parties : la première se base sur l'identification à base topologique des faces du modèle ; la seconde est une information géométrique dont le but est de lever des ambiguïtés potentielles.

Wu et al. distinguent trois schémas de nomination : l'un pour les faces issues d'extrusions (présenté dans les concepts communs), un autre pour les faces dérivées d'entités topologiques, et un troisième pour les faces résultant d'autres caractéristiques de formes.

La nomination des faces dérivées d'entités topologiques se basent sur les on des faces adjacentes à ces entités. Par exemple l'on de la face F d'un chanfrein est déterminé ainsi :

$$on(F) = [FeatID, FeatID_{FF1}, ID_{element1}, FeatID_{FF2}, ID_{element2}]$$

où $FeatID_{FF1}$ et $FeatID_{FF2}$ sont les identifiants de deux faces adjacentes à l'arête dont est dérivée F, $FeatID_{element1}$ et $FeatID_{element2}$ les identifiants de $FF1$ et $FF2$ dans leurs on respectifs.

La nomination des faces issues d'autres caractéristiques de formes est, elle aussi, dérivée de la nomination de faces issues d'extrusion. Par exemple l'on d'une face F construite à partir d'une répétition de motif linéaire (*linear pattern*) est déterminé ainsi :

$$on(F) = [FeatID, FeatID_p, ID_{element}, FeatID_{path}, ID_{trajectory}, InsNum]$$

où seul le dernier élément $InsNum$, correspondant à un indice unique de création, est ajouté par rapport au nom décrit précédemment d'une face issue d'un profil d'extrusion. Prenons le cas d'une opération qui duplique une caractéristique de forme, par exemple un cylindre sur un bloc, le long d'une trajectoire. La valeur $InsNum$ désigne chacun des cylindres à l'aide d'un numéro d'instanciation où 0 désigne l'instance d'origine, c'est-à-dire celle qui est dupliquée. Comme nous l'avons vu précédemment, Wu et al. proposent d'associer leur système de nomination à un mécanisme de propagation des on des faces, lorsqu'elles sont scindées et fusionnées, à leurs faces descendantes.

Pour compléter la nomination et permettre la distinction des entités dont le nom est ambigu, Wu et al. proposent d'utiliser une information d'espace paramétrique PSI (pour *Parametric Space Information*). Présenté simplement, les entités portant le même nom sont ordonnancées dans un espace normalisé (u, v) puis désignées par le numéro séquentiel qui leur est attribué. À terme, le PSI d'une entité topologique est un couple :

$$[SeqNo, Total]$$

où $SeqNo$ est le numéro d'apparition d'une entité dans un espace (u, v) et où $Total$ est le nombre d'entités qui portent le même nom.

À l'issue du calcul du PSI, un nom persistant, appelé nom réel rn (pour *real name*) est utilisé pour la nomination de chacune des entités topologiques :

— Une face est nommée

$$rn(F) = [on(F), PSI]$$

— Une arête, définie par ses faces incidentes est nommée :

$$rn(E) = [rn(F1), rn(F2), PSI]$$

— Un sommet, également défini par ses faces incidentes, est nommé :

$$rn(V) = [rn(F1), rn(F2), rn(F3), PSI]$$

Synthèse

En résumé, les travaux de Wu et al. abordent le problème de la nomination persistante dans le contexte des opérations booléennes. Ils proposent un système de nomination basé sur les faces, enrichi d'informations géométriques supplémentaires pour résoudre les ambiguïtés lorsque plusieurs entités portent le même nom. Les auteurs proposent que seules les entités explicitement enregistrées dans la spécification paramétrique doivent être nommées. Ce système permet la création de noms uniques, à l'exception des cas ambigus résultant d'intersections non-planaires.

Néanmoins, cette approche propose un usage systématique des PSI, et donc de la géométrie, y compris dans des cas simples où la topologie seule peut suffire à lever l'ambiguïté, comme la scission d'une face. Les informations contenues dans les PSI peuvent, en effet, rapidement provoquer des résultats erronés en cas de modification de l'objet. De plus, l'usage des faces comme

type d'entité de base pour la nomination des autres entités est un facteur limitant. Ajouté au fait que la nomination d'une face dépend de la catégorie d'opérations qui la crée, ce système de nomination est particulièrement difficile, voire impossible, à généraliser dans d'autres dimensions.

Dans les travaux publiés plus récemment, Farjana et al. [FMH15 ; FHM16] proposent une extension des travaux de Wu et al. avec un enrichissement des noms persistants. Le problème abordé est l'ambiguïté occasionnée lorsque des entités scindées ou fusionnées partagent le même nom persistant. Farjana et al. proposent alors d'ajouter un nom secondaire, attribué pendant la réévaluation, qui est une simple mise à jour du nom persistant attribué à l'évaluation initiale. Ainsi, si une entité est issue d'une scission, son nom secondaire la distingue avec un numéro d'instanciation sur la totalité des entités scindées. Si une entité est issue d'une fusion, l'entité principale garde le même nom et le nom secondaire renseigne le nom de l'autre entité fusionnée. Malgré tout, les travaux de Farjana et al. s'appuient sur les travaux de Wu et al., et leurs travaux présentent les mêmes limitations.

Bidarra et al. (2005)

Bidarra et al. [BNB05] suggèrent que le problème de la nomination est généralement abordé avec une solution incomplète, voire contre-productive. En effet, les auteurs soulignent que les systèmes de réévaluation existants cherchent à suivre et maintenir des noms persistants pour des entités topologiques susceptibles de disparaître lors de la réévaluation. Ils avancent que procéder ainsi est incompatible avec une approche véritablement générique.

À partir de ce constat, les auteurs proposent une méthode basée sur la nomination des caractéristiques de forme et posent deux critères pour garantir le fonctionnement de cette approche :

1. les noms persistants doivent porter exclusivement sur des entités topologiques définies dans la spécification paramétrique du modèle ;
2. à chaque étape de l'évaluation, il est crucial de garantir que la sémantique des entités (c'est-à-dire, l'ensemble des contraintes qui les caractérisent dans la spécification paramétrique) demeure valide.

Ainsi, le premier critère concerne la nomination persistante d'un ensemble d'entités topologiques, limité aux paramètres explicitement définis dans la spécification paramétrique. Le second critère, quant à lui, porte sur la validité de la sémantique, qui garantit que le mécanisme d'appariement peut établir un lien entre les entités issues du modèle initial et celles du modèle réévalué.

Dans cette démarche, Bidarra et al. considèrent qu'un modèle paramétrique consiste en une corrélation entre trois classes d'entités : les *classes de référence*, les *classes de caractéristiques déclaratives* et les *classes de caractéristiques procédurales*. Elles sont définies comme suit.

Classe de références Une *référence* est définie comme une entité auxiliaire qui possède un nom unique et une structure prédéfinie (comme un plan, une ligne, un cercle, etc.). Cette entité peut être associée à n'importe quel paramètre topologique de la spécification paramétrique à travers un ensemble de contraintes. En outre, les classes de références sont des objets simples à partir desquelles il est aisé de dériver ou composer d'autres classes. Enfin, celles-ci doivent déclarer leurs propres position et orientation afin de fixer le degré de liberté d'une entité auxiliaire par rapport aux entités persistantes du modèle.

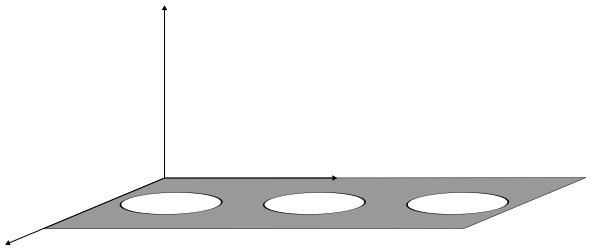


FIGURE 2.9 – Plan de référence

Afin de produire, par exemple, l'objet illustré dans la figure 2.9, il est possible de définir un plan de référence dans le modèle paramétrique sur lequel vont être instanciés une série de trous, tous positionnés relativement au plan de référence. Ainsi, un changement de position du plan implique le déplacement de chacun des trous placés dessus.

Classes de caractéristiques déclaratives Une *caractéristique déclarative* est définie comme un patron qui permet de créer des instances d'un type de caractéristique (par exemple une caractéristique de forme) à partir de la description de ses propriétés. Conceptuellement, chaque objet créé à partir d'un patron appartient ainsi à une même famille. De cette manière, tout objet qui appartient à une famille de caractéristiques possède les propriétés liées à cette famille, quels que soient ses paramètres d'instanciation. Un exemple est présenté

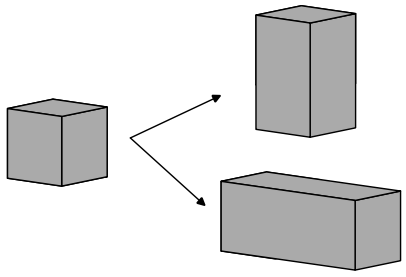


FIGURE 2.10 – Caractéristique déclarative de bloc

dans la figure 2.10, qui illustre les faces de la caractéristique de forme de bloc (appelées *faces de caractéristique*) que toute instance possède. Tout bloc dispose de six faces caractéristiques : *haut*, *bas*, *avant*, *arrière*, *gauche* et *droite*. Ainsi, pour toute instance de cette caractéristique de forme, il est possible de désigner la face avant de la manière suivante :

$$[NomBloc, FaceAvant]$$

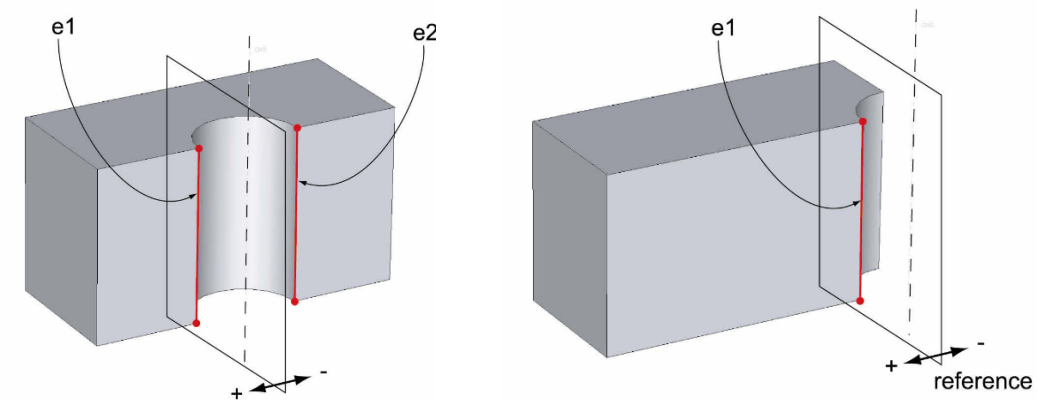
Les auteurs précisent qu'une caractéristique essentielle des caractéristiques déclaratives est que leurs faces ne sont jamais scindées, fusionnées ou supprimées, indépendamment de leur représentation géométrique.

Classes de caractéristiques procédurales Une *caractéristique procédurale* est une caractéristique dont les paramètres topologiques sont des arêtes, et non des faces, comme le chanfrein ou l'arrondi. Les auteurs justifient l'existence de cette classe par les deux propriétés suivantes : (1) contrairement aux caractéristiques déclaratives, une caractéristique procédurale dépend de l'existence de l'arête (ou des arêtes) qui lui est (sont) passée(s) en paramètre ; (2) la forme d'une caractéristique procédurale ne peut être déterminée qu'après que ses arêtes sont identifiées et localisées. Ainsi, les auteurs proposent de nommer les arêtes d'après leurs faces incidentes.

(a) (b)

Figure 2.11 – Exemple de nomination d'une arête, Bidarra et al.

Par exemple, dans la figure 2.11, Bidarra et al. illustrent comment identifier une des deux arêtes en rouge pour appliquer un chanfrein. Tout d'abord (figure 2.11a), l'arête est définie par ses faces caractéristiques incidentes, la face du dessus et la face de droite. Cette sélection filtre deux arêtes de part et d'autre de la rainure. Pour discriminer l'arête sur laquelle appliquer le chanfrein, ils proposent d'utiliser un discriminant à l'aide des faces caractéristiques aux extrémités de l'arête à chanfreiner. Dans cet exemple, il s'agit du couple face avant du bloc et face gauche de la rainure.



(a) (b)

Figure 2.12 – Résolution de l'ambiguïté d'une intersection non-planaire, Bidarra et al.

Toutefois, Bidarra et al. indiquent que cette méthode d'identification des arêtes devient inopérante dans le cas d'une intersection où au moins l'une entités n'est pas planaire (par exemple la face de côté d'un cylindre). Le discriminant (et donc le voisinage topologique) ne suffit plus pour désigner une arête parmi celles issues de l'intersection puisqu'elles ont toutes le même discriminant, comme dans l'exemple illustré figure 2.12 Pour résoudre ce problème, ils proposent alors d'utiliser un plan de référence pour découper la caractéristique déclarative en deux demi-espaces. De cette manière, chaque arête se voit assigner un discriminant supplémentaire qui est le domaine de référence auquel il appartient, positif ou négatif. Par exemple, dans là figure 2.12a, les arêtes $e1$ et $e2$ sont toutes les deux incidentes aux mêmes faces caractéristiques. Néanmoins, le demi-espace défini le long de la rainure assigne $e1$ au domaine positif et $e2$ au domaine négatif. Cette discrimination permet alors, lors de la réévaluation (figure 2.12b), de ne pas appliquer de chanfrein sur $e1$ si $e2$ était l'arête définie comme paramètre dans la spécification paramétrique.

Synthèse

Ainsi, ces travaux définissent trois classes de caractéristiques que les auteurs utilisent pour développer différents concepts destinés à apporter de la stabilité à la nomination persistante des entités et garantir leurs appariements lors de la réévaluation d'une spécification paramétrique. Toutefois, cette approche reprend indirectement les concepts communs que l'on trouvait déjà dans la plupart des approches précédentes. Les classes de caractéristiques déclaratives, par exemple, définissent les faces invariantes des caractéristiques de formes. Le fait de caractériser les arêtes à partir des faces incidentes et des faces adjacentes aux sommets revient à utiliser les voisinages topologiques, comme le faisaient déjà Kripac et Capoyleas. L'utilisation d'un plan de référence est une autre manière d'utiliser une information géométrique permettant de lever les ambiguïtés lorsque la topologie ne permet plus de caractériser de manière unique une entité.

Par ailleurs, cette approche présente plusieurs limitations. Le système de nomination s'appuie sur des faces invariantes. Bien que ces faces soient persistantes lors de la réévaluation et immuables, car définies exclusivement à partir de caractéristiques de forme, ce système est spécifiquement adapté à la modélisation 3D. Ainsi, il n'est pas généralisable à d'autres dimensions, où les faces peuvent résulter d'intersections de volumes. Une autre limitation est l'usage d'un plan pour la discrimination des différentes arêtes issues d'une intersection avec une entité non-planaire. En effet, un plan ne permet de scinder efficacement l'espace que pour un échantillon de structures compatibles avec leurs classes de caractéristiques, à savoir les plans, les cylindres, les sphères et les tores.

Cardot et al. (2019)

Dans la littérature, les travaux de Cardot [Car19] sont les premiers à proposer une solution à la nomination persistante basée sur des règles de transformation de graphe. Ses contributions comprennent la spécification d'un système de nomination, une historisation de l'évolution des entités référencées dans une spécification paramétrique et l'édition d'une spécification paramétrique avec l'appariement des entités topologiques. Dans le cadre de ses travaux, Cardot tire parti des cartes généralisées [Lie94 ; DL14] pour formaliser les objets n-dimensionnels et les représenter. Elle a également recours aux règles Jerboa [Bel+14] pour formaliser les opérations de modélisation géométrique.

Dans la première partie de ses travaux, Cardot propose un système de nomination en deux couches : (1) création d'un identifiant persistant pour tout brin d'une G-carte ; (2) création d'un nom persistant à partir d'un ensemble minimal d'identifiants persistants pour chaque entité topologique, ici une orbite au sens des cartes généralisées, référencée dans la spécification paramétrique.

L'identification persistante des brins est justifiée par les propriétés suivantes :

1. un brin est l'entité topologique de plus petite dimension, et donc indivisible, dans une G-carte ;
2. un brin créé par une règle ne peut qu'être créé *ex nihilo* ou par copie d'un autre brin.

Avec ces propriétés en tête, Cardot propose de définir un identifiant persistant (noté *PI*) basé sur l'historique d'un brin, c'est-à-dire l'ensemble des règles et de leurs nœuds qui expriment de manière unique la création d'un brin dans une G-carte. Un identifiant est alors défini comme

une collection de couples $\{numeroEtap - noeud\}$, où $numeroEtape$ est le numéro d'application d'une règle dans la spécification paramétrique et $noeud$ est le nom d'un nœud dans cette même règle.

À sa création *ex nihilo*, par exemple suite à l'application de la toute première règle de la spécification paramétrique, l'identifiant persistant d'un brin est $\{1 - noeud\}$ et chacune des opérations qui réécrivent ce brin, ajoute un couple $\{i - noeud\}$ à son identifiant persistant. Enfin, l'identifiant persistant d'un brin créé par copie est aussi l'identifiant persistant du brin copié auquel s'ajoute sa propre histoire.

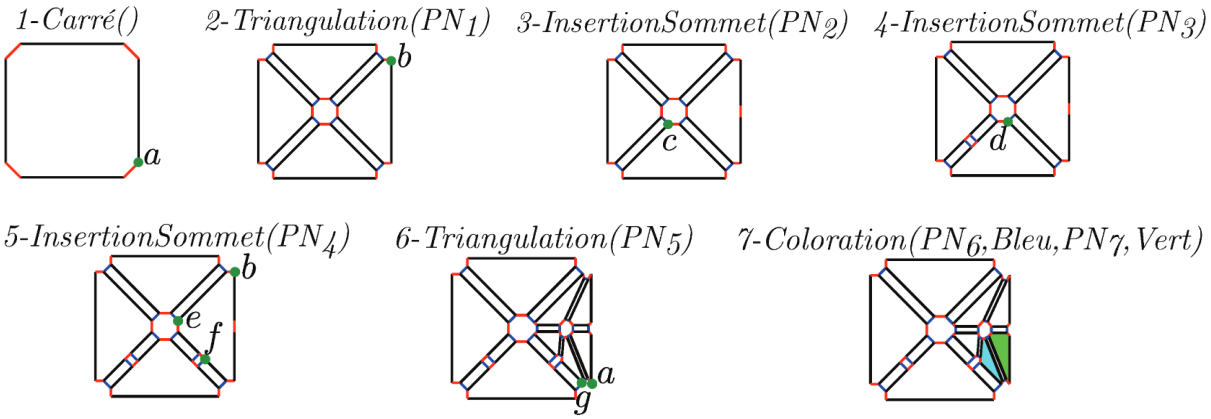
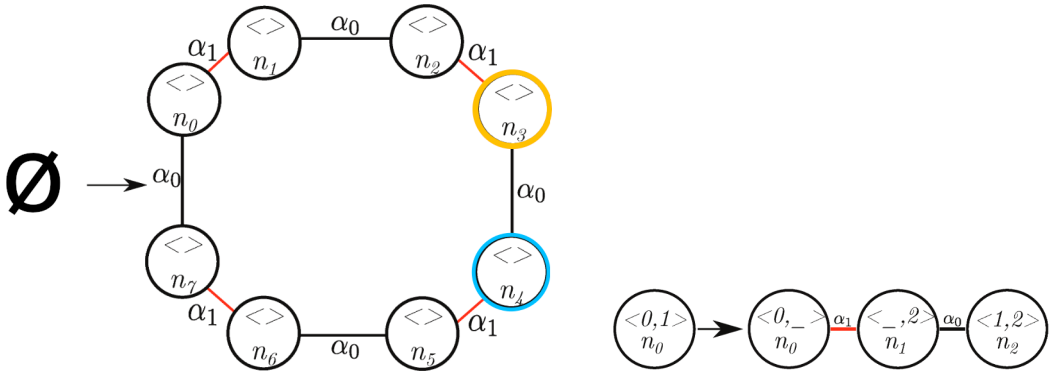


Figure 2.13 – Exemple de construction, Cardot



(a) Création de face carrée (b) Triangulation de face

Figure 2.14 – Règles de l'exemple figure 2.13

Dans l'exemple de construction illustré figure 2.13, les brins a et b sont tous les deux créés par la première règle de création d'une face carrée (figure 2.14a) et plus spécifiquement par les nœuds n_4 (en **bleu**) et n_3 (en **jaune**) respectivement. Leurs identifiants persistants sont $PI_{a1} = \{1 - n_4\}$ et $PI_{b1} = \{1 - n_3\}$. Dans la deuxième étape de la construction, a et b sont réécrits à l'identique par le nœud n_0 de la règle de triangulation (figure 2.14b). Leurs identifiants persistants sont donc mis à jour en $PI_{a2} = \{1 - n_4; 2 - n_0\}$ et $PI_{b2} = \{1 - n_3; 2 - n_0\}$.

Une fois les brins dotés d'un identifiant persistant, Cardot propose de les utiliser pour nommer, de manière unique, les orbites auxquelles ils appartiennent. De cette manière, chaque brin représente une partie des changements subis par les orbites auxquelles il appartient. Le nom persistant d'une orbite (noté *PN*) est donc défini comme un ensemble d'identifiants persistants.

Un nom persistant est *minimal* lorsqu'un nombre minimum d'identifiants persistants énumèrent l'ensemble des règles qui ont impacté une orbite donnée.

Ainsi, toujours dans l'exemple figure 2.13, la deuxième règle d'application a pour paramètre le nom persistant PN_1 qui désigne la face carrée. D'après la méthode de Cardot, puisque tous les brins de la face sont créés par la même règle, l'identifiant persistant du brin a après la première opération, PI_{a1}, est ajouté au nom persistant PN_1. De même, la troisième opération, une insertion de sommet, est appliquée sur une arête désignée par le nom persistant PN_2, dont les identifiants persistants sont $PI_{a2} = \{1 - n_4; 2 - n_0\}$ et $PI_{b2} = \{1 - n_3; 2 - n_0\}$. Ces deux identifiants persistants décrivent la même succession d'opérations sur la même face, ils sont donc **équivalents**. En cas d'équivalence, Cardot propose d'utiliser l'identifiant du brin désigné par l'utilisateur dans le nom persistant (ici PI_{b2}). Le nom persistant dépend donc du brin sélectionné par l'utilisateur lors de l'évaluation initiale.

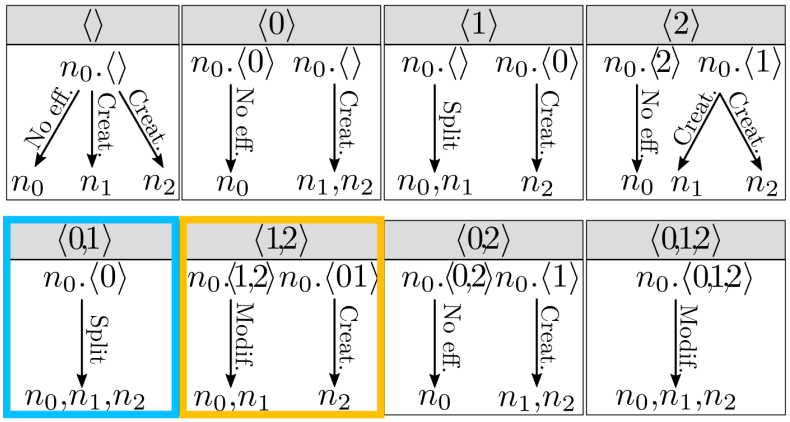


FIGURE 2.15 – Journal de bord de la triangulation, Cardot

Pour permettre l'appariement des entités topologiques, Cardot utilise un **journal de bord**, c'est-à-dire une structure qui référence pour chaque règle Jerboa l'ensemble des changements topologiques qui interviennent sur les entités lors de l'application d'une règle. Les changements classifiés sont la création, la suppression, la scission, la fusion, le non-changement et la modification d'entités topologiques.

Considérons le journal de bord de la règle de triangulation illustré dans la figure 2.15. Nous constatons, par exemple, dans l'entrée de type `face` (orbite $\langle 0, 1\rangle$) (encadrée en **bleu**), que la face filtrée par la règle est scindée (« *split* ») en autant de parties qu'il y avait d'arêtes (orbite $\langle 0\rangle$) dans la face initiale. Si l'on considère l'entrée de type `sommet` $\langle 1, 2\rangle$ (encadrée en **jaune**), nous observons deux événements : d'une part, la modification (« *Modif.* ») des sommets (orbite $\langle 1, 2\rangle$) de la face initiale après la triangulation ; et d'autre part, la création (« *Creat.* ») du sommet central de la face triangulée à partir de la face (orbite $\langle 0, 1\rangle$) initiale.

A la fin du jeu initial, les journaux de bord des règles sont consultés pour construire des *journaux d'historique* permettant de tracer l'évolution, au cours du processus de modélisation, des entités topologiques référencées via les noms persistants. Un journal d'historique est construit en remontant progressivement l'historique d'un identifiant persistant. Prenons le journal d'historique de $PI_{7a} = 1 - n_4; 2 - n_0; 3 - n_0; 6 - n_0$, l'identifiant persistant correspondant au brin utilisé dans PN_7 pour désigner la face à colorier en vert à la septième étape de la spécification paramétrique. Ce journal d'historique est illustré dans la figure 2.16. Dans cet exemple, le journal d'historique remonte progressivement l'historique des évolutions de la face (orbite $\langle 0, 1\rangle$) à

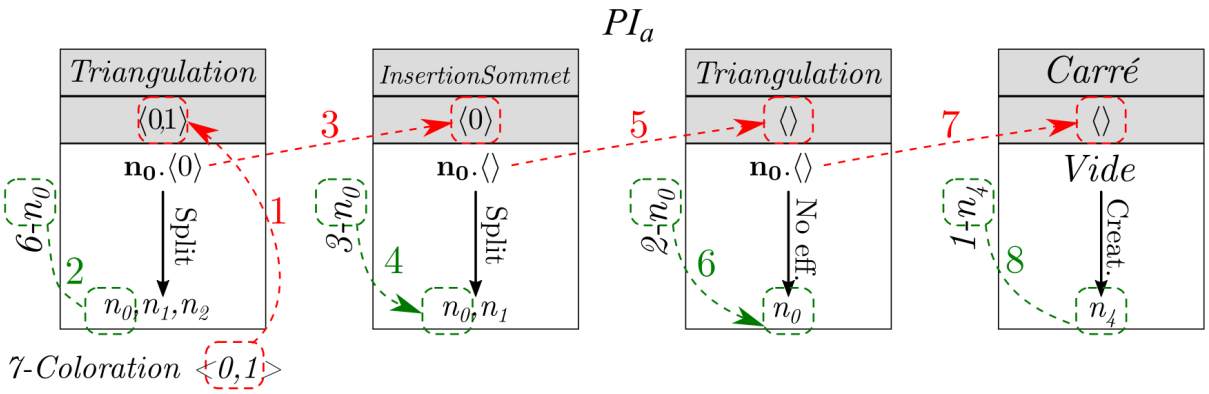


Figure 2.16 – Journal d'historique de PI_{7a}, Cardot

colorier en suivant l'identifiant persistant du brin a. Le journal se lit de la manière suivante : (1) l'orbite suivie est une face à colorier ; (2) la face est scindée par la sixième opération (*Triangulation*) ; (3) elle est identifiée par une arête (orbite $\langle 0\rangle$) ; (4) cette arête est scindée par la troisième opération (*Insertionsommet*) ; (5) elle est identifiée par un brin unique (orbite $\langle\rangle$) ; (6) ce brin est réécrit à l'identique par la deuxième opération (une autre *Triangulation*) ; et (7) ce brin est créé avec le carré.

Enfin, dans ses travaux, Cardot propose d'étendre l'édition d'une spécification paramétrique non plus à la seule édition des paramètres géométriques, mais également à l'ajout, la suppression et le déplacement d'opérations dans une spécification paramétrique. Cette extension permet de générer une plus grande diversité de modèles paramétriques. Trois types d'éditions supplémentaires sont alors définis sur les opérations, c'est-à-dire l'ajout, la suppression et le changement d'ordre des opérations dans une spécification paramétrique.

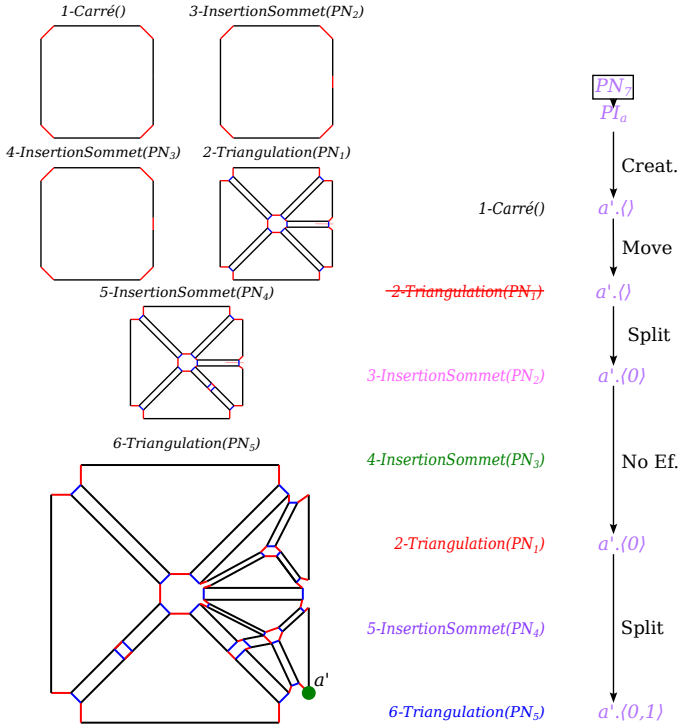


Figure 2.17 – Arbre d'appariement de PI_{7a}, Cardot

Une fois la spécification paramétrique éditée, lors de la réévaluation, les journaux d'historiques sont mis à jour en fonction des modifications apportées. Un DAG (graphe orienté acyclique) est ensuite construit étape par étape lors du rejeu à partir de ces journaux d'historiques. Reprenons l'exemple de l'identifiant persistant PI_{7a} dont l'arbre d'appariement est illustré dans la figure 2.17. L'arbre construit représente l'instanciation du journal d'historique sur l'objet réévalué et intègre les différents évènements survenus sur les orbites composant ce journal. Suite à l'édition de la spécification paramétrique, la triangulation (barrée dans la figure) est déplacée pour être appliquée après `4-InsertionSommet(PN3)`, juste avant la seconde triangulation. À cause de ce déplacement d'opération, `4-InsertionSommet(PN3)` n'est pas appliquée car l'arbre d'appariement de PN_3) permet de détecter la non-création de l'arête générée par la triangulation déplacée. Au moment de l'application `2-Triangulation(PN1)`, l'arête de droite du carré étant scindée en deux, la triangulation qui produisait une face sur le bord droit de l'objet au jeu initial en produit cette fois deux. Enfin, la seconde triangulation `6-Triangulation(PN5)` scinde la face incidente au brin a. Le dernier nœud de cet arbre $a'.\langle 0, 1\rangle$ désigne la face incidente au brin a', apparié grâce à son historique d'évolution, qui est l'un des paramètres topologiques de l'application `7-Coloration(PN6, Bleu, PN7, Vert)`.

Synthèse

Les travaux de Cardot proposent une extension du problème de la nomination persistante aux règles de transformation de graphe. Grâce aux formalismes des G-cartes et des règles Jerboa, elle propose un système de nomination persistante qui permet l'identification de toute entité topologique de manière unique, avec un schéma homogène et généralisable en toutes dimensions. Cette contribution apporte également une extension des spécifications paramétriques afin d'ajouter, de supprimer et de déplacer des opérations en plus des habituelles modifications de paramètres géométriques. L'apport du journal d'historique, basé sur un accès statique à un journal de bord pour accéder à tous les changements par orbite et réaliser le suivi des entités topologiques, est intéressant. Néanmoins, ils doivent être construits manuellement pour chaque règle et ils omettent d'intégrer une origine (et leurs propres suivis) pour les entités suivies, ce qui peut conduire à un appariement erroné des entités topologiques lors de la réévaluation et à plus forte raison lorsque des opérations ont été ajoutées, supprimées ou encore déplacées dans l'édition de la spécification paramétrique. Notons aussi que dans ses travaux, Cardot étudie uniquement des opérations élémentaires (règles simples) et non des opérations complexes, c'est-à-dire des opérations dont la structure interne fait usage de boucles ou d'alternatives. De plus, comme l'indique Cardot, la méthode proposée est « purement théorique » et n'a pas été implantée [Car19] (page 117). Il est difficile dans ce contexte de mesurer l'impact réel de cette approche et la pertinence du mécanisme de nomination sur des cas concrets.

Cascaval et al. (2023)

Dans leurs travaux [CBS23], Cascaval et al. proposent de répondre au problème de la nomination persistante dans les opérations de modélisation en se basant sur les concepts suivants.

- Un modèle paramétrique est une fonction prenant des paramètres topologiques et géométriques.
- Un nom persistant peut être obtenu grâce à une requête sur une entité topologique.
- Les types géométriques traités sont les points, segments (lignes ou arcs), polygones (conceptuellement des faces sans trou) et faces (éléments bordés par un ou plusieurs

polygones).

Cascaval et al. présentent les propriétés qu'un langage conçu pour la CAO devrait, selon eux, respecter. Premièrement, il devrait permettre la sélection de tout ou partie d'un ensemble d'entités topologiques résultant de la partition d'une entité donnée en entrée d'une opération. Deuxièmement, le langage devrait également être capable de saisir l'intention de conception de l'utilisateur en fournissant des moyens d'exclure certaines configurations topologiques indésirables dans les résultats produits par le programme.

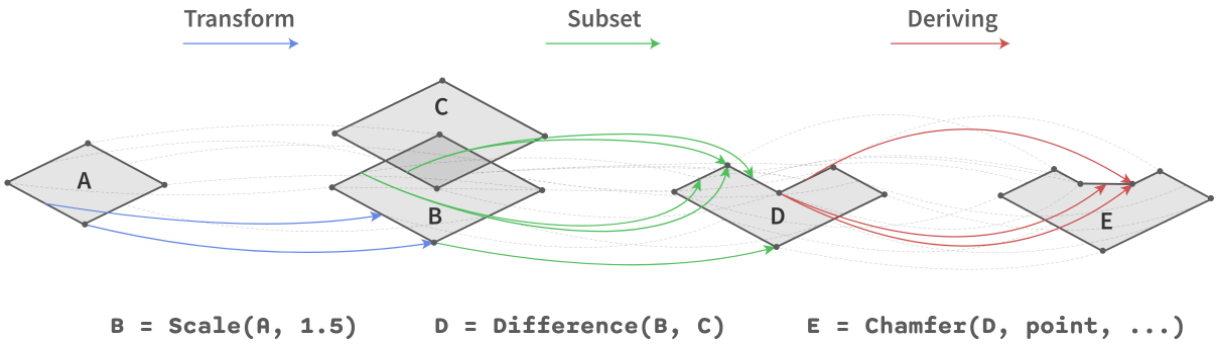


Figure 2.18 – Types de relations pour le suivi des entités, Cascaval et al.

Dans ce contexte, Cascaval et al. proposent un langage spécifique à la modélisation paramétrique qui met en œuvre un système de généalogie permettant de décrire les relations ancêtre/descendant entre plusieurs entités topologiques en entrées et en sortie d'une opération de modélisation. À cette fin, trois relations sont définies (figure 2.18) entre les entités topologiques en entrée et en sortie des opérations de modélisation : (1) transformation (flèches bleues) ; (2) sous-ensemble (flèches vertes) ; (3) dérivation (flèches rouges).

1. La relation de *transformation* définit un lien direct entre deux entités, une en entrée et l'autre en sortie, dans le cadre d'une opération qui ne modifie pas la topologie de l'objet. Cette relation est caractérisée par l'existence d'une bijection entre les éléments en entrée et en sortie ; par exemple, dans des opérations telles que la translation ou la rotation.
2. La relation de *sous-ensemble* établit un lien entre deux entités topologiques en entrée et en sortie d'opérations ensemblistes telles que les opérations booléennes. Il s'agit, typiquement dans les opérations booléennes, de faire le lien entre, par exemple, deux faces et l'arête issue de l'intersection de ces deux faces.
3. La relation de *dérivation* établit un lien entre une entité topologique créée et la ou les entités qui en sont à l'origine. Dans l'exemple du chanfrein de sommet illustré dans la figure 2.18, un lien de dérivation est créé entre le sommet de D et l'arête créée sur E. De la même façon, des liens de dérivation sont créés entre ce sommet et les deux sommets créés sur E à l'extrémité de l'arête du chanfrein.

   Ces deux sommets ayant, de fait, exactement les mêmes liens de dérivation, Cascaval et al. posent deux contraintes sur la construction de ces liens : (1) Les relations de dérivation entre les entités en entré et sortie d'une opération sont exprimées de manière statique, de sorte que toute entité créée dans la dérivation puisse être désignée de manière unique par une ou plusieurs entités passées en entrée ; (2) Une relation de dérivation entre une entité créée dans la dérivation et son ancêtre doit être telle que la modification de son ancêtre a nécessairement un impact sur son entité descendante. Afin de vérifier la première contrainte, ils proposent donc d'ajouter un lien de dérivation entre les arêtes adjacentes au sommet chanfreiné et les sommets correspondants sur E.

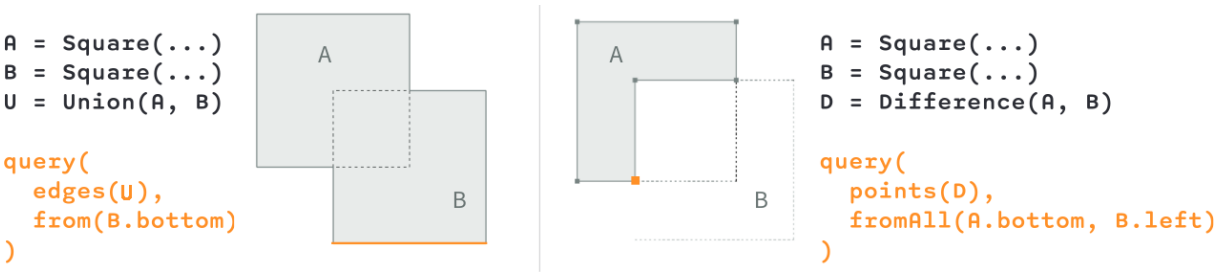


FIGURE 2.19 – Requêtes de filtrage par Cascaval

Afin de saisir les intentions de conception, les noms persistants sont définis par les utilisateurs pendant la conception du modèle paramétrique, à l'aide de requêtes, comme illustré dans la figure 2.19. Ces requêtes permettent aux utilisateurs de contraindre l'appariement des entités topologiques passées en paramètres des opérations. Les entités sont alors identifiées par rapport à un ou plusieurs ancêtres qui sont explicitement désignés comme paramètres dans les requêtes. Il est possible, par exemple, de désigner toutes les entités de toute une construction, (`all`), qui font partie d'une autre entité e (`from(e)`) ou alors plusieurs (`fromAll((e|e)*)`, voire de spécifier son appartenance à une entité e dans un ensemble (`fromAny((e | {e})*)`). Il est également possible d'organiser les requêtes dans des critères logiques (`and(q,q)`) et (`orq,q`), d'en imposer (`contains(q)`), voire d'en exclure (`not(q)`). Enfin, l'utilisateur peut s'appuyer sur les relations de transformation, de sous-ensemble et de dérivation pour exprimer au mieux ses intentions de modélisation, par exemple `deriveFrom|All|Any`.

Dans les cas topologiquement ambigus, c'est-à-dire où il devient difficile d'identifier formellement une entité parmi plusieurs, comme l'identification d'un sommet lorsque deux cercles s'intersectent, les auteurs ont recours à des contraintes géométriques semblables à l'approche de Bidarra [BNB05] avec, par exemple, la mise en œuvre d'un demi-espace pour discriminer plusieurs entités symétriques.

Synthèse

Dans cette contribution, Cascaval et al. proposent un langage spécifique pour la conception de modèles paramétriques. Ce langage intègre un système de nomination persistante des entités où les noms sont donnés explicitement par l'utilisateur sous forme de requêtes. Pour cela, le modèle est enrichi, d'une part, de trois types de relation qui permettent de garder une trace de l'évolution des entités topologiques par rapport à leurs ancêtres et, d'autre part, d'informations géométriques lorsque les ancêtres et la topologie ne suffisent plus. Néanmoins, cette approche présente plusieurs inconvénients :

— il s'agit d'un système de modélisation 2.5D, c'est-à-dire que les opérations sont essentiellement définies pour la conception 2D avec deux opérations d'extrusion et de révolution pour passer à la 3D. Autrement dit, généraliser les systèmes de contraintes et d'appariement en toutes dimensions est au minimum difficile. Par exemple, les ambiguïtés topologiques et géométriques ne nécessitent pas les mêmes traitements, voire se complexifient lorsqu'il s'agit de monter en dimension ;
— les opérations de modélisation sont explicitement responsables de la définition et de la création des relations de transformation, de sous-ensemble et de dérivation. En particulier, il convient de définir manuellement ces relations tout en garantissant les contraintes posées par le développeur pour éviter les ambiguïtés ;
— la nomination des entités est à la charge des utilisateurs afin que ceux-ci expriment leurs

intentions de modélisation. Pour cela, les auteurs proposent une nomination sous la forme de requêtes intégrant des contraintes qui permettent de filtrer certaines parties de l'objet : par exemple, une arête ou un sommet créés par l'union ou la différence de deux faces, comme illustré dans la figure 2.19. Le problème de cette approche est que ces contraintes sont exprimées manuellement par l'utilisateur pour lequel plus un objet se complexifie, moins il a une idée précise des contraintes à intégrer, *a priori*. Il est donc difficile de prévoir et garantir la persistance d'un nom avant l'édition et la réévaluation d'une spécification paramétrique, c'est-à-dire avant l'apparition des problèmes d'appariement.

5 Conclusion

Dans ce chapitre, nous avons commencé par présenter les outils que nous utiliserons tout au long de cette thèse pour formaliser les objets et les opérations de modélisation géométrique. Puis, nous avons passé en revue un état de l'art des systèmes de réévaluation proposés dans la littérature et des problèmes qu'ils tentent de résoudre. Parmi ceux-ci se trouvent la nomination persistante des entités topologiques, leurs appariements dans la réévaluation d'une spécification paramétrique éditée et, parfois, l'appariement d'entités pour lesquelles la topologie est devenue insuffisante pour les distinguer les unes des autres.

Nous avons constaté qu'une grande partie de ces approches ont développé un système de nomination persistante en utilisant les faces comme entités de base pour nommer les autres. Ce choix est compréhensible pour un modeleur spécialisé dans la modélisation 3D. Dans un tel modeleur, une face est une entité créée comme une forme de base (entité caractéristique, profil d'extrusion, etc.), ou bien à partir d'une ou plusieurs autres faces dans des opérations booléennes (intersection de volumes, de faces co-planaires). Dans un tel modeleur, une face est une entité soit créée de manière invariante (faces d'une caractéristique de forme, faces issues d'une construction par extrusion à partir d'un profil invariant, etc.), soit générée par scission, fusion ou modification de faces existantes suite à l'application d'une opération de modélisation (opération booléenne, triangulation, etc.). Dans le premier cas la face est invariante, dans le second elle est toujours avec source, ce qui permet de suivre aisément son évolution tout au long d'un processus constructif. Néanmoins, ces méthodes ont pour inconvénient d'être relativement rigides et d'inciter à développer un schéma de nomination différent pour chaque entité topologique. Par conséquent, il est difficile de généraliser de telles méthodes pour des modeleurs qui travaillent avec des dimensions supérieures à la 3D.

Certains systèmes proposent d'intégrer des informations purement géométriques afin de garantir la distinction des entités topologiques partageant un même nom basé sur leurs topologies respectives. Certaines approches proposent une distinction basée sur l'orientation des arêtes et sommets dans l'objet modélisé ou encore de classer les entités en fonction de leur ordre d'apparition dans un espace paramétrique (u, v). D'autres systèmes proposent des solutions qui se distinguent par la création d'un langage spécifique au développement de modèles paramétrique avec une nomination persistante définie comme un jeu de contraintes ou encore la formalisation d'un objet et des opérations de modélisation géométrique pour proposer un suivi de tout type d'entité et un schéma de nomination persistante générique. Le tableau 2.1 synthétise les concepts employés par différents systèmes de réévaluation. Celui-ci met en évidence plusieurs éléments concernant la nomination et la construction des historiques. En effet, la majorité des systèmes propose une nomination qui n'est ni homogène en fonction de la dimension des entités topo-

Approches / Concepts	Kripac	Capoyleas et al.	Wu et al.	Bidarra et al.	Cardot et al.	Cascaval et al.
Type d'édition : paramètres géométriques opérations	 ✓ ✗	 ✓ ✗	 ✓ ✗	 ✓ ✗	 ✓ ✓	 ✓ ✓
Base de la nomination	face	face	face	face	brin	requête
Nomination homogène	✗	✗	✗	✗	✓	✓
Suivi local des cellules	✗	✗	✗	✗	✓	✗
Historique complet d'une Cellule	✗	✗	✗	✗	✗	✓
Calcul des évènements : • Statique • Automatique	 ✗ ✓	 ✗ ✓	 ✗ ✓	 ✗ ✓	 ✓ ✗	 ✓ ✗
Dimensions de modélisation	3D	3D	3D	3D	n-D	2.5D
Résolution d'ambiguïté topologique	✓	✓	✓	✓	✓	✓
Résolution d'ambiguïté géométrique	✓	✓	✓	✓	✗	✓
Tout type d'opération	✓	✓	✓	✗	✗	✗

TABLE 2.1 – Synthèse des systèmes de réévaluation

logiques, ni générique en toutes dimensions. De plus, les historiques sont généralement créés de manière systématique pour toutes les entités d'un objet et ne représentent que partiellement leurs évolutions. Enfin, bien que le coût de l'analyse dynamique d'un modèle augmente au fur et à mesure que celui-ci évolue, aucun système ne propose un calcul des évènements qui soit à la fois statique et automatique. Comme nous pouvons le constater, l'approche proposée par Cardot répond en partie à certains de ces points, mais aucune approche n'y répond de manière globale.

Dans cette thèse, nous étendons entre autres les travaux de Cardot afin, d'une part, de proposer un calcul des évènements qui soit à la fois statique et automatique et, d'autre part, d'enrichir le suivi des changements topologiques qui impactent les entités topologiques enregistrées dans une spécification paramétrique permettant ainsi de construire des historiques complets. Dans la suite de ce manuscrit, nous proposons quatre contributions. La première est une méthode qui permet la détection automatique des changements topologiques pour toutes entités topologiques statiquement à partir des seules règles Jerboa. Notre deuxième contribution est la mise en œuvre d'un système de réévaluation basé sur un historique complet d'évolution intégrant les origines et le suivit de chaque entité référencée dans la spécification. Notre troisième contribution concerne l'implémentation et l'intégration dans l'application Jerboa de l'ensemble des mécanismes proposés. Enfin, nous étendons ce système de réévaluation à des scripts qui permettent de manipuler des règles Jerboa à l'aide de structures de contrôle et plus particulièrement des itérations de sous-orbites et des structures conditionnelles.

CHAPITRE 3

Formalisation des évènements

Sommaire

1 Introduction

Aujourd'hui, les noyaux des modeleurs 3D proposent de suivre l'évolution des entités topologiques dans un objet. Ces évolutions sont représentées par les changements topologiques, ou plus simplement *évènements*, que les opérations de modélisation appliquent aux différentes entités. Ces évènements, qui diffèrent selon les outils, peuvent être catégorisés. OCAF [Ope], par exemple, propose de catégoriser la création de primitives, la génération, la modification, la destruction ou encore l'absence d'évolution d'une entité topologique. D'autres outils, comme CATIA [Dasa] proposent de suivre la création, la modification, la fusion, la subdivision et la préservation d'entités topologiques.

Pour effectuer le suivi des entités et enregistrer les évènements, ces outils ont deux options. Soit ils effectuent un parcours complet de l'objet pour mettre chaque entité à jour, soit les évènements sont explicitement codés dans les opérations de modélisation. Dans le premier cas, bien que l'approche soit exhaustive, le coût en temps d'exécution augmente avec la complexité de l'objet modélisé. Dans le second cas, le coût en temps d'exécution est instantané, mais des erreurs peuvent être introduites lors de la conception de l'opération. Il existe en effet le risque de référencer le mauvais évènement pour la mauvaise entité topologique, voire d'en oublier.

Dans ce chapitre, nous présentons notre approche qui consiste à formaliser les opérations de modélisation géométrique à l'aide de règles Jerboa. Ainsi, nous tirons avantage de ce formalisme dont l'analyse syntaxique permet de détecter les évènements appliqués à chaque entité topologique représentée dans une règle. Dans la section 2, nous présentons la liste exhaustive des évènements que nous proposons de suivre et les formalisons dans le contexte des G-cartes. Dans la section 3, nous formalisons ces évènements dans le contexte des règles de transformation de graphe afin de caractériser les conditions suivant lesquelles chaque évènement est détecté. Dans la section 4, nous étendons la détection des évènements aux règles Jerboa, notamment avec le concept d'origine d'une entité. Enfin, après avoir présenté des exemples de détection d'évènements, nous concluons dans la section 5.

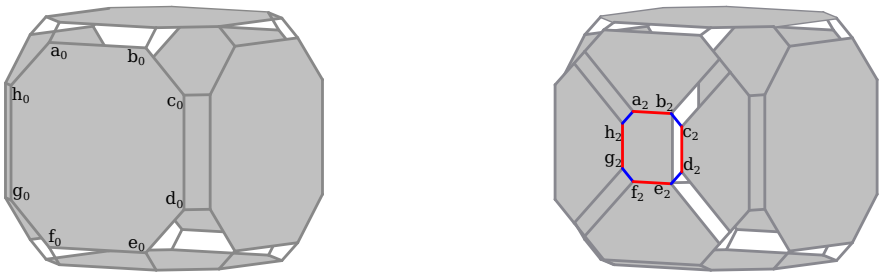
2 Les évènements dans les transformations d'objets

Considérons deux G-cartes G et H. Nous voulons caractériser les modifications d'orbites de type $\langle\omega\rangle$ entre G et H. Ces dernières peuvent être créées, supprimées, scindées, fusionnées, inchangées ou modifiées. Nous parlons dans ce cas d'événements survenus sur les orbites.

2.1 Création

Nous commençons avec l'évènement de création. Intuitivement, une orbite est créée lorsque celle-ci n'existait pas dans un objet avant sa transformation.

Définition 24 (Création d'orbite)
Soient G et H deux G-cartes de dimension $n \in \mathbb{N}$, $\langle\omega\rangle$ un type d'orbite de dimension n et v un brin de H.
L'orbite $H\langle\omega\rangle(v)$ est créée *si et seulement si aucun élément de l'orbite $\langle\omega\rangle$ incidente au brin v dans H n'existait dans G, i.e. $G \cap H\langle\omega\rangle(v) = \emptyset$.*



(a) G (b) H

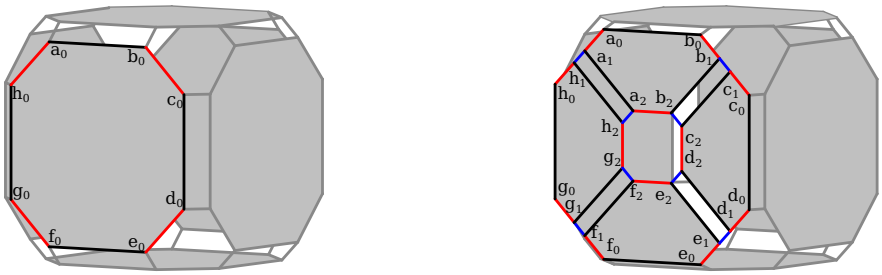
FIGURE 3.1 – Création d'un sommet dans une face

L'orbite sommet $H\langle 1,2,3\rangle(a_2)$ est créée à l'issue de la transformation du graphe G vers H (voir figure 3.1). En particulier, aucun des brins $a_2, b_2 \ldots h_2$ de H n'existent dans G.

2.2 Scission

Nous présentons la scission comme l'évènement qui divise une orbite en plusieurs sous-orbites de même dimensions.

Définition 25 (Scission d'orbite)
Soient G et H deux G-cartes de dimension $n \in \mathbb{N}$, $\langle\omega\rangle$ un type d'orbite de dimension n et v un brin de G.
L'orbite $G\langle\omega\rangle(v)$ est scindée *si et seulement si il existe deux brins v_1 et v_2 de $G\langle\omega\rangle(v)$ tels que v_1 et v_2 sont aussi des brins de H et qu'ils appartiennent à des orbites $\langle\omega\rangle$ différentes, i.e. $H\langle\omega\rangle(v_1) \neq H\langle\omega\rangle(v_2)$.*



(a) G (b) H

FIGURE 3.2 – Scission d'une face

Reprenons l'exemple figure 3.1 La face $G\langle 0,1,3\rangle(a_0)$ est scindée en quatre faces distinctes : $H\langle 0,1,3\rangle(a_0)$, $H\langle 0,1,3\rangle(c_0)$, $H\langle 0,1,3\rangle(e_0)$, $H\langle 0,1,3\rangle(g_0)$

2.3 Suppression

Intuitivement, la suppression est l'inverse de la création. Une orbite est donc supprimée lorsqu'elle celle-ci n'existe plus dans un objet après sa transformation.

Définition 26 (Suppression d'orbite)
Soient G et H deux G-cartes de dimension $n \in \mathbb{N}$, $\langle\omega\rangle$ un type d'orbite de dimension n et v

un brin de G.
L'orbite $G\langle\omega\rangle(v)$ est supprimée *si et seulement si aucun élément de l'orbite $\langle\omega\rangle$ incidente à v dans G n'existe dans H, i.e. $H \cap G\langle\omega\rangle(v) = \emptyset$.*

Reprenons l'exemple de la figure 3.1, cette fois de H vers G. Le sommet $H\langle 1, 2, 3\rangle(a_2)$ est supprimé dans G car aucun des brins $a_2, b_2 \dots h_2$ dans H n'existent dans G.

2.4 Fusion

La fusion est l'évènement inverse de la scission. Ainsi, une orbite est fusionnée lorsque plusieurs orbites distinctes appartiennent à une orbite de même type dans un objet après sa transformation.

Définition 27 (Fusion d'orbites)
Soient G et H deux G-cartes de dimension $n \in \mathbb{N}$, $\langle\omega\rangle$ un type d'orbite de dimension n et v et v' deux brins de G.
Les orbites distinctes $G\langle\omega\rangle(v)$ et $G\langle\omega\rangle(v')$ sont fusionnées *si et seulement si il existe deux brins respectivement v_1 de $G\langle\omega\rangle(v)$ et v_2 de $G\langle\omega\rangle(v')$ tels que v_1 et v_2 sont des brins de H de la même orbite $\langle\omega\rangle$, i.e. $H\langle\omega\rangle(v_1) = H\langle\omega\rangle(v_2)$.*

Reprenons l'exemple de la figure 3.1, toujours de H vers G. Les quatre faces $H\langle 0, 1, 3\rangle(a_0)$, $H\langle 0, 1, 3\rangle(c_0)$, $H\langle 0, 1, 3\rangle(e_0)$ et $H\langle 0, 1, 3\rangle(g_0)$ sont fusionnées dans G. En effet, les brins a_0, c_0, e_0 et g_0 appartiennent à la même face dans G.

2.5 Non-changement

Une orbite est considérée comme inchangée lorsqu'elle reste la même après la transformation d'un objet.

Définition 28 (Non-changement d'orbite)
Soient G et H deux G-cartes de dimension $n \in \mathbb{N}$, $\langle\omega\rangle$ un type d'orbite de dimension n et v un brin de G.
L'orbite $G\langle\omega\rangle(v)$ est inchangée *si et seulement si elle identique à $H\langle\omega\rangle(v)$, i.e. $G\langle\omega\rangle(v) = H\langle\omega\rangle(v)$.*

Dans la figure 3.1, l'arête incidente au brin a_0 est inchangée, puisqu'elle est identique dans G et H.

2.6 Modification

Présentée simplement, la modification est un évènement qui exclut les autres. Ainsi, une orbite qui n'est ni créée, ni scindée, ni fusionnée, ni inchangée est donc simplement modifiée.

Définition 29 (Modification d'orbite)
Soient G et H deux G-cartes de dimension $n \in \mathbb{N}$, $\langle\omega\rangle$ un type d'orbite de dimension n et v un brin de G.
L'orbite $G\langle\omega\rangle(v)$ est modifiée *si et seulement si il existe un brin v' de $G\langle\omega\rangle(v)$ et H tel que tout brin préservé (de $G \cap H$) v'' appartient à $G\langle\omega\rangle(v')$ si et seulement si v'' appartient à $H\langle\omega\rangle(v')$*

Dans la figure 3.2, le sommet $G\langle 1,2,3\rangle(a_0)$ est modifié dans H, sans fusion ni scission, puisque $G\langle 1,2,3\rangle(a_0) \neq H\langle 1,2,3\rangle(a_0)$.

3 Les évènements dans les règles de transformation de graphe

Dans la section précédente, nous avons caractérisé les évènements entre deux objets suite à une transformation de G vers H. Dans le cas d'une transformation directe, nous pouvons caractériser les évènements à partir des seules règles de transformation de graphe et plus précisément grâce à leur analyse syntaxique. Une telle analyse nous permet alors de connaître l'ensemble des évènements produits par l'application d'une règle, indépendamment des objets sur lesquels elle est appliquée. De cette manière, nous pouvons réaliser le suivi des évènements pour toute règle et tout objet.

Posons les paramètres suivants. Soient G une G-carte, $r : L \longrightarrow R$ une règle, $m : L \to G$ un morphisme de filtrage et H la transformation directe $G \xrightarrow{(r,m)} H$. Étudions l'évolution d'une orbite d'intérêt $G\langle\omega\rangle(v)$ dans H en fonction de la règle r. Toutes les définitions des évènements entre deux G-cartes G et H s'étendent naturellement entre deux motifs L et R.

3.1 Création

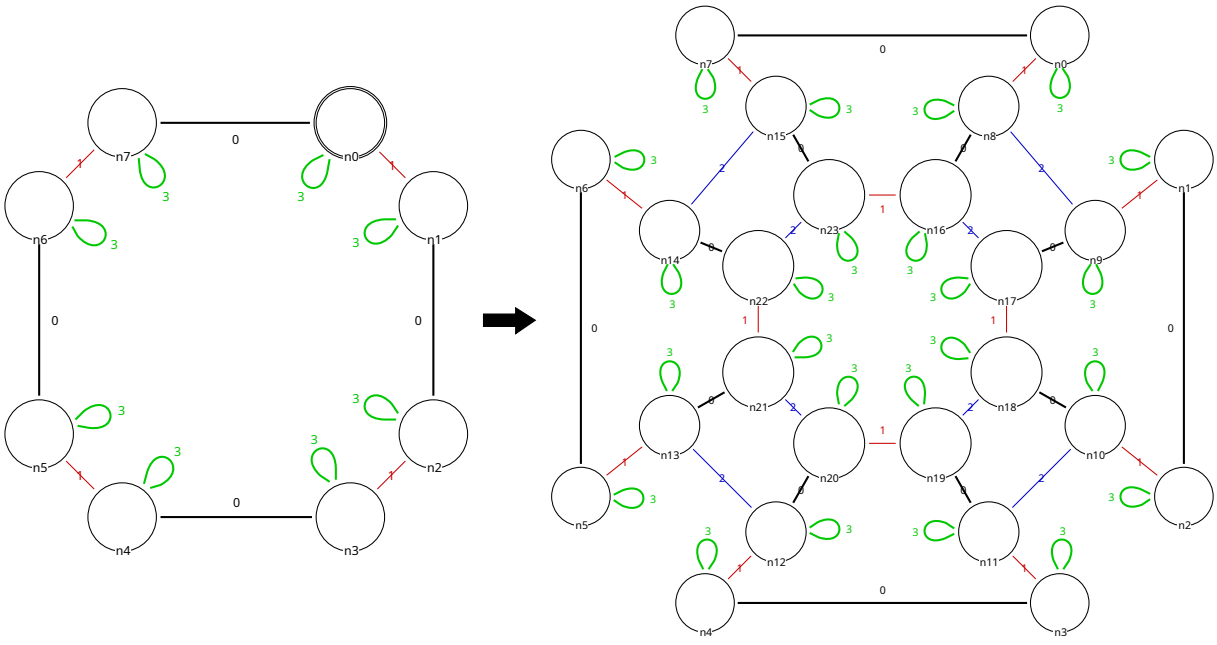


FIGURE 3.3 – Règle de triangulation d'une face carrée

Comme nous l'avons vu plus tôt, la triangulation d'une face crée un sommet en son centre. La règle illustrée dans la figure 3.3 triangule une face carrée. Celle-ci crée le sommet $R\langle 1,2,3\rangle(n16)$. Nous allons montrer que, pour toute face carrée filtrée par cette règle, ce sommet est également créé pour tout objet dans la transformation directe.

Proposition 1 (Création d'orbite)
Soient une règle $r : L \longrightarrow R$, un morphisme de filtrage $m : L \to G$, la transformation directe $G \xrightarrow{(r,m)} H$, un type d'orbite $\langle\omega\rangle$ et v un nœud de R.
Une orbite $R\langle\omega\rangle(v)$ est créée dans la règle r, c'est-à-dire entre L et R, si et seulement si $H\langle\omega\rangle(m'(v))$ est créée dans la transformation directe, c'est-à-dire entre G et H.

Preuve : *Supposons que l'orbite $R\langle\omega\rangle(v)$ est créée dans la règle $r : L \longrightarrow R$. D'après la condition d'arcs incidents sur la règle r, tout nœud créé de R et donc tout nœud de l'orbite $R\langle\omega\rangle(v)$ est la source d'un i-arc, pour tout i de $\langle\omega\rangle$. Notons que les arcs incidents aux nœuds créés sont nécessairement eux-même créés. Donc l'orbite $H\langle\omega\rangle(m'(v))$ est entièrement filtrée par la règle, c'est-à-dire $H\langle\omega\rangle(m'(v)) = m'(R\langle\omega\rangle(v))$. Et donc $H\langle\omega\rangle(m'(v))$ est une orbite créée.*
Supposons que l'orbite $H\langle\omega\rangle(m'(v))$ est créée dans H. Alors par définition de la transformation directe, $H\langle\omega\rangle(m'(v))$ est l'image de l'orbite $R\langle\omega\rangle(v)$ par m'. Et donc $R\langle\omega\rangle(v)$ est une orbite créée par la règle r. □

Nous pouvons voir, dans la règle de triangulation, que tous les nœuds de l'orbite $R\langle 1,2,3\rangle($ $n16)$ sont ajoutés et donc sources d'un i-arc pour tout i dans $\langle 1,2,3\rangle$ conformément à la condition d'arcs incidents (théorème 2 page 17). De la même manière, puisque l'orbite est créée, elle est nécessairement complète (voir la définition 10 page 17). Ainsi, lors de l'application et par transformation directe, l'orbite $R\langle 1,2,3\rangle(n16)$ crée l'orbite $H\langle 1,2,3\rangle(b_2)$ (figures 3.1a et 3.1b page 59).

3.2 Scission

Reprenons la règle de triangulation (figure 3.3). Cette règle scinde une face carrée $L\langle 0,1,3\rangle($ $n0)$ en quatre faces triangulaires $R\langle 0,1,3\rangle(n0)$, $R\langle 0,1,3\rangle(n2)$, $R\langle 0,1,3\rangle(n4)$ et $R\langle 0,1,3\rangle(n6)$. Nous allons montrer que toute face carrée filtrée par cette règle est également scindée dans la transformation directe.

Proposition 2 (Scission d'orbite)
Soient une règle $r : L \longrightarrow R$, un morphisme de filtrage $m : L \to G$, la transformation directe $G \xrightarrow{(r,m)} H$, un type d'orbite $\langle\omega\rangle$ et deux nœuds préservés v_1 et v_2 de $K = L \cap R$.

1. *Si l'orbite $R\langle\omega\rangle(v_1)$ est entièrement filtrée, c'est-à-dire que pour tout nœud v de $R\langle\omega\rangle(v_1)$ et toute dimension i de $\langle\omega\rangle$, $R\langle\omega\rangle(v_1)$ possède un arc étiqueté i et de source v, si l'orbite $L\langle\omega\rangle(v_1)$ est scindée en deux orbites $R\langle\omega\rangle(v_1)$ et $R\langle\omega\rangle(v_2)$ dans la règle, alors $G\langle\omega\rangle(m(v_1))$ est scindée en deux orbites $H\langle\omega\rangle(m'(v_1))$ et $H\langle\omega\rangle(m'(v_2))$ dans la transformation directe.*
2. *Si les orbites $R\langle\omega\rangle(v_1)$ et $R\langle\omega\rangle(v_2)$ sont partiellement filtrées,*
   si l'orbite $L\langle\omega\rangle(v_1)$ est scindée en deux orbites $R\langle\omega\rangle(v_1)$ et $R\langle\omega\rangle(v_2)$ dans la règle, alors $G\langle\omega\rangle(m(v_1))$ est soit scindée en deux orbites $H\langle\omega\rangle(m'(v_1))$ et $H\langle\omega\rangle(m'(v_2))$ dans la transformation directe, soit modifiée (et alors $H\langle\omega\rangle\ (m'(v_1)) = H\langle\omega\rangle(m'(v_2))$).
3. *Réciproquement,*

si l'orbite $G\langle\omega\rangle(m(v_1))$ est scindée en deux orbites $H\langle\omega\rangle(m'(v_1))$ et $H\langle\omega\rangle(m'(v_2))$ dans la transformation directe, alors il existe k_1 et k_2, tous deux dans $K = L \cap R$, tels que $m(k_1) \in G\langle\omega\rangle(m(v_1))$, $m(k_2) \in G\langle\omega\rangle(m(v_2))$ et $L\langle\omega\rangle(k_1)$ est scindée en $R\langle\omega\rangle(k_1)$ et $R\langle\omega\rangle(k_2)$.

Preuve :

1. *Supposons que l'orbite $L\langle\omega\rangle(v_1)$ est scindée dans la règle $r : L \longrightarrow R$ avec l'orbite $R\langle\omega\rangle(v_1)$ entièrement filtrée. L'orbite $G\langle\omega\rangle(m(v_1))$ n'est pas nécessairement entièrement filtrée, c'est-à-dire que $m(L\langle\omega\rangle(v_1)) \subset G\langle\omega\rangle(m(v_1))$. De même, $m'(R\langle\omega\rangle(v_2)) \subset H\langle\omega\rangle(m'(v_2))$. Par hypothèse, l'orbite $R\langle\omega\rangle(v_1)$ est entièrement filtrée, donc $H\langle\omega\rangle(m'(v_1)) = m'(R\langle\omega\rangle(v_1))$, et est distincte de $R\langle\omega\rangle(v_2))$, donc $m'(R\langle\omega\rangle(v_1))\cap m'(R\langle\omega\rangle(v_2)) = \emptyset$. On en déduit que $H\langle\omega\rangle(m'(v_1))\cap m'(R\langle\omega\rangle(v_2)) = \emptyset$. Or $m'(v_2)$ n'est pas un nœud de $H\langle\omega\rangle(m'(v_1))$, donc $H\langle\omega\rangle(m'(v_1))$ et $H\langle\omega\rangle(m'(v_2))$ sont des orbites distinctes. Par conséquent, $G\langle\omega\rangle(m(v_1))$ est une orbite scindée.*
2. *Supposons que l'orbite $L\langle\omega\rangle(v_1)$ est scindée dans la règle $r : L \longrightarrow R$ avec $R\langle\omega\rangle(v_1)$ et $R\langle\omega\rangle(v_2)$ toutes deux partiellement filtrées.*

   Si $m'(v_1)$ et $m'(v_2)$ appartiennent à la même $\langle\omega\rangle$-orbite dans H, c'est-à-dire $H\langle\omega\rangle(m'(v_1)) = H\langle\omega\rangle(m'(v_2))$, alors l'orbite $G\langle\omega\rangle(m(v_1))$ est modifiée dans la transformation directe.

   Si $m'(v_1)$ et $m'(v_2)$ appartiennent à deux $\langle\omega\rangle$-orbites dans H, c'est-à-dire $H\langle\omega\rangle(m'(v_1)) \cap H\langle\omega\rangle(m'(v_2)) = \emptyset$, alors l'orbite $G\langle\omega\rangle(m(v_1))$ est scindée dans la transformation directe.
3. *Supposons que l'orbite $G\langle\omega\rangle(m(v_1))$ est scindée dans H en $H\langle\omega\rangle(m'(v_1))$ et $H\langle\omega\rangle(m'(v_2))$, c'est-à-dire que $G\langle\omega\rangle(m(v_1)) = G\langle\omega\rangle(m(v_2))$ et $H\langle\omega\rangle(m'(v_1)) \neq H\langle\omega\rangle(m'(v_2))$.*

   Raisonnons par l'absurde. Supposons que pour tout couple de nœuds préservés k_1 et k_2 de $K = L \cap R$ tels que $m(k_1) \in G\langle\omega\rangle(m(v_1))$ et $m(k_2) \in G\langle\omega\rangle(m(v_1))$, $L\langle\omega\rangle(k_1)$ n'est pas scindée en $R\langle\omega\rangle(k_1)$ et $R\langle\omega\rangle(k_2)$.

   Alors pour tout chemin c dans G de $m(v_1)$ à $m(v_2)$ étiqueté dans $\langle\omega\rangle^*$ (i.e., c est un chemin de $G\langle\omega\rangle(m(v_1))$), c n'existe pas dans H. Notons $c = c_0c_1c_2\dots c_p$ tel que tout sous-chemin c_i ait une source et une cible filtrés par K et aucun autre nœud filtré par K. Tout sous-chemin c_i est donc soit entièrement filtré par la règle, soit pas du tout. Si c_i n'est pas filtré par la règle, alors il appartient à H et donc sa source et sa cible sont dans la même $\langle\omega\rangle$-orbite de H. Si c_i est (entièrement) filtré par la règle, alors la source et la cible sont nécessairement filtrés par deux nœuds de la même $\langle\omega\rangle$-orbite de L. Et par hypothèse, la source et la cible sont filtrés par deux nœuds de la même $\langle\omega\rangle$-orbite de R. Et donc, la source et la cibles sont dans la même $\langle\omega\rangle$-orbite dans H. En conséquence, par transitivité, v_1 et v_2 sont dans la même $\langle\omega\rangle$-orbite de H ce qui est contraire à l'hypothèse.

   En conclusion, il existe k_1 et k_2 dans K tels que $m(k_1) \in G\langle\omega\rangle(m(v_1))$, $m(k_2) \in G\langle\omega\rangle(m(v_1))$ et $L\langle\omega\rangle(k_1)$ est scindée en $R\langle\omega\rangle(k_1)$ et $R\langle\omega\rangle(k_2)$.

Nous pouvons voir dans la règle que la face $L\langle0,1,3\rangle(n0)$ est scindée en quatre faces distinctes dont $R\langle0,1,3\rangle(n0)$. Réciproquement, la face $R\langle0,1,3\rangle(n0)$ est issue de la scission de la face $L\langle0,1,3\rangle(n0)$ localisée par l'arête $L\langle0,2,3\rangle(n0)$. Cette dernière est l'*origine* de $R\langle0,1,3\rangle($

$n0$), c'est-à-dire que $R\langle 0,1,3\rangle(n0)$ est l'unique face issue de la scission de $L\langle 0,1,3\rangle(n0)$ dont l'intersection avec l'arête $L\langle 0,2,3\rangle(n0)$ (inchangée par la règle) est non vide. Remarquons que dans la règle, la face $L\langle 0,1,3\rangle(n0)$ est complète. Cela signifie que dans la transformation directe, la face est entièrement filtrée et est donc bien scindée en quatre faces distinctes. De même, chacune de ces faces est incidente à une arête préservée, que nous appelons leur origine

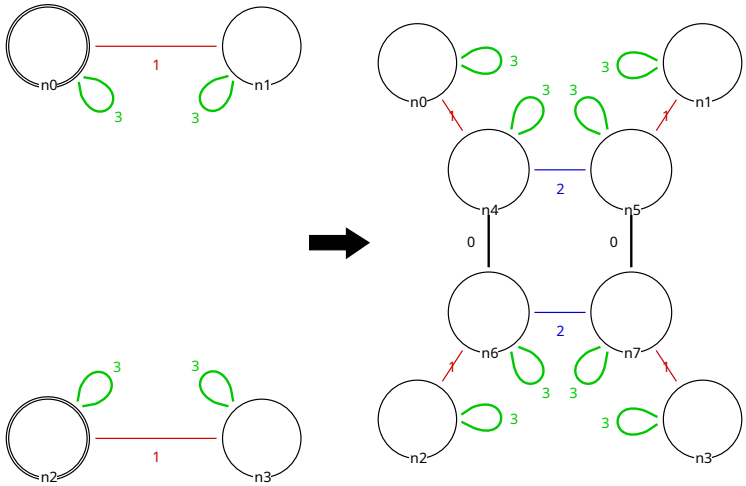
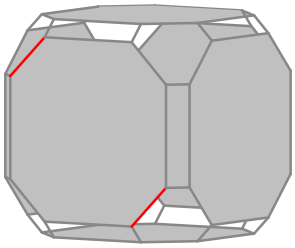
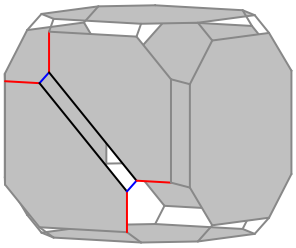


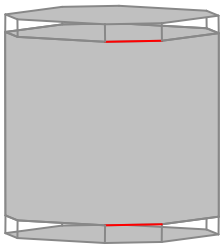
Figure 3.4 – Règle d'insertion d'arête



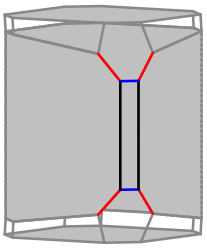
(a) Face d'un cube



(b) Face scindée d'un cube



(c) Cylindre



(d) Cylindre avec une arête

Figure 3.5 – Exemples d'application de la règle d'insertion d'arête (figure 3.4)

Si aucune des orbites concernées n'est entièrement filtrée par la règle, alors la scission peut exister ou non selon l'objet. La figure 3.4 représente une règle d'insertion d'arête. Celle-ci ne filtre que partiellement la face à scinder, il est donc possible, mais pas certain, que l'application de cette règle provoque la scission d'une face. La figure 3.5 illustre deux cas d'application de l'insertion d'arête. Dans les figures 3.5a et 3.5b, la déconnexion locale provoque une scission de la face bien que celle-ci ne soit pas entièrement filtrée par la règle. Dans les figures 3.5c et

3.5d, la déconnexion locale ne provoque pas de scission de la face. En effet, les deux parties déconnectées dans la règle restent connectées dans la partie non filtrée de l'objet.

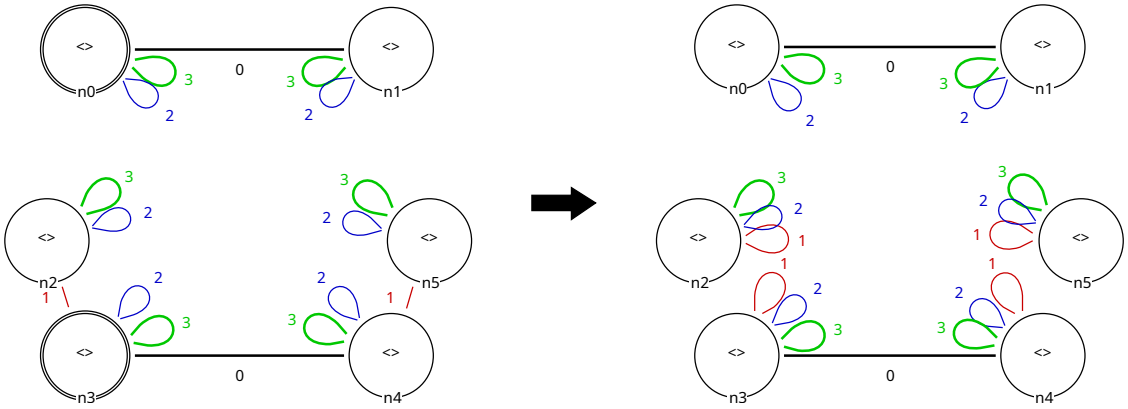


FIGURE 3.6 – Règle de décousure des sommets d'une arête

(a) G (b) H

FIGURE 3.7 – Décousure d'une arête de face par ses sommets

Dans certains cas, lorsque plusieurs composantes sont filtrées par une règle, une scission peut être détectée au sein d'une seule composante dans la règle, tout en scindant les deux composantes filtrées dans l'objet. Par exemple, la règle (figure 3.6) filtre deux arêtes dont l'une avec ses sommets . Seule la seconde arête est transformée car la règle la découd de ses sommets incidents. Pourtant, si cette règle est appliquée et que ses deux composantes dans L sont appliquées sur deux arêtes d'une seule et même face dans G, la transformation crée bien une scission entre les deux composantes filtrées. Par exemple, dans les figures 3.7a et 3.7b, la composante incidente à $n0$ filtre l'arête incidente au brin v et la seconde composante filtre l'arête et les sommets atteignables à partir du brin k_2 et qui comprend le brin k_1. Suite à l'application de la règle, nous pouvons observer qu'il n'existe plus de sous-chemin dans $\langle 0, 1\rangle$ permettant de relier v à k_2 alors que dans la règle, les seules scissions détectées sont celles qui déconnectent une arête de ses sommets.

Un autre exemple peut être trouvé avec la règle (figure 3.8) qui découd les 2-arcs de deux arêtes, mais qui n'exprime aucune scission entre les deux composantes. Tout comme dans l'exemple précédent, nous pouvons observer dans les figures 3.9a et 3.9b que dans la transformation directe, il n'existe plus de sous-chemin dans $\langle 0, 1, 2\rangle$ permettant de relier v_1 à v_2. Notons toutefois, qu'une seule des deux scissions suffirait pour déconnecter la $\langle 0, 1, 2\rangle$-orbite incidente à v_1 de celle incidente à v_2.

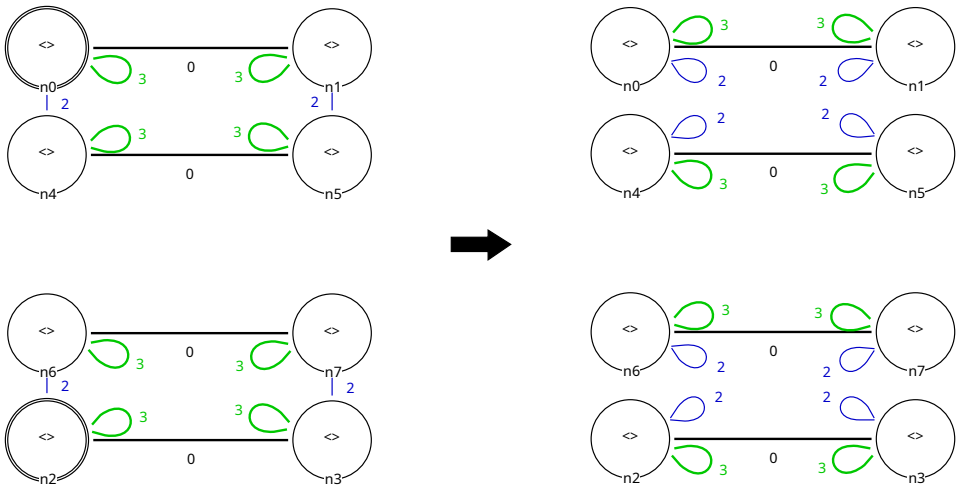


Figure 3.8 – Règle de double décousure

(a) G (b) H

Figure 3.9 – Double décousure de ruban

3.3 Suppression

La règle, dans la figure 3.10, est l'inverse de la triangulation d'une face carrée, c'est-à-dire celle qui supprime un sommet dans une face. Dans ce cas, nous pouvons voir que l'orbite $L\langle 1,2,3\rangle(n16)$ est supprimée dans R. Nous allons montrer que pour tout sommet au centre d'une face carrée, celui-ci est supprimé dans la transformation directe.

Proposition 3 (Suppression d'orbite)
Soient une règle $r : L \longrightarrow R$, un morphisme de filtrage $m : L \to G$, la transformation directe $G \xrightarrow{(r,m)} H$, un type d'orbite $\langle\omega\rangle$ et v un nœud de L.
Une orbite $L\langle\omega\rangle(v)$ est supprimée *dans la règle r, c'est-à-dire entre L et R, si est seulement si $G\langle\omega\rangle(m(v))$ est* supprimée *dans la transformation directe, c'est-à-dire entre G et H.*

Preuve : *Cette preuve est symétrique à celle de la création (voir la proposition 1).* □

Comme pour la création, les orbites supprimées sont toujours entièrement filtrées. Ainsi, quel que soit l'objet transformé, ses suppressions sont toujours détectées dans la règle et appliquées

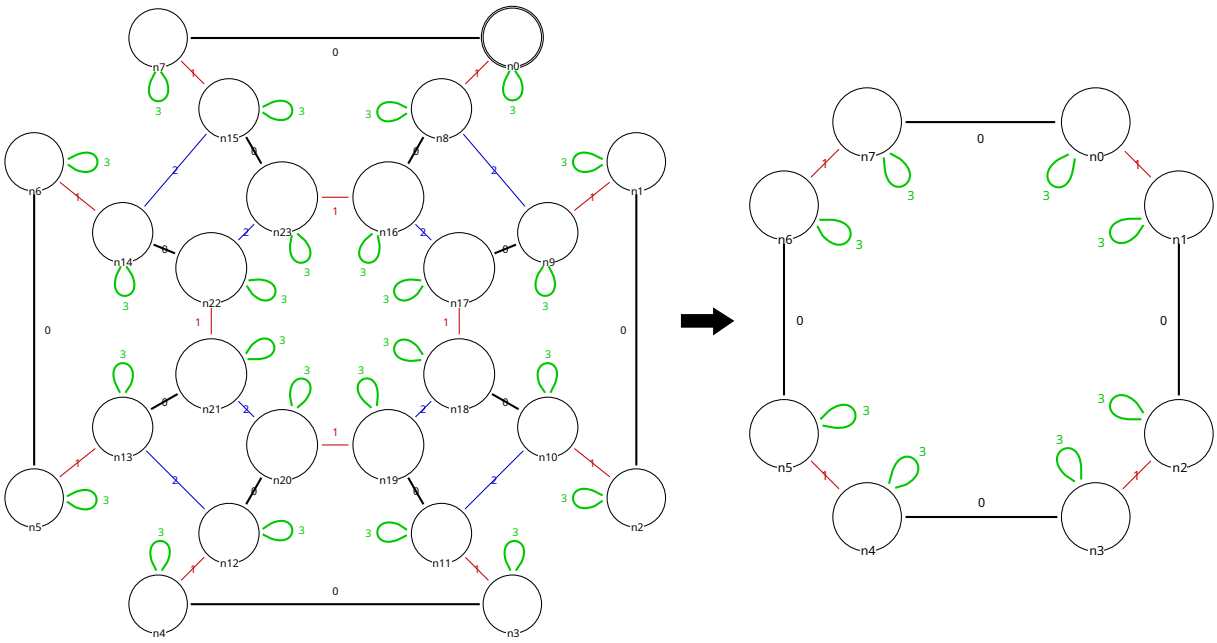


FIGURE 3.10 – Règle de contraction d'un sommet dans une face carrée

dans les objets au cours de la transformation directe.

3.4 Fusion

Reprenons la règle de suppression de sommet (figure 3.10). Du point de vue de la fusion, les faces incidentes aux nœuds $n0$, $n2$, $n4$ et $n6$ dans L sont fusionnées en une unique face $R\langle 0,1,3\rangle(n0)$. Nous allons montrer que cette règle provoque la fusion des faces autour d'un sommet dans la transformation directe.

Proposition 4 (Fusion d'orbites)
Soient une règle $r : L \longrightarrow R$, un morphisme de filtrage $m : L \to G$, la transformation directe $G \xrightarrow{(r,m)} H$, un type d'orbite $\langle\omega\rangle$ et deux nœuds préservés v_1 et v_2 de $K = L \cap R$.

1. *Si l'orbite $L\langle\omega\rangle(v_1)$ est entièrement filtrée, c'est-à-dire que pour tout nœud v de $L\langle\omega\rangle(v_1)$ et toute dimension i de $\langle\omega\rangle$, $L\langle\omega\rangle(v_1)$ possède un arc étiqueté i et de source v.*
   Si les orbites $L\langle\omega\rangle(v_1)$ et $L\langle\omega\rangle(v_2)$ sont fusionnées en une orbite $R\langle\omega\rangle(v_1)$ dans la règle, i.e. $R\langle\omega\rangle(v_1) = R\langle\omega\rangle(v_2)$, alors $G\langle\omega\rangle(m(v_1))$ et $G\langle\omega\rangle(m(v_2))$ sont fusionnées en une orbite $H\langle\omega\rangle(m'(v_1))$ dans la transformation directe.
2. *Si les orbites $L\langle\omega\rangle(v_1)$ et $L\langle\omega\rangle(v_2)$ sont partiellement filtrées.*
   Si les orbites $L\langle\omega\rangle(v_1)$ et $L\langle\omega\rangle(v_2)$ sont fusionnées en une orbite $R\langle\omega\rangle(v_1)$ dans la règle, alors $G\langle\omega\rangle(m(v_1))$ et $G\langle\omega\rangle(m(v_2))$ sont soit fusionnées en une orbite $H\langle\omega\rangle(m'(v_1))$ dans la transformation directe, soit modifiées et alors $G\langle\omega\rangle(m(v_1)) = G\langle\omega\rangle(m(v_2))$.
3. *Réciproquement,*
   Si les orbites $G\langle\omega\rangle(m(v_1))$ et $G\langle\omega\rangle(m(v_2))$ sont fusionnées en une orbite $H\langle\omega\rangle(m'(v_1))$ dans la transformation directe, alors il existe k_1 et k_2, tous deux dans

$K = L \cap R$, tels que $m(k_1) \in H\langle\omega\rangle(m(v_1))$, $m(k_2) \in H\langle\omega\rangle(m(v_2))$ et $L\langle\omega\rangle(k_1)$ et $L\langle\omega\rangle(k_2)$ sont fusionnées en $R\langle\omega\rangle(k_1)$.
les orbites $L\langle\omega\rangle(v_1)$ et $L\langle\omega\rangle(v_2)$ sont fusionnées en une orbite $R\langle\omega\rangle(v_1)$ dans la règle.

Preuve : *La preuve est symétrique à celle de la scission (voir la proposition 2).* □

Nous pouvons voir dans la règle que la face $R\langle 0,1,3\rangle(n0)$ est issue de la fusion des faces incidentes aux nœuds $n0$, $n2$, $n4$ et $n6$. Le sommet $L\langle 1,2,3\rangle(n16)$ est l'*origine* de la face fusionnée $R\langle 0,1,3\rangle(n0)$, car sa suppression (par dissolution) entraîne la fusion de ses faces incidentes. Puisque les sommets et les faces sont complets, cela signifie que toutes les faces incidentes à ce sommet sont distinctes les unes des autres. Ainsi, au cours de la transformation directe, cette règle fusionne les faces incidentes à un sommet supprimé.

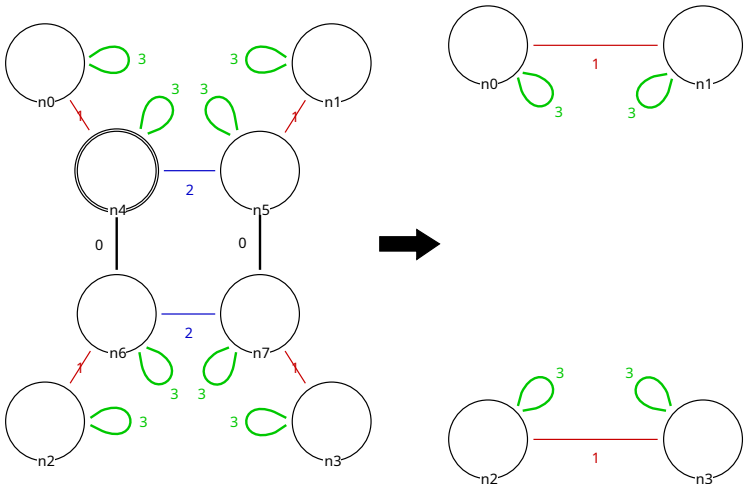
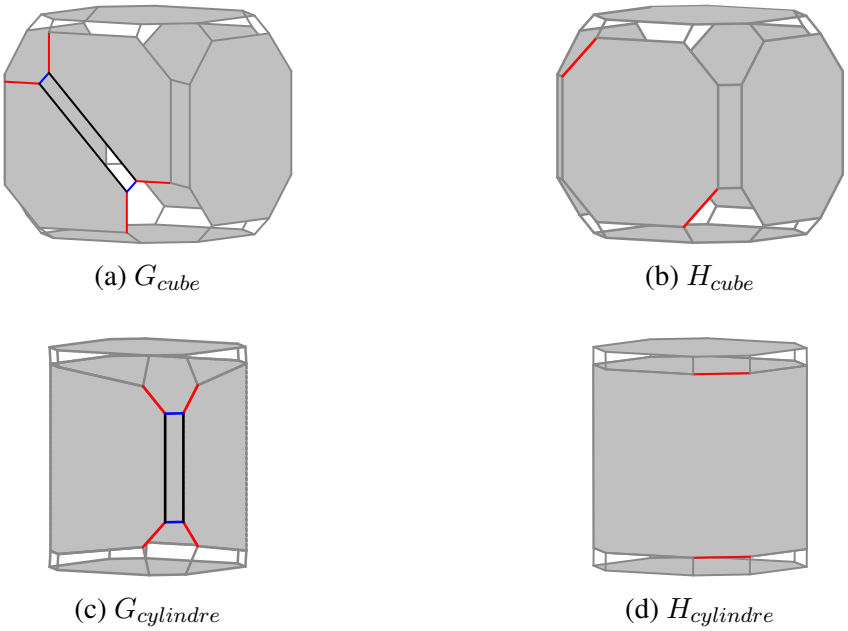


Figure 3.11 – Règle de suppression d'une arête



(a) G_{cube} (b) H_{cube}

(c) $G_{cylindre}$ (d) $H_{cylindre}$

Figure 3.12 – Suppressions d'arêtes dans des faces

Comme pour la scission, lorsqu'au moins une des orbites fusionnées est entièrement filtrée, comme dans l'exemple de la contraction de sommet (figure 3.10), les fusions sont précisément

détectées dans les règles. Dans le cas contraire, selon l'objet transformé, les orbites fusionnées dans la règle peuvent filtrer des orbites différentes ou une seule et même orbite dans l'objet. C'est le cas par exemple de la règle de suppression d'arête, figure 3.11 qui peut fusionner deux faces de part et d'autre de l'arête s'il s'agit de deux faces distinctes.

En effet, selon l'objet considéré, la fusion pourra ou non avoir lieu dans l'objet. Par exemple, la figure 3.12 montre la suppression d'une arête entre deux faces distinctes qui sont donc fusionnées par application de la règle figures 3.12a et 3.12b.

Au contraire, dans les figures 3.12c et 3.12d, l'application de la même règle (figure 3.11) supprime une arête incidente à deux cotés opposés de la même face. La face est donc simplement modifiée.

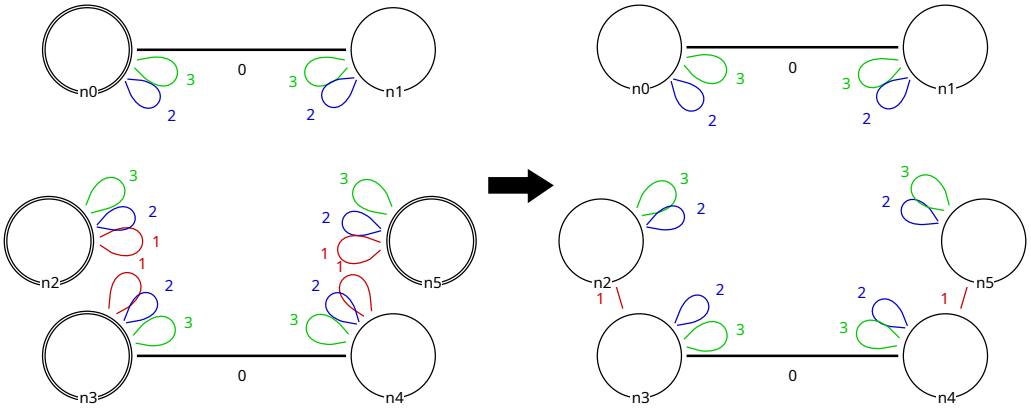


FIGURE 3.13 – Couture de sommets aux extrémités d'une arête

(a) G (b) H

FIGURE 3.14 – Couture de sommets aux extrémités d'une arête

Si nous inversons la règle figure 3.6 (page 65), nous obtenons la règle figure 3.13. Dans ce cas, contrairement à la règle de scission, il apparaît évident qu'il suffit de reconnecter un seul des deux sommets pour reconnecter l'arête au reste de la face (figures 3.14a et 3.14b).

Pour la règle figure 3.15, en revanche, l'ensemble des fusions sont nécessaires pour obtenir le ruban à partir des trois faces (figures 3.16b et 3.16a).

Remarquons que dans certains cas, il peut y avoir simultanément une fusion et une scission. Par exemple, la figure 3.17 représente une règle de modification de maillage triangulaire appelée « edge flip ». Cette dernière provoque simultanément la scission (la face incidente à $n2$ est scindée en deux faces respectivement incidentes à $n2$ et $n3$) et la fusion de deux faces (les faces incidentes respectivement à $n0$ et $n1$ sont fusionnées).

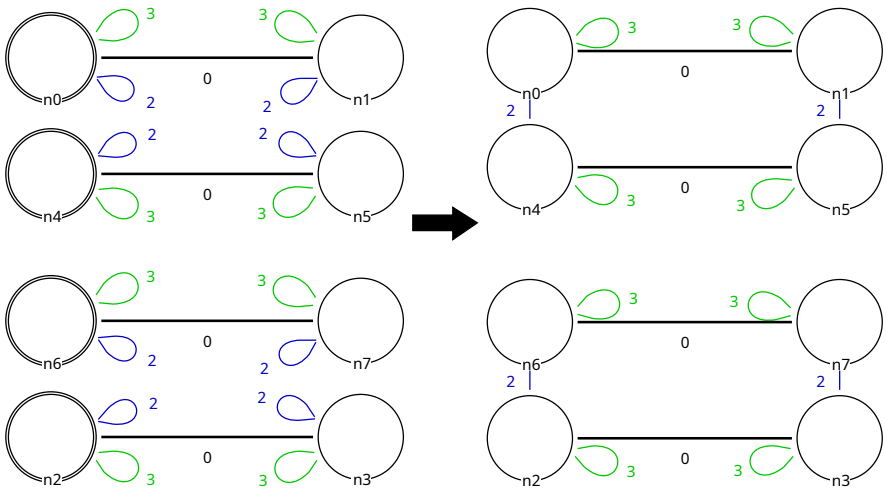
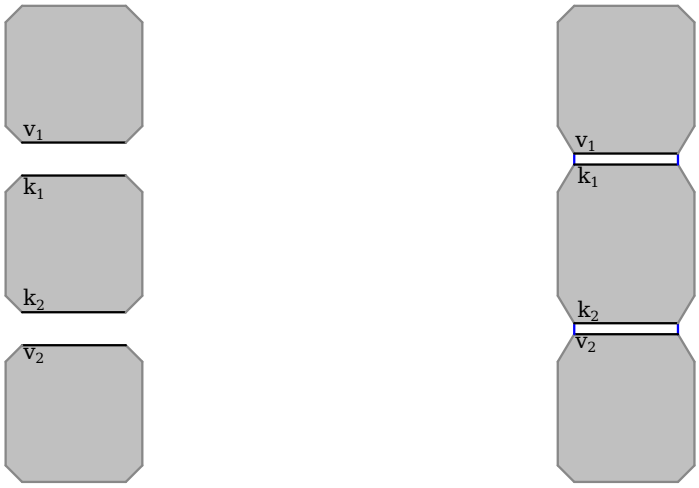


Figure 3.15 – Double couture d'arêtes



(a) G (b) H

Figure 3.16 – Double couture d'arêtes par leurs 2-arcs

La figure 3.18 illustre l'application de cette règle sur un maillage. Les deux faces triangulaires sont à la fois fusionnées et scindées en deux nouvelles faces triangulaires.

Notons que le même phénomène se produit sur la règle de suppression d'arête de la figure 3.11 où nous observons une scission entre les faces incidentes à $n0$ et $n2$ et une fusion entre les faces incidentes à $n0$ et $n1$. Selon l'objet transformé, l'un ou l'autre de ces évènements se produit. Comme vu précédemment (figures 3.12a et 3.12b), la fusion se produit mais il n'y a pas de scission. Au contraire (figures 3.12c et 3.12d) la fusion ne se produit pas, mais la suppression de la seule arête qui relie le haut et le bas de la face, déconnecte la G-carte en deux composantes connexes, et donc la face en deux nouvelles faces carrées.

3.5 Non-changement

Reprenons la règle de triangulation (figure 3.3 page 61). Nous pouvons voir que l'arête $L\langle 0,2,3\rangle(n0)$ est inchangée entre L et R (et il en va de même pour les arêtes incidentes aux

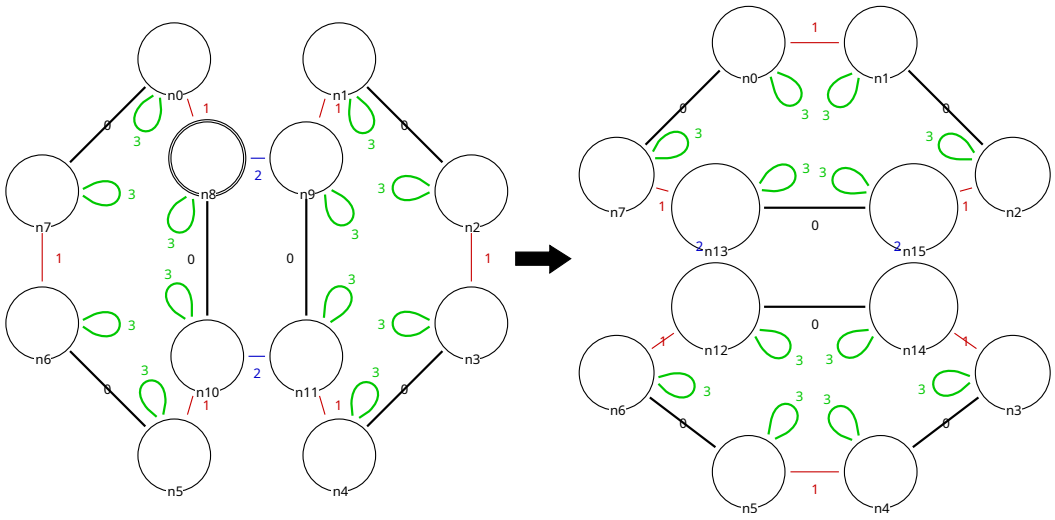
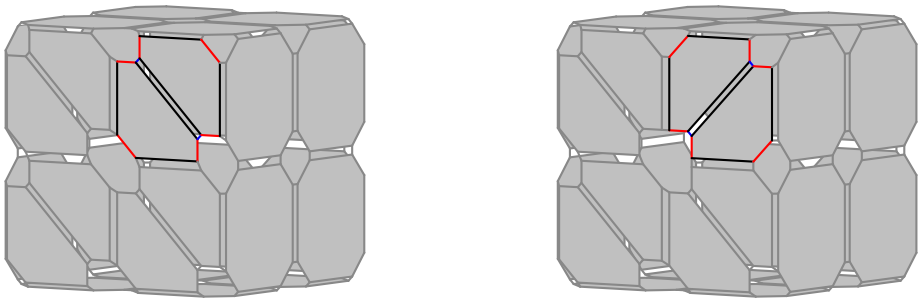


FIGURE 3.17 – Règle de rotation d'arête « edge flip »



(a) G (b) H

FIGURE 3.18 – Application de la règle de « edge flip » (figure 3.17)

nœuds $n2$, $n4$ et $n6$). Nous allons montrer que cette arête est également inchangée pour toute transformation directe.

Proposition 5 (Non-changement d'orbite)
Soient une règle $r : L \longrightarrow R$, un morphisme de filtrage $m : L \to G$, la transformation directe $G \xrightarrow{(r,m)} H$, un type d'orbite $\langle\omega\rangle$, v un nœud préservé de $K = L \cap R$.
Une orbite $L\langle\omega\rangle(v)$ est inchangée *dans la règle r entre L et R, c'est-à-dire $L\langle\omega\rangle(v) = R\langle\omega\rangle(v)$, si et seulement si $G\langle\omega\rangle(m(v))$ est* inchangée *dans la transformation directe entre G et H.*

Preuve : *Supposons que l'orbite $L\langle\omega\rangle(v)$ est inchangée dans la règle $r : L \longrightarrow R$. L'orbite $G\langle\omega\rangle(m(v))$ n'est pas nécessairement entièrement filtrée, c'est-à-dire $m(L\langle\omega\rangle(v)) \subset G\langle\omega\rangle(m(v))$ et $m'(R\langle\omega\rangle(v)) \subset H\langle\omega\rangle(m'(v))$. Par définition de la transformation directe, la partie filtrée de l'orbite est inchangée : $m(L\langle\omega\rangle(v)) = m'(R\langle\omega\rangle(v))$. Par ailleurs, la partie non filtrée est aussi inchangée, i.e. $G\langle\omega\rangle(m(v)) \setminus m(L\langle\omega\rangle(v)) = H\langle\omega\rangle(m'(v)) \setminus m'(R\langle\omega\rangle(v))$. Donc $G\langle\omega\rangle(m(v))$ est une orbite inchangée.*
Supposons que l'orbite $G\langle\omega\rangle(m(v))$ est inchangée entre G et H, c'est-à-dire que $G\langle\omega\rangle(m(v)) = H\langle\omega\rangle(m'(v))$. Alors, par définition de la transformation directe, $L\langle\omega\rangle(v)$ est une orbite inchangée par la règle r. □

Remarque : *En plus des orbites (partiellement) filtrées, les orbites de G non filtrées (telles que*

$G\langle\omega\rangle(v) \cap m(L) = \emptyset$) sont par définition inchangées. Nous dirons que la règle est sans effet *sur l'orbite.*

(a) G (b) H

Figure 3.19 – Non-changement des arêtes de la face triangulée

Par exemple, dans la figure 3.19, la triangulation laisse l'arête incidente au brin b_0 inchangée. Par ailleurs l'application de la règle de triangulation n'a aucun effet sur la partie grisée dans G (figure 3.19a) car celle-ci n'est pas filtrée par la règle. Le résultat est que la partie grisée de G est identique à la partie grisée de H (figure 3.19b).

3.6 Modification

La règle de triangulation (figure 3.3) produit un certain nombre de transformations lors de son application sur un objet. Parmi ces transformations se trouve la modification des sommets. Nous pouvons voir que le sommet $L\langle 1,2,3\rangle(n0)$ ainsi que ceux incidents aux nœuds $n2$, $n4$ et $n6$ sont modifiés. Nous allons montrer que l'application de cette règle modifie bien les sommets filtrés dans G au cours de la transformation directe.

Proposition 6 (Modification d'orbite)
Soient une règle $r : L \longrightarrow R$, un morphisme de filtrage $m : L \to G$, la transformation directe $G \xrightarrow{(r,m)} H$, un type d'orbite $\langle\omega\rangle$ et v un nœud préservé de $K = L \cap R$.
Si une orbite $L\langle\omega\rangle(v)$ est modifiée dans la règle r, alors $G\langle\omega\rangle(m(v))$ est modifiée dans la transformation directe.
Réciproquement, si une orbite $G\langle\omega\rangle(m(v))$ est modifiée dans la transformation directe, alors $L\langle\omega\rangle(v)$ est modifiée, fusionnée ou scindée dans la règle.

Preuve : *Soit v un nœud préservé de $K = L \cap R$ tel que l'orbite $L\langle\omega\rangle(v)$ est modifiée dans la règle r. Donc par définition :*

- *$L\langle\omega\rangle(v) \neq R\langle\omega\rangle(v)$; et donc, par transformation directe, $G\langle\omega\rangle(m(v)) \neq H\langle\omega\rangle(m'(v))$;*
- *pour tout nœud préservé v' de $K = L \cap R$, v' est un nœud de $L\langle\omega\rangle(v)$ si et seulement si c'est un nœud de $R\langle\omega\rangle(v)$; donc $m(v')$ est un nœud de $G\langle\omega\rangle(m(v))$ si et seulement si c'est un nœud de $H\langle\omega\rangle(m'(v))$. Par ailleurs, les nœuds non filtrés de $G\langle\omega\rangle(m(v))$ sont nécessairement des nœuds non filtrés dans $H\langle\omega\rangle(m'(v))$. Par conséquent, pour tout nœud v'' conservé de $G \cap H$, v'' est un nœud de $G\langle\omega\rangle(m(v))$ si et seulement si c'est un nœud de $H\langle\omega\rangle(m'(v))$;*

Réciproquement, d'après les propositions précédentes, il est impossible que $L\langle\omega\rangle(v)$ soit inchangée ou supprimée dans la règle r, de même $R\langle\omega\rangle(v)$ ne peut pas être créée dans r. En revanche, $L\langle\omega\rangle(v)$ peut être scindée, fusionnée ou modifiée dans la règle r.

(a) G (b) H

FIGURE 3.20 – Modification de sommets par triangulation

Nous pouvons voir dans la règle de triangulation (figure 3.3 page 61), que les $\langle 1, 2\rangle$-orbites de L incidentes aux nœuds $n0$, $n2$, $n4$ et $n6$ sont modifiées par la triangulation dans R suite à l'ajout de 2-arcs dans les orbites. Notons que ces orbites ne sont ni créées, car une partie des nœuds ne sont pas créés, ni scindées, car il s'agit d'une $\langle 1, 2\rangle$-orbite et pas d'une $\langle 1\rangle$-orbite. Dans la transformation directe (figures 3.20a et 3.20b), les sommets sont bien simplement modifiés.

4 Les évènements dans les règles Jerboa

Dans la section précédente, nous avons formalisé la détection des évènements à l'aide de l'analyse syntaxique des règles de transformation de graphe. Dans cette section, nous proposons d'étendre cette analyse aux règles Jerboa ainsi que de formaliser la notion d'origine d'une orbite. Grâce à cette notion d'origine, il devient possible d'enrichir le suivi des orbites en distinguant chaque orbite créée, scindée ou fusionnée, à l'aide d'une entité qui joue alors le rôle d'ancêtre.

4.1 Création

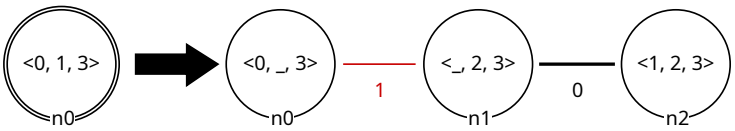


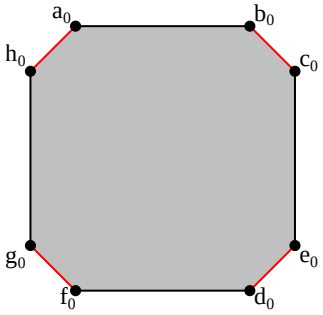
FIGURE 3.21 – Règle de triangulation de face

Continuons avec la règle de triangulation, cette fois sous la forme d'une règle Jerboa (figure 3.21). Dans la section 3.1 (page 61), nous avons étudié la création d'un sommet dans une face carrée suite à sa transformation directe par la règle instanciée de triangulation (figure 3.3 page 73). Nous allons maintenant montrer que la création du sommet $R\langle 1, 2, 3\rangle(n2)$ dans la règle Jerboa détermine, elle aussi, la création d'un sommet dans la transformation directe.

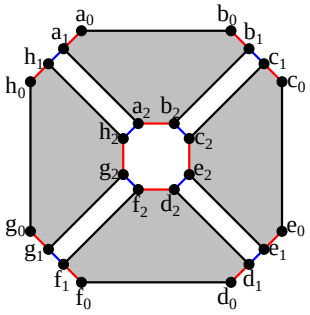
Proposition 7 (Création d'orbite)
Soient une règle Jerboa $r : L \longrightarrow R$, un nœud d'accroche h de L, un morphisme de filtrage $\underline{m} : \underline{L} \rightarrow G$ sur une G-carte G, un type d'orbite $\langle\omega\rangle$ et v un nœud de R.
L'orbite $\underline{R}\langle\omega\rangle(v)$ est créée dans la règle si et seulement si toutes les orbites $H\langle\omega\rangle(m((v, v')))$ sont créées dans la transformation directe $G \xrightarrow{r,\underline{m}} H$, pour tous les brins v' de l'orbite d'instanciation $O = G\langle l(h)\rangle(\underline{m}(h))$.

Preuve : *Soit v un nœud de R tel que l'orbite $\underline{R}\langle\omega\rangle(v)$ est créée par la règle, c'est-à-dire telle que $\underline{R}\langle\omega\rangle(v) \cap \underline{L} = \emptyset$. Soit $O = G\langle l(h)\rangle(\underline{m}(h))$, l'orbite d'instanciation. Soit w un nœud quelconque de l'orbite $\underline{R}\langle\omega\rangle(v)$. w est un nœud créé de la règle (c'est-à-dire de $R \setminus L$). Donc, par construction de la règle instanciée, tous les nœuds (w, v') pour tous les brins v' de l'orbite d'instanciation O, sont des nœuds créés dans la règle instanciée $r(\underline{m})$.*
Donc toutes les orbites $R(O)\langle\omega\rangle((v, v'))$ pour tous les brins v' de l'orbite d'instanciation O sont des orbites créées dans la règle instanciée $r(\underline{m})$. Et donc, d'après la proposition 1, toutes les orbites $H\langle\omega\rangle(m((v, v')))$ sont créées dans la transformation directe $G \xrightarrow{r,m} H$, pour tous les brins v' de l'orbite d'instanciation O.

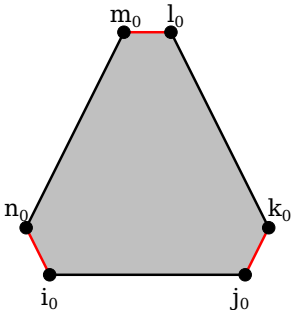
Réciproquement, supposons que toutes les orbites $H\langle\omega\rangle(m'((v, v')))$ sont créées dans la transformation directe $G \xrightarrow{r,m} H$, pour tous les brins v' de l'orbite d'instanciation O. Donc, d'après la proposition 1, toutes les orbites $R(O)\langle\omega\rangle((v, v'))$ pour tous les brins v' de l'orbite d'instanciation O sont des orbites créées dans la règle instanciée $r(\underline{m})$. Donc par construction de l'instanciation, $\underline{R}\langle\omega\rangle(v)$ est une orbite créée dans la règle r. □



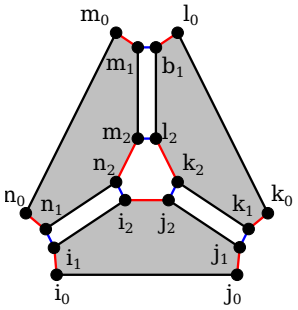
(a) Instanciation d'une face de G_{quad}



(b) Instanciation d'une face de H_{quad}



(c) Instanciation d'une face de G_{tri}



(d) Instanciation d'une face de H_{tri}

Figure 3.22 – Instanciations de la règle de triangulation pour différents polygones

Dans la règle de triangulation (figure 3.21), l'orbite $R\langle 1, 2, 3\rangle(n2)$ contient uniquement le nœud $n2$. Puisque celui-ci n'appartient pas à L, nous pouvons en déduire qu'il est créé et donc l'orbite est créée dans R. Dans la transformation directe, pour une face carrée, la règle est instanciée sous la forme de la règle 3.3. Nous avons montré dans la section précédente que l'application de cette règle instanciée sur une face carrée de G crée bien un sommet dans H. Toutefois, dans ce cas, cette règle peut être appliquée sur tout type de face peu importe le nombre de côtés comme une face carrée (figures 3.22a et 3.22b) ou triangulaire (figures 3.22c et 3.22d). Pour chacune des deux instanciations, le sommet est bien créé $M(G_{quad}\langle 1, 2, 3\rangle(a_2))$ et $M(G_{tri}\langle 1, 2, 3\rangle(m_2))$ ainsi que dans la transformation directe avec, par exemple, la face triangulaire (figures 3.23a et

(a) G_{tri} (b) H_{tri}

FIGURE 3.23 – Triangulation d'une face triangulaire

3.23b). Enfin, puisque le sommet est créé dans la règle Jerboa et ses instanciations, celui-ci est créé dans H_{tri} (figures 3.23a et 3.23b)

L'analyse des règles Jerboa nous permet de définir ce que nous appelons les orbites d'origines. Pour le présenter simplement, une orbite d'origine est une orbite qui caractérise la création, la scission ou la fusion d'une autre orbite. D'un point de vue plus pratique, l'origine est : dans une création, l'orbite à partir de laquelle une autre entité est créée ; dans une scission, l'orbite qui permet d'en discriminer une autre parmi toutes celles issues de la scission ; dans une fusion, l'orbite qui permet d'identifier toutes les orbites fusionnées ensemble.

Définition 30 (Orbite d'origine)
Soit une règle $r : L \longrightarrow R$, n *un nœud de* R *et* $\langle\omega\rangle$ *un type d'orbite.*
Une orbite d'origine *de* $\underline{R}\langle\omega\rangle(n)$ *dans* L *est une orbite* $\underline{L}\langle\omega'\rangle(n')$ *où :*
- n' *est un nœud de* L,
- $\langle\omega'\rangle$ *est l'ensemble des dimensions* i, *où* i *est renommé dans un arc implicite de* n' ; *c'est-à-dire que* $(l_L(n') \mapsto l_R(n))(i) = j \in \langle\omega\rangle$

Soient un morphisme de filtrage $\underline{m} : \underline{L} \to G$ *sur une G-carte* G, $G \xrightarrow{r,\underline{m}} H$ *la transformation directe correspondante,* O *l'orbite d'instanciation et* v' *un brin de* O.
Une origine *de l'orbite* $R(O)\langle\omega\rangle((n, v'))$ *est une orbite* $L(O)\langle\omega'\rangle((n', v'))$.

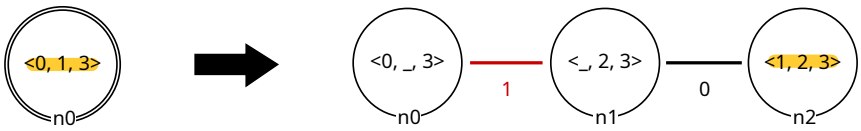


FIGURE 3.24 – Origine du sommet créé dans la règle de triangulation

(a) G (b) H

FIGURE 3.25 – Calcul de l'origine du sommet créé dans la triangulation

Dans notre exemple, le sommet créé est le dual de la face triangulée et donc, intuitivement, la face est l'origine du sommet. Plus formellement, nous pouvons voir dans la règle (figure 3.24) que l'orbite $R\langle 1,2,3\rangle(n2)$, les étiquettes 1, 2 et 3 sont des réécritures des étiquettes 0, 1 et 3 respectivement du nœud d'accroche. L'ensemble des étiquettes réécrites désignent alors une face $\langle 0,1,3\rangle$ comme origine du sommet. Ainsi, le sommet $H\langle 1,2,3\rangle(a_2)$ est créé à partir de la face $G\langle 0,1,3\rangle(a_0)$ dans la transformation directe (figures 3.25a et 3.25b).

4.2 Scission

Dans la section 3.2 (page 62), nous avons étudié la scission d'une face suite à sa transformation directe par la règle instanciée de triangulation. Nous allons maintenant montrer que la scission de la face $L\langle 0,1,3\rangle(n0)$ dans la règle Jerboa détermine, elle aussi, la scission d'une face dans la transformation directe.

Proposition 8 (Scission d'orbite)
Soient une règle $r : L \longrightarrow R$, un nœud d'accroche h de L, un morphisme de filtrage $\underline{m} : \underline{L} \rightarrow R$ sur une G-carte G, un type d'orbite $\langle\omega\rangle$, deux nœuds préservés v_1 et v_2 de $\underline{K} = \underline{L} \cap \underline{R}$. Notons $O = G\langle l(h)\rangle(\underline{m}(h))$ l'orbite d'instanciation.
$\underline{L}\langle\omega\rangle(v_1)$ est scindé dans la règle si :

- *la scission est due à des arcs explicites, c'est-à-dire si $\underline{L}\langle\omega\rangle(v_1)$ est scindée dans $\underline{R}$;*
- *la scission est due à des arcs implicites, c'est-à-dire si il existe un nœud v de $\underline{L}\langle\omega\rangle(v_1)$ dont l'étiquette $l(v)$ a pour k-ème position une dimension de $\langle\omega\rangle$, et que pour tous les nœuds w de $\underline{R}\langle\omega\rangle(v_1)$ l'étiquette de $l(w)$ a pour k-ème position un « _ » ou une dimension qui n'appartient pas à $\langle\omega\rangle$.*

1. *Si $\underline{L}\langle\omega\rangle(v_1)$ est scindée en deux orbites $\underline{R}\langle\omega\rangle(v_1)$ et $\underline{R}\langle\omega\rangle(v_2)$ dans la règle en raison d'arcs explicites. Et si l'orbite $\underline{R}\langle\omega\rangle(v_1)$ est complète. Alors, pour tout brin v' de l'orbite d'instanciation O, l'orbite $G\langle\omega\rangle(m((v_1,v')))$ est scindée en deux orbites $H\langle\omega\rangle(m'((v_1,v')))$ et $H\langle\omega\rangle(m'((v_2,v')))$ dans la transformation directe $G \xrightarrow{r,m} H$.*
2. *Si l'orbite $\underline{L}\langle\omega\rangle(v_1)$ est scindée en deux orbites $\underline{R}\langle\omega\rangle(v_1)$ et $\underline{R}\langle\omega\rangle(v_2)$ dans la règle en raison d'un arc explicite. Si les orbites $\underline{R}\langle\omega\rangle(v_1)$ et $\underline{R}\langle\omega\rangle(v_2)$ sont toutes deux incomplètes. Alors, pour chaque brin v' de l'orbite d'instanciation O, soit $G\langle\omega\rangle(m((v_1,v')))$ est scindée en deux orbites $H\langle\omega\rangle(m'((v_1,v')))$ et $H\langle\omega\rangle(m'((v_2,v')))$, soit l'orbite est modifiée, dans la transformation directe $G \xrightarrow{r,m} H$.*
3. *Si l'orbite $\underline{L}\langle\omega\rangle(v_1)$ est scindée dans la règle du au k-ième arc implicite. Alors pour tout brin v'_1 de l'orbite d'instanciation O et v'_2 le i-voisin de v'_1 dans O avec i la dimension à la k-ième position de $l(h)$, $G\langle\omega\rangle(m((v_1,v'_1)))$ est soit scindée en deux orbites $H\langle\omega\rangle(m'((v_1,v'_1)))$ et $H\langle\omega\rangle(m'((v_1,v'_2)))$, soit modifiée en une orbite unique $H\langle\omega\rangle(m'((v_1,v'_1))) = H\langle\omega\rangle(m((v_1,v'_2)))$ dans la transformation directe $G \xrightarrow{r,m} H$.*
4. *Réciproquement,*
   si l'orbite $G\langle\omega\rangle(m((v_1,v'_1))$ est scindée en deux orbites $H\langle\omega\rangle(m'((v_1,v'_1)))$ et $H\langle\omega\rangle(m'((v_1,v'_2)))$ alors il existe deux nœuds préservés v_3 et v_4 (de $\underline{K} = \underline{L} \cap \underline{R}$) tels que $m((v_3,v'_1)) \in G\langle\omega\rangle(m((v_1,v'_1))$ et $m((v_4,v'_1)) \in G\langle\omega\rangle(m((v_1,v'_1))$ et $\underline{L}\langle\omega\rangle(v_3)$ est scindée en $\underline{R}\langle\omega\rangle(v_3)$ et $\underline{R}\langle\omega\rangle(v_4)$.

Preuve : *Soit deux nœuds préservés v_1 et v_2 de $\underline{K} = \underline{L} \cap \underline{R}$*

1. *Supposons que $\underline{L}\langle\omega\rangle(v_1)$ est scindée en deux orbites $\underline{R}\langle\omega\rangle(v_1)$ et $\underline{R}\langle\omega\rangle(v_2)$, avec au moins l'une des deux complète, dans la règle en raison d'arcs explicites. Alors, par définition, il existe un chemin $e_1e_2...e_j$ dans $\underline{L}$ de source v_1, de cible v_2 et étiqueté par $i_1i_2...i_j$ sur $\langle\omega\rangle$. Par contre, il n'existe pas de tel chemin $f_1f_2...f_k$ dans R de source v_1, de cible v_2 et étiqueté sur $\langle\omega\rangle$.*

   Donc dans la règle instanciée, pour tout nœud v' de l'orbite d'instanciation $O = G\langle l(h)\rangle(\underline{m}(h))$, il existe un chemin $(e_1, v')(e_2, v')...(e_j, v')$ dans le motif gauche de la règle instanciée $L(O)$ dont la source est $(s(e_1), v') = (v_1, v')$, la cible est $(t(e_j), v') = (v_2, v')$ et étiqueté par le même mot $i_1i_2...i_j$ sur $\langle\omega\rangle$.

   Raisonnons par l'absurde. Supposons que pour un nœud v' de l'orbite d'instanciation O, il existe un chemin $(f_1, v')(f_2, v')...(f_k, v')$ dans le motif droit de la règle instanciée $R(O)$ dont la source est $(s(f_1), v') = (v_1, v')$, la cible est $(t(e_j), v') = (v_2, v')$ et étiqueté sur $\langle\omega\rangle$. Alors, par construction de l'instanciation, le chemin $f_1f_2...f_k$ dans R de source v_1, de cible v_2 et étiqueté sur $\langle\omega\rangle$ existe dans la règle de départ. Ce qui est contradictoire avec l'hypothèse. Donc il n'existe pas de chemin $(f_1, v')(f_2, v')...(f_k, v')$ dans le motif droit de la règle instanciée ayant les propriétés attendues. En conséquence, $R(O)\langle\omega\rangle((v_1, v'))$ et $R(O)\langle\omega\rangle((v_2, v'))$ sont deux $\langle\omega\rangle$-orbites distinctes.

   D'après la proposition 2, si $\underline{R}\langle\omega\rangle(v_1)$ et donc $R(O)\langle\omega\rangle((v_1, v'))$ est entièrement filtrée, alors l'orbite $G\langle\omega\rangle(m((v_1, v')))$ est scindée en deux orbites $H\langle\omega\rangle(m'((v_1, v')))$ et $H\langle\omega\rangle(m'((v_2, v')))$ dans la transformation directe $G \xrightarrow{r,m} H$ pour tout nœud v' de l'orbite d'instanciation O.

2. *Supposons que $\underline{L}\langle\omega\rangle(v_1)$ est scindée en deux orbites incomplètes $\underline{R}\langle\omega\rangle(v_1)$ et $\underline{R}\langle\omega\rangle(v_2)$ dans la règle en raison d'arcs explicites.*

   Pour les mêmes raisons que dans le cas 1, $R(O)\langle\omega\rangle((v_1, v'))$ et $R(O)\langle\omega\rangle((v_2, v'))$ sont deux $\langle\omega\rangle$-orbites distinctes dans la règle instanciée, pour tout nœud v' de l'orbite d'instanciation O.

   En conséquence, d'après la proposition 2, si $\underline{R}\langle\omega\rangle(v_1)$ et donc $R(O)\langle\omega\rangle((v_1, v'))$ est incomplète, alors, pour tout nœud v' de l'orbite d'instanciation O, l'orbite $G\langle\omega\rangle(m((v_1, v')))$ est soit scindée en deux orbites $H\langle\omega\rangle(m'((v_1, v')))$ et $H\langle\omega\rangle(m'((v_2, v')))$, soit modifiée dans la transformation directe $G \xrightarrow{r,m} H$.

3. *Supposons que $\underline{L}\langle\omega\rangle(v_1)$ est scindée en raison du k-ième arc implicite. Soit v'_1 un nœud de l'orbite d'instanciation $O = G\langle l(h)\rangle(\underline{m}(h))$ et v'_2 le i-voisin de v'_1 dans O, avec i la k-ième dimension de l'étiquette $l(h)$ du nœud d'accroche h de L.*

   Par hypothèse, l'un des nœuds v de $\underline{L}\langle\omega\rangle(v_1)$ a une k-ième dimension j dans l'$\langle\omega\rangle$-orbite ($j \in \langle\omega\rangle$), alors que tous les nœuds de $\underline{R}\langle\omega\rangle(v_1)$ ont leur k-ième dimension hors de $\langle\omega\rangle$. Ainsi, les nœuds (v, v'_1) et (v, v'_2) sont j-voisins dans l'instanciation $L(O)$, donc (v, v'_1) et (v, v'_2) sont dans la même $\langle\omega\rangle$-orbite dans $L(O)$. On en déduit que (v_1, v'_1) et (v_2, v'_2) aussi.

   Nous avons plusieurs cas :

   — *si (v_1, v'_1) et (v_2, v'_2) sont dans deux $\langle\omega\rangle$-orbites différentes de $R(O)$, d'après la proposition 2, selon les cas, $H\langle\omega\rangle(m'((v_1, v'_1)))$ et $H\langle\omega\rangle(m'((v_1, v'_2)))$ sont deux orbites différentes ou non de H, avec H le résultat de la transformation directe $G \xrightarrow{r,m} H$. Autrement dit, $G\langle\omega\rangle(m((v_1, v'_1))$ est scindée en $H\langle\omega\rangle(m'((v_1, v'_1)))$ et $H\langle\omega\rangle(m'((v_1, v'_2)))$ ou juste modifiée par la transformation directe $G \xrightarrow{r,m} H$;*

— *si au contraire* (v_1, v_1') *et* (v_2, v_2') *sont dans la même* $\langle\omega\rangle$*-orbites de* $R(O)$ *(car il existe un chemin* $e_1e_2...e_l$ *de* O *de source* v_1' *et de cible* v_2' *qui ne comprend pas l'étiquette* j*, ce chemin est réécrit en un chemin* $(w, e_1)(w, e_2)...(w, e_l)$ *de* $R(O)$ *dont toutes les étiquettes sont dans* $\langle\omega\rangle$*). D'après la proposition 6, on en déduit que* $G\langle\omega\rangle(m((v_1, v_1'))$ *est modifiée par la transformation directe* $G \xrightarrow{r,m} H$*.*

4. *Soient* $m((v_1, v_1'))$ *et* $m((v_2, v_2'))$ *deux brins de* G *filtrés par la règle avec* v_1 *et* v_2 *deux nœuds de* $\underline{K} = \underline{L} \cap \underline{R}$ *et* v_1' *et* v_2' *deux brins de* O*. Supposons que l'orbite* $G\langle\omega\rangle(m((v_1, v')))$ *est scindée en deux orbites* $H\langle\omega\rangle(m'((v_1, v_1')))$ *et* $H\langle\omega\rangle(m'((v_2, v_2')))$ *dans la transformation directe. D'après la proposition 2, il existe deux nœuds* k_1 *et* k_2 *préservés de la règle instanciée (de* $K(O) = L(O) \cap R(O)$*) tels que* $m(k_1) \in G\langle\omega\rangle(m((v_1, v_1'))$ *et* $m(k_2) \in G\langle\omega\rangle(m((v_2, v_2')$ *et* $L(O)\langle\omega\rangle(k_1)$ *est scindée en* $R(O)\langle\omega\rangle(k_1)$ *et* $R(O)\langle\omega\rangle(k_2)$*. Notons* $k_1 = (v_3, v_3')$ *et* $k_2 = (v_4, v_4')$ *avec* v_3 *et* v_4 *deux nœuds préservés de* $\underline{K} = \underline{L} \cap \underline{R}$ *et* v_3' *et* v_4' *deux brins de* O*. Nous avons donc* $L(O)\langle\omega\rangle((v_3, v_3'))$ *scindée en* $R(O)\langle\omega\rangle((v_3, v_3'))$ *et* $R(O)\langle\omega\rangle((v_4, v_4'))$*. Il existe donc au moins chemin dans* $L(O)$ *entre* (v_3, v_3') *et* (v_4, v_4')*, mais aucun chemin dans* $R(O)$ *entre* (v_3, v_3') *et* (v_4, v_4')*. En conséquence, par construction de l'instanciation* $L(O)$*, il existe un chemin entre* (v_3, v_3') *et* (v_4, v_3') *(dont tous les nœuds sont de la forme* (v_i, v_3')*) et un chemin entre* (v_4, v_3') *et* (v_4, v_4') *(dont tous les nœuds sont de la forme* (v_4, v_i')*).*

Raisonnons par l'absurde. Supposons que la règle Jerboa ne comporte aucune scission. Par construction de l'instanciation :

— *si la règle Jerboa n'a pas de scission explicite entre* v_3 *et* v_4*, c'est-à-dire que s'il existe un chemin entre* v_3 *et* v_4 *dans* L *alors il en existe un dans* R*. Alors, d'après l'instanciation, s'il existe un chemin de* (v_3, v_3') *vers* (v_4, v_3') *dans* $L(O)$*, il en existe un dans* $R(O)$*;*

— *si la règle Jerboa n'a pas de scission implicite entre* $\underline{L}\langle\omega\rangle(v_3)$ *et* $\underline{R}\langle\omega\rangle(v_3)$*. Alors, d'après l'instanciation, s'il existe un chemin de* (v_4, v_3') *vers* (v_4, v_4') *dans* $L(O)$*, il en existe un dans* $R(O)$*.*

Il existe donc un chemin dans $R(O)$ *de* (v_3, v_3') *vers* (v_4, v_4')*, ce qui est contradictoire avec l'hypothèse. En conséquence, la règle Jerboa comprend au moins une scission implicite ou explicite.* □

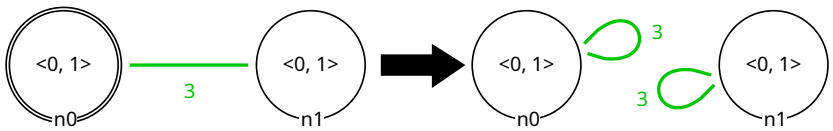
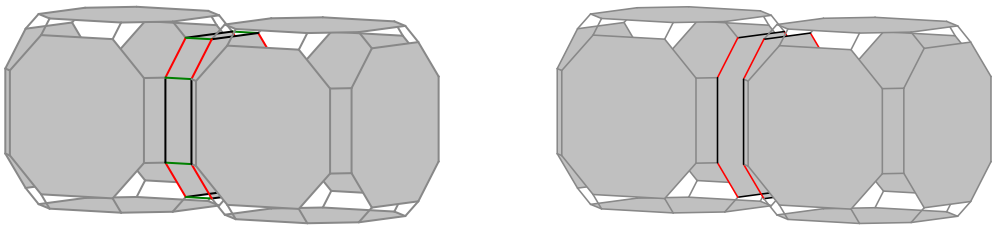


Figure 3.26 – Règle de décousure de deux faces liées par des 3-liaisons

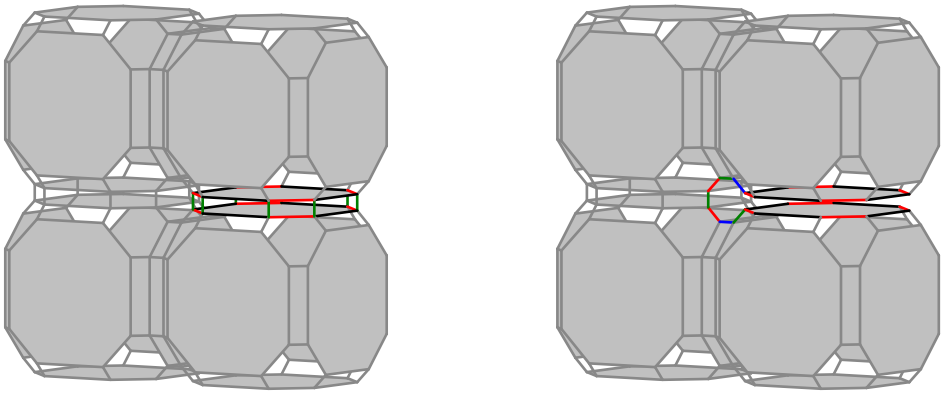
Pour illustrer le premier point, une scission explicite où il existe au moins une $\langle\omega\rangle$-orbite complète, nous allons étudier la règle de décousure des 3-arcs d'une face (figure 3.26). L'orbite $L\langle 0, 1, 3\rangle(n0)$ est complète, de même, les deux orbites scindées sont, elles aussi, complètes. Puisque dans la règle au moins l'une des $\langle 0, 1, 3\rangle$-orbites est complète, alors tous les chemins entre les deux faces à séparer sont représentés. En conséquence, la scission du 3-arc explicite dans la règle provoque, comme nous pouvons le voir dans la transformation directe (figure 3.27), la scission de toutes les 3-liaisons entre les deux faces.

Pour le deuxième point, une scission explicite où il n'y a aucune $\langle\omega\rangle$-orbite complète, nous gardons la même règle, mais nous étudions cette fois une $\langle 0, 1, 2, 3\rangle$-orbite, c'est-à-dire toute



(a) G (b) H

FIGURE 3.27 – Décousure des 3-liaisons dans une face



(a) G (b) H

FIGURE 3.28 – Absence de scission de composante connexe

la composante connexe. Dans cette règle, comme dans le point précédent, la coupure du 3-arc permet de détecter une scission même pour $L\langle 0,1,2,3\rangle(n0)$. Cependant cette orbite là est incomplète. Ainsi, la règle ne provoque pas forcément de scission dans la transformation directe. Par exemple, dans la transformation illustrée figures 3.28a et 3.28b, la scission d'une face ne suffit pas à scinder toute la composante. Nous pouvons encore y voir au moins un chemin qui relie les deux volumes par des arcs étiquetés dans $\langle 0,1,2,3\rangle$.

Pour le troisième point, commençons avec la règle de triangulation dont les $\langle 0,1,3\rangle$-orbites sont complètes. La scission est due aux arcs implicites et plus particulièrement à la réécriture des 1-arcs. Dans chaque nœud de $R\langle 0,1,3\rangle(n0)$, les arcs implicites sont réécrits à partir du 1-arc de $L\langle 0,1,3\rangle(n0)$. Ils sont soit supprimés, soit réécris en 2-arcs. Dans les instanciations de la règle (figure 3.22), nous pouvons voir que les arêtes des faces filtrées sont séparées par la suppression des 1-liaisons et que les faces issues de la scission sont elles-mêmes distinctes, car il n'existe pas de 1-chemin qui connecte deux faces.

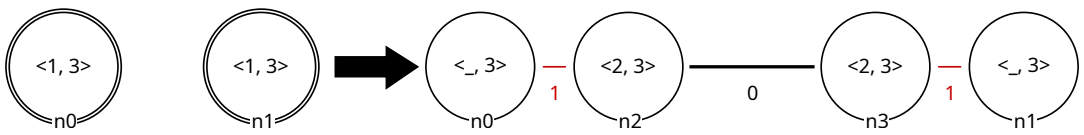


FIGURE 3.29 – Règle d'insertion d'arête entre deux sommets

Prenons maintenant la règle d'insertion d'arête (figure 3.29). Une scission est détectée dans l'orbite $R\langle 0,1,3\rangle(n0)$ en raison de la suppression des 1-arcs implicites. Toutefois, cette règle n'a aucune $\langle 0,1,3\rangle$-orbite complète. Il est donc nécessaire de vérifier si la règle provoque une scission dans la transformation directe. Dans ce cas, Si les orbites filtrées appartiennent à deux

faces distinctes, alors la règle ne provoque pas de scission dans la transformation directe, comme dans l'exemple du cylindre (figures 3.5c et 3.5d).

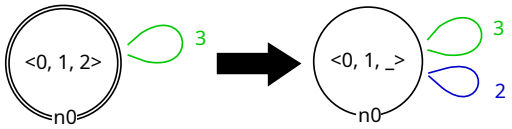
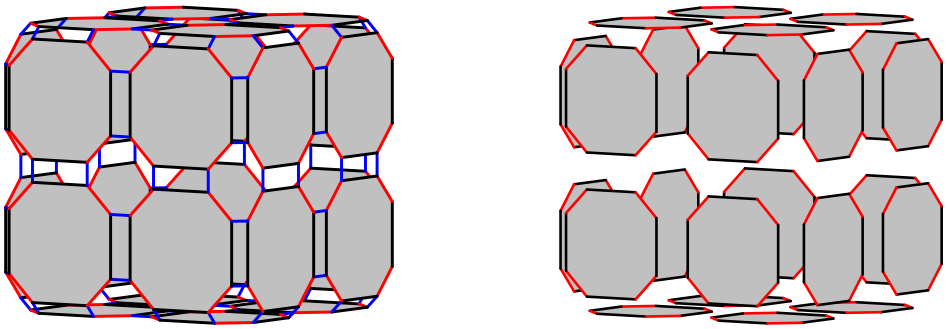


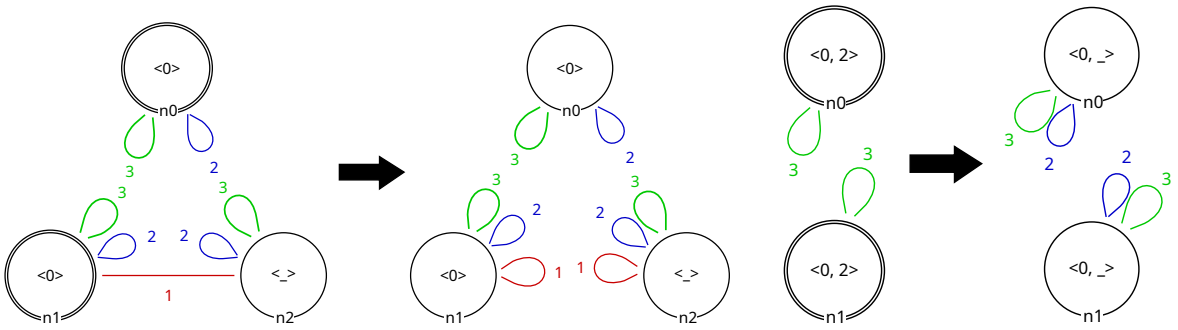
Figure 3.30 – Règle de décousure des 2-liaisons d'un volume



(a) G (b) H

Figure 3.31 – Décousure des faces d'un volume

Nous étudions à présent ce même cas, mais avec la règle de décousure des 2-liaisons dans un volume, c'est-à-dire une $\langle 0, 1, 2\rangle$-orbite. Dans cette règle, la scission est due à la suppression des 2-arcs implicites Dans ce cas, l'orbite est complète et son application déconnecte toutes les faces d'un volume (figure 3.31). Toutefois, si le volume correspond à une unique face, la scission sera tout de même détectée. Dans une telle configuration, vérifier localement qu'il existe au moins une 2-liaison qui n'est pas une boucle permet de confirmer qu'une scission a bien eu lieu dans la transformation directe.



(a) Règle de décousure des sommets d'une arête (b) Règle de double décousure

Figure 3.32 – Règles de décousures

Enfin, nous considérons le quatrième cas où une règle filtre deux composantes, mais où la scission ne concerne que l'une des deux orbites dans la règle. Prenons une première règle qui découd les sommets d'une arête (figure 3.32a). Les $\langle 0, 1, 3\rangle$-orbites incidentes à $n0$ et $n1$ sont toutes deux incomplètes et l'orbite $R\langle 0, 1, 3\rangle(n1)$ est scindée par son 1-arc explicite. Puisqu'aucune des orbites n'est complète, il est possible que les orbites filtrées appartiennent à la même

face. Dans ce cas, il y peut y avoir une scission entre les deux orbites filtrées comme dans les figures 3.7a et 3.7b (page 65).

Il s'agit du même principe pour la règle qui fait une double décousure (figure 3.32b). Les $\langle 0, 1, 2\rangle$-orbites incidentes à $n0$ et $n1$ sont toutes deux incomplètes. Dans ce cas, il n'y a pas forcément de scission. Appliquée à un ruban (figure 3.9), une seule des deux scissions est nécessaire pour que le ruban soit effectivement scindé, comme nous pouvons l'observer dans les figures. Autre cas, si les deux orbites désignent les extrémités du ruban, il ne se passe rien, car les 2-liaisons sont déjà des boucles.

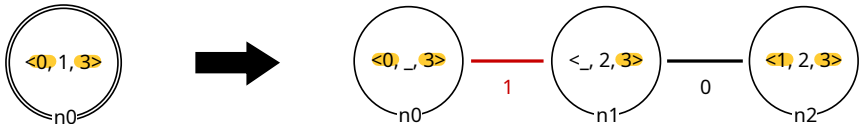


FIGURE 3.33 – Calcul de l'origine d'une face triangulée

(a) G (b) H

FIGURE 3.34 – Origines des faces issues d'une triangulation

La scission est le deuxième évènement pour lequel il est possible de calculer un type d'origine. Calculons le type d'origine $\langle \omega' \rangle$ d'une face dans la règle de triangulation (figure 3.33). Dans cet exemple, le nœud d'accroche contient trois dimensions. $\langle \omega' \rangle$ contient donc trois dimensions. Dans l'orbite $R\langle 0, 1, 3\rangle(n0)$ les réécritures sont les suivantes :

— le premier arc est soit réécrit en 0 soit supprimé, donc nous ajoutons 0 à $\langle \omega' \rangle$;
— le deuxième arc implicite est soit supprimé, soit réécrit en 2 qui n'est pas une dimension de $\langle \omega \rangle$. Nous ne l'ajoutons donc pas à $\langle \omega' \rangle$;
— le troisième arc implicite est systématiquement réécrit en 3, nous l'ajoutons donc à $\langle \omega' \rangle$.

Enfin, nous obtenons $\langle \omega' \rangle = \langle 0, 3\rangle$. Ainsi chaque face issue de la triangulation a pour origine une arête préservée de la règle. Par exemple, la face $H\langle 0, 1, 3\rangle(c_0)$ a pour origine l'arête $G\langle 0, 3\rangle(c_0)$ (figures 3.34a et 3.34b).

4.3 Suppression

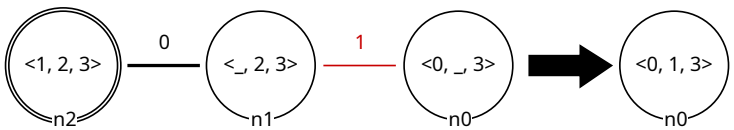


FIGURE 3.35 – Règle de suppression de l'étoile (d'un sommet et de ses arêtes incidentes)

Reprenons la règle qui supprime un sommet et son étoile (figure 3.35). Dans la section 3.3 (page 66), nous avons vu qu'un sommet est supprimé dans la transformation si et seulement si il est supprimé dans la règle instanciée. Nous allons montrer que la suppression du sommet $L\langle 1,2,3\rangle(n2)$ dans la règle Jerboa exprime la suppression d'un sommet dans la transformation directe.

Proposition 9 (Suppression d'orbite)
Soient une règle $r : L \longrightarrow R$, un nœud d'accroche h de L, un morphisme de filtrage $\underline{m} : \underline{L} \to G$ sur une G-carte G, un type d'orbite $\langle\omega\rangle$, v un nœud de L et $O = G\langle l(h)\rangle(\underline{m}(h))$.
L'orbite $\underline{L}\langle\omega\rangle(v)$ est supprimée dans la règle Jerboa si et seulement si, pour tout brin v' de O, l'orbite $G\langle\omega\rangle(m((v,v')))$ est supprimée dans la transformation directe $G \xrightarrow{r,\underline{m}} H$.

Preuve : *Cette preuve est symétrique à celle de la création (voir la proposition 7).* □

(a) G (b) H

Figure 3.36 – Suppression de l'étoile sur un cube

D'après la condition d'arcs incidents du théorème 2 (page 17), une orbite supprimée est nécessairement complète. En conséquence, l'orbite à supprimer dans la règle instanciée est, elle aussi, complète. L'orbite dans la G-carte est donc supprimée dans la transformation directe. Dans notre exemple (figure 3.35), l'orbite $L\langle 1,2,3\rangle(n0)$ est complète. Ainsi, sa suppression dans la règle implique la suppression du sommet filtrée dans la transformation directe (figures 3.36a3.36b).

4.4 Fusion

Reprenons la règle qui supprime une étoile (figure 3.35). Dans la section 3.4 (page 67), nous avons vu que les faces incidentes au sommet supprimé sont fusionnées dans la transformation directe si elles sont transformées dans la règle instanciée. Nous allons montrer que la fusion des $\langle 0,1,3\rangle$-orbites dans la règle implique une fusion de faces dans la transformation directe.

Proposition 10 (Fusion d'orbites)
Soient une règle Jerboa $r : L \longrightarrow R$, un nœud d'accroche h de L, un morphisme de filtrage $\underline{m} : \underline{L} \to G$ sur une G-carte G, un type d'orbite $\langle\omega\rangle$, deux nœuds préservés v_1 et v_2 de $\underline{K} = \underline{L} \cap \underline{R}$ et l'orbite d'instanciation $O = G\langle l(h)\rangle(\underline{m}(h))$.
$\underline{L}\langle\omega\rangle(v_1)$ est fusionnée dans la règle :
- *si la fusion est due à des arcs explicites, c'est-à-dire si $\underline{L}\langle\omega\rangle(v_1)$ est fusionnée avec $\underline{L}\langle\omega\rangle(v_2)$ dans $\underline{R}$;*
- *si la fusion est due à des arcs implicites, c'est-à-dire si pour tout nœud v de $\underline{L}\langle\omega\rangle(v_1)$, l'étiquette $l(v)$ a pour k-ème position un « _ » ou une dimension qui n'appartient pas à $\langle\omega\rangle$, et qu'il existe un nœud w de $\underline{R}\langle\omega\rangle(v_1)$ dont l'étiquette $l(w)$ a pour k-ème*

position une dimension de $\langle\omega\rangle$.

1. *Si $\underline{L}\langle\omega\rangle(v_1)$ et $\underline{L}\langle\omega\rangle(v_2)$ sont fusionnées en une orbite $\underline{R}\langle\omega\rangle(v_1)$ dans la règle en raison d'arcs explicites. Si, de plus, l'orbite $\underline{L}\langle\omega\rangle(v_1)$ est complète. Alors, pour tout brin v' de O, les orbites $G\langle\omega\rangle(m((v_1,v')))$ et $G\langle\omega\rangle(m((v_2,v')))$ sont fusionnées en une orbite $H\langle\omega\rangle(m'((v_1,v')))$ dans la transformation directe $G \xrightarrow{r,m} H$.*
2. *Si les orbites $\underline{L}\langle\omega\rangle(v_1)$ et $\underline{L}\langle\omega\rangle(v_2)$ sont fusionnées en une orbite $\underline{R}\langle\omega\rangle(v_1)$ dans la règle en raison d'arcs explicites. Si, de plus, les orbites $\underline{L}\langle\omega\rangle(v_1)$ et $\underline{L}\langle\omega\rangle(v_2)$ sont toutes deux incomplètes. Alors, pour chaque brin v' de O, soit $G\langle\omega\rangle(m((v_1,v')))$ et $G\langle\omega\rangle(m((v_2,v')))$ sont fusionnées en une orbite $H\langle\omega\rangle(m'((v_1,v')))$, soit elles sont la même orbite modifiée dans la transformation directe $G \xrightarrow{r,m} H$.*
3. *Si l'orbite $\underline{L}\langle\omega\rangle(v_1)$ est fusionnées dans la règle du au k-ème arc implicite. Alors pour tout brin v'_1 de O et v'_2 le i-voisin de v'_1 dans O avec i la dimension à la k-ème position de $l(h)$, $G\langle\omega\rangle(m((v_1,v'_1)))$ et $G\langle\omega\rangle(m((v_1,v'_2)))$ sont soit fusionnées en une orbite $H\langle\omega\rangle(m'((v_1,v'_1)))$, soit c'est une même orbite modifiée $H\langle\omega\rangle(m'((v_1,v'_1))) = H\langle\omega\rangle(m'((v_1,v'_2)))$ dans la transformation directe $G \xrightarrow{r,m} H$*
4. *Réciproquement,*
   si les orbites $L\langle\omega\rangle(m((v_1,v'_1))$ et $L\langle\omega\rangle(m((v_2,v'_2))$ sont fusionnées en une orbite $H\langle\omega\rangle(m'((v_1,v'_1)))$, alors il existe deux nœuds préservés v_3 et v_4 (de $\underline{K} = \underline{L} \cap \underline{R}$) tels que $m((v_3,v'_1)) \in H\langle\omega\rangle(m'((v_1,v'_1))$, $m((v_4,v'_1)) \in H\langle\omega\rangle(m'((v_1,v'_1))$, $\underline{L}\langle\omega\rangle(v_3)$ et $\underline{L}\langle\omega\rangle(v_4)$ sont fusionnées en $\underline{R}\langle\omega\rangle(v_3) = \underline{R}\langle\omega\rangle(v_4)$.

Preuve : *Cette preuve est symétrique à celle de la scission (voir la proposition 8).* □

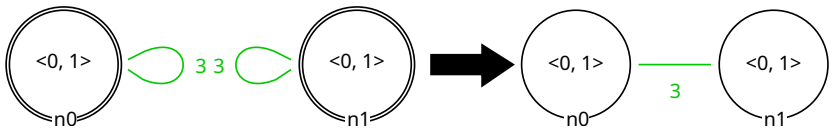
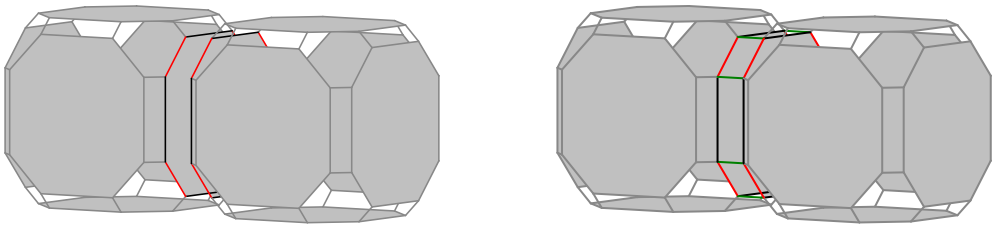


FIGURE 3.37 – Règle de coutures de faces de volumes

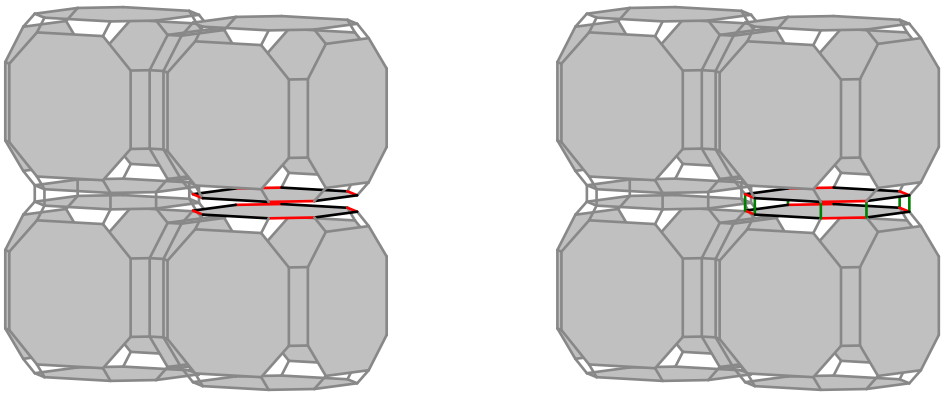


(a) G (b) H

FIGURE 3.38 – Décousure des 3-liaisons dans une face

L'approche pour détecter la fusion est symétrique à celle de la scission. Ainsi, nous retrouvons les mêmes cas de figure. Lorsqu'une fusion est explicite et que la règle possède au moins une $\langle\omega\rangle$-orbite complète, les deux faces fusionnées dans la règle sont aussi fusionnées dans la transformation directe (figures 3.38a et 3.38b). C'est par exemple le cas de la règle de couture de deux faces par des 3-arcs.

Si, au contraire, cette même règle ne possède aucune $\langle\omega\rangle$-orbite complète, par exemple pour $\langle\omega\rangle = \langle 0,1,2,3\rangle$, alors il est nécessaire de vérifier si cette règle provoque une fusion dans la



(a) G (b) H

Figure 3.39 – Absence de fusion de composante connexe

transformation directe. Nous pouvons voir dans les figures 3.39a et 3.39b que les deux faces fusionnées appartiennent déjà à la même $\langle 0, 1, 2, 3\rangle$-orbite. Il n'y a donc pas de nouvelle fusion pour ce type d'orbite.

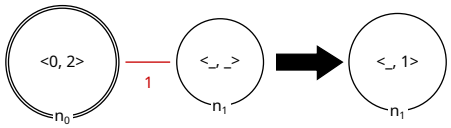


Figure 3.40 – Règle de suppression d'une arête

Pour les fusions implicites, la fusion des faces dans la règle de suppression d'une étoile (figure 3.35) est garantie ($\langle 0, 1, 3\rangle$-orbite complète). La fusion des faces incidentes à une arête (figure 3.40) est incertaine ($\langle 0, 1, 3\rangle$-orbite incomplète).

Notons que pour la fusion implicite, contrairement à la scission, il n'existe pas de règle impliquant qu'une orbite est strictement identique à son image par la transformation directe. En effet, si dans le cadre d'une scission il est possible réécrire des arcs sous forme de boucles (figure 3.26), il est en revanche impossible de réaliser le procédé inverse sans transgresser les conditions de cohérence des règles Jerboa. Par exemple, il est impossible d'inverser le sens de la transformation de cette même règle, car elle serait alors invalide : un nœud d'accroche doit avoir une étiquette pleine.

Pour le quatrième point, les règles figures 3.6 et 3.8 ne sont pas inversibles en utilisant des arcs implicites. Nous utilisons donc leurs versions dépliées, c'est-à-dire sans arcs implicites, équivalentes aux règles instanciées (figures 3.13 et 3.15) (page 69). L'observation est donc la même, le traitement de la fusion est symétrique à celui de la scission. Pour la règle de couture des sommets, la reconnexion à au moins un sommet est suffisante, tandis que pour la double couture des arêtes, l'ensemble des fusions est nécessaire pour recréer un chemin entre les différentes orbites filtrées.

Enfin, la fusion est le troisième évènement pour lequel nous pouvons déterminer l'origine d'une orbite. L'analyse de la règle révèle (figure 3.41) que les arcs de l'orbite $R\langle 0, 1, 3\rangle(n0)$ sont des réécritures des arcs 1, 2 et 3 du nœud d'accroche. La face $R\langle 0, 1, 3\rangle(n0)$ a donc pour origine d'origine le sommet $L\langle 1, 2, 3\rangle(n2)$. Ainsi, la face $H\langle 0, 1, 3\rangle(a_0)$ a pour origine le sommet $G\langle 1, 2, 3\rangle(a_2)$ dans la transformation directe (figures 3.42a et 3.42b).

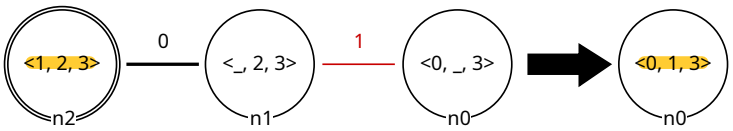


FIGURE 3.41 – Origine de la face dans la règle de suppression d'une étoile

(a) G (b) H

FIGURE 3.42 – Calcul de l'origine du sommet créé dans la triangulation

4.5 Non-changement

Reprenons la règle de triangulation (figure 3.21 page 73). Dans la section 3.5 (page 70), nous avons vu que les arêtes filtrées de la face triangulée restent inchangées dans la transformation directe si et seulement si elles restent inchangées dans la règle instanciée. Nous allons montrer que le non-changement d'une orbite filtrée dans la règle Jerboa exprime le non-changement d'une orbite dans la transformation directe.

Proposition 11 (Non-changement d'orbite)
Soient une règle Jerboa $r : L \longrightarrow R$, un nœud d'accroche h de L, un morphisme de filtrage $\underline{m} : \underline{L} \rightarrow G$ sur une G-carte G, un type d'orbite $\langle\omega\rangle$, v un nœud de L et l'orbite d'instanciation $O = G\langle l(h)\rangle(\underline{m}(h))$.
L'orbite $\underline{L}\langle\omega\rangle(v)$ est inchangée dans la règle si et seulement si, pour tout brin v' de O, l'orbite $G\langle\omega\rangle(m((v,v')))$ est inchangée dans la transformation directe $G \xrightarrow{r,m} H$.

Preuve : *Supposons que $\underline{L}\langle\omega\rangle(v)$ est inchangée dans la règle Jerboa.*
Soit w un nœud quelconque de l'orbite $\underline{L}\langle\omega\rangle(v)$. w est un donc nœud inchangé de la règle de $\underline{K} = \underline{L} \cap \underline{R}$. Par construction de la règle instanciée, pour tout brin v' de O, le nœud (w,v') est inchangé dans la règle instanciée $r(\underline{m})$. Or l'orbite $G\langle\omega\rangle(m((v,v')))$ n'est pas nécessairement complète. Par définition, la partie non filtrée est aussi inchangée. Donc $L(O)\langle\omega\rangle((v,v'))$ est inchangée dans la règle instanciée. Par conséquent, d'après la proposition 5, pour tout brin v' de O, l'orbite $G\langle\omega\rangle(m((v,v')))$ est inchangée dans la transformation directe $G \xrightarrow{r,m} H$.
Réciproquement, supposons que pour tout brin v' de O l'orbite $G\langle\omega\rangle(m((v,v')))$ est inchangée dans la transformation directe $G \xrightarrow{r,m} H$. D'après la proposition 5, l'orbite $L(O)\langle\omega\rangle((v,v'))$ est inchangée dans la règle instanciée. Par conséquent, par construction de l'instanciation, l'orbite $\underline{L}\langle\omega\rangle(v)$ est inchangée dans la règle Jerboa. □

Dans la règle de triangulation, les arêtes de la face à trianguler sont filtrées par l'orbite $L\langle 0\rangle(n0)$. Or, cette même orbite est préservée dans R puisqu'aucun de ses 0-arcs n'est supprimé ni réécrit en un arc d'une autre dimension. Ainsi, les arêtes de la face sont préservées dans la transformation directe (figures 3.43a et 3.43b).

(a) G (b) H

Figure 3.43 – Non-changement des arêtes de la face triangulée

Remarque : *L'orbite $G\langle\omega\rangle(b)$, où b est un brin non filtré de G, n'est* pas affectée *par l'application de r si $G\langle\omega\rangle(b) \cap \underline{m}(L) = \emptyset$ et* inchangée *sinon.*

4.6 Modification

Dans la section 3.6, nous avons vu que les sommets de la face à trianguler sont modifiés dans la transformation directe si et seulement si ils sont modifiés dans la règle instanciée. Nous allons montrer que la modification de ces sommets dans la règle Jerboa exprime leur modification dans la transformation directe.

Proposition 12 (Modification d'orbite)
Soient une règle Jerboa $r : L \longrightarrow R$, un nœud d'accroche h de L, un morphisme de filtrage $\underline{m} : \underline{L} \to G$ sur une G-carte G, un type d'orbite $\langle\omega\rangle$, v un nœud préservé de $\underline{K} = \underline{L} \cap \underline{R}$ et l'orbite d'instanciation $O = G\langle l(h)\rangle(\underline{m}(h))$.
L'orbite $\underline{L}\langle\omega\rangle(v)$ est modifiée dans la règle si et seulement si :

- *si la modification est due à des arcs explicites, alors :*
  - *$\underline{L}\langle\omega\rangle(v) \neq \underline{R}\langle\omega\rangle(v)$,*
  - *pour tout nœud préservé w de $\underline{K} = \underline{L} \cap \underline{R}$, $w \in \underline{L}\langle\omega\rangle(v)$ si et seulement si $w \in \underline{R}\langle\omega\rangle(v)$,*
  - *pour toute position d'étiquette k, il existe un nœud $w \in \underline{L}\langle\omega\rangle(v)$ dont la k-ème étiquette de $l(w)$ est une dimension de $\langle\omega\rangle$ si et seulement si il existe un nœud $w' \in \underline{R}\langle\omega\rangle(v)$ dont la k-ème étiquette de $l(w')$ est une dimension de $\langle\omega\rangle$;*
- *si la modification est due à des arcs implicites, alors :*
  - *$\underline{L}\langle\omega\rangle(v) \neq \underline{R}\langle\omega\rangle(v)$,*
  - *il existe un nœud préservé $w \in \underline{L}\langle\omega\rangle(v)$ tel que l'étiquette de w est différente dans L et R,*
  - *pour toute position d'étiquette k, il existe un nœud $w \in \underline{L}\langle\omega\rangle(v)$ dont la k-ème étiquette de $l(w)$ est une dimension de $\langle\omega\rangle$ si et seulement si il existe un nœud $w' \in \underline{R}\langle\omega\rangle(v)$ dont la k-ème étiquette de $l(w')$ est une dimension de $\langle\omega\rangle$;*

Si l'orbite $\underline{L}\langle\omega\rangle(v)$ est modifiée dans la règle Jerboa, alors, pour tout brin v' de O, $G\langle\omega\rangle(m((v,v')))$ est modifiée dans la transformation directe $G \xrightarrow{r,\underline{m}} H$. Réciproquement, si $G\langle\omega\rangle(m((v,v')))$ est modifiée dans la transformation directe $G \xrightarrow{r,\underline{m}} H$, alors $\underline{L}\langle\omega\rangle(v)$ est modifiée, scindée ou fusionnée dans la règle.

Preuve : *Supposons que l'orbite $\underline{L}\langle\omega\rangle(v)$ soit modifiée dans la règle en raison des arcs explicites. Montrons que pour tout brin v' de l'orbite d'instanciation O, l'orbite $L(O)\langle\omega\rangle((v,v'))$ est modifiée dans la règle instanciée.*

D'après le premier point, les orbites $L(O)\langle\omega\rangle((v,v'))$ et $R(O)\langle\omega\rangle((v,v'))$ sont différentes. Elles ne peuvent donc pas être inchangées. Il nous reste donc à montrer qu'elles ne sont ni scindées ni fusionnées.
Soient un nœud préservé w de $\underline{K} = \underline{L} \cap \underline{R}$ et deux brins v'_1 et v'_2 de l'orbite d'instanciation O. Supposons que (w, v'_1) et (v, v'_2) soient dans la même $\langle\omega\rangle$-orbite. Alors, il existe un chemin $c(O)$ dans $L(O)$ étiqueté sur $\langle\omega\rangle$ de source (w, v'_1) et de cible (v, v'_2). Les arcs de ce chemin issus d'arcs explicites de L, forment un chemin c de L entre w et v qui s'instancie en un chemin entre (w, v'_1) et (v, v'_1) dans $L(O)$. Or, par hypothèse, il existe un chemin issu d'arcs explicites de R, qui forme un chemin de R entre w et v et qui s'instancie en un chemin entre (w, v'_1) et (v, v'_1) dans $R(O)$. Les arcs de $c(O)$ issus d'arcs implicites de L forment un chemin de O entre v'_1 et v'_2. Or, pour chacun de ces arcs de v'_i à v'_j correspond un arc implicite de $L\langle\omega\rangle(v)$. Et il existe un arc implicite dans $R\langle\omega\rangle(v)$ étiqueté sur $\langle\omega\rangle$, éventuellement sur un autre nœud $w' \in \underline{R}\langle\omega\rangle(v)$. Nous pouvons donc construire un chemin dans $R(O)$ qui utilise l'instanciation d'arcs explicites de (w, v'_i) à (w', v'_i), puis l'instanciation de l'arc implicite pour passer à (w', v'_j), puis à nouveau l'instanciation d'arcs explicites pour passer à (w, v'_j). Nous pouvons donc construire un chemin de (w, v'_1) à (w, v'_2) dans $R(O)$. Nous avons donc construit un chemin de (w, v'_1) à (v, v'_2) dans $R(O)$ étiqueté sur $\langle\omega\rangle$.
La construction inverse d'un chemin de $R(O)$ vers un chemin de $L(O)$ peut se faire de la même manière. Donc pour tout nœud v' de l'orbite d'instanciation O, l'orbite $L(O)\langle\omega\rangle((v,v'))$ est modifiée dans la règle instanciée.

Supposons que l'orbite $\underline{L}\langle\omega\rangle(v)$ est modifiée dans la règle en raison d'arcs implicites. Montrons que pour tout nœud v' de l'orbite d'instanciation O, l'orbite $L(O)\langle\omega\rangle((v,v'))$ est modifiée dans la règle instanciée.
Certains arcs implicites des orbites considérées sont différents dans L et R, donc pour tout nœud v' de l'orbite d'instanciation O, les orbites $L(O)\langle\omega\rangle((v,v'))$ et $R(O)\langle\omega\rangle((v,v'))$ sont différentes. Il nous reste donc à montrer qu'il n'y a ni scission ni fusion.
Cela se fait de la même manière que dans le cas explicite, avec la partie de chemin issue des arcs explicites identiques dans $L(O)$ et $R(O)$, puisque les structures $\underline{L}\langle\omega\rangle(v)$ et $\underline{R}\langle\omega\rangle(v)$ sont identiques.

Donc, d'après la proposition 6, pour tout brin v' de O, $G\langle\omega\rangle(m((v,v')))$ est modifiée dans la transformation directe $G \xrightarrow{r,m} H$.

Réciproquement, soit un brin v' de O, supposons que $G\langle\omega\rangle(m((v,v')))$ est modifiée dans la transformation directe $G \xrightarrow{r,m} H$. D'après les propositions précédentes, l'orbite $L\langle\omega\rangle(v)$ ne peut être ni inchangée, ni scindée, ni fusionnée (et bien sûr ni créée, ni supprimée). Elle est donc nécessairement modifiée. □

Dans la règle de triangulation de face, l'orbite $L\langle 1,2,3\rangle(n0)$ est l'orbite qui filtre les sommets incidents aux brins $a0$, $c0$, $e0$ et $g0$. Dans R, cette orbite est modifiée par la créations d'arcs de dimension 1, 2 et 3. Dans la transformation directe (figures 3.44a et 3.44b), nous pouvons observer que ces sommets possèdent chacun une 2-liaison qui les connecte au sommet central.

Enfin, comme défini dans les sections de scission de fusion, une $\langle\omega\rangle$-orbite peut être simplement modifiée malgré la détection d'une scission ou d'une fusion parce que la règle ne possède aucune $\langle\omega\rangle$-orbite complète.

(a) G (b) H

Figure 3.44 – Modification de sommets par triangulation

4.7 Exemples de détections d'évènements

Dans cette section, nous présentons des extraits de détections d'évènements à partir de l'analyse syntaxique des règles Jerboa.

Triangulation d'une face

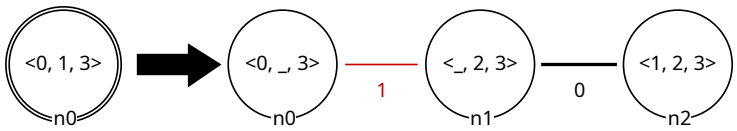


Figure 3.45 – Règle de triangulation

(a) G (b) H

Figure 3.46 – Triangulation d'une face

Type d'orbite	Évènements détectés
$\langle 0 \rangle$	Création $R\langle 0 \rangle(n1)$; Non-changement $R\langle 0 \rangle(n0)$
$\langle 1 \rangle$	Création $R\langle 0 \rangle(n2)$; Création $R\langle 1 \rangle(n2)$ Scission $L\langle 1 \rangle(n0)$
$\langle 2 \rangle$	Création $R\langle 2 \rangle(n1)$; Non-changement $R\langle 2 \rangle(n0)$
$\langle 1, 2 \rangle$	Modification$R\langle 1, 2 \rangle(n0)$; Création $R\langle 1, 2 \rangle(n2)$
$\langle 0, 1 \rangle$	Scission $L\langle 0, 1 \rangle(n0)$
$\langle 0, 2 \rangle$	Création $R\langle 0, 2 \rangle(n1)$ Non-changement $R\langle 0, 2 \rangle(n0)$
$\langle 0, 1, 2 \rangle$	Modification $R\langle 0, 1, 2 \rangle(n0)$

Table 3.1 – Extrait des évènements détectés dans la règle de triangulation d'une face

Insertion d'une arête

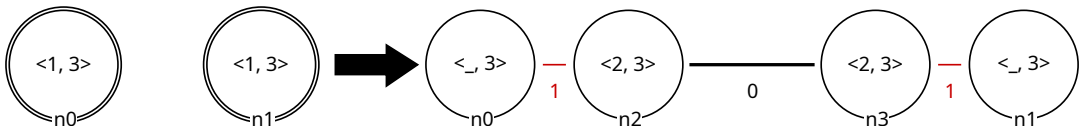
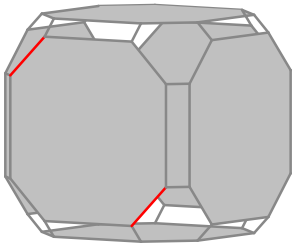
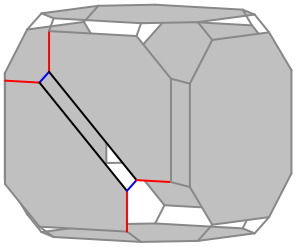


FIGURE 3.47 – Règle d'insertion d'une arête entre deux sommets



(a) G



(b) H

Type d'orbite	Évènements détectés
$\langle 0 \rangle$	Création $R\langle 0 \rangle(n2)$ Non-changement $R\langle 0 \rangle(n0)$ Non-changement $R\langle 0 \rangle(n1)$
$\langle 1 \rangle$	Scission $L\langle 1 \rangle(n0)$ Scission $L\langle 1 \rangle(n1)$
$\langle 2 \rangle$	Non-changement $R\langle 2 \rangle(n0)$ Non-changement $R\langle 2 \rangle(n1)$ Création $R\langle 2 \rangle(n2)$ Création $R\langle 2 \rangle(n3)$
$\langle 1, 2 \rangle$	Modification $R\langle 1, 2 \rangle(n0)$ Modification $R\langle 1, 2 \rangle(n0)$
$\langle 0, 1 \rangle$	Scission $L\langle 0, 1 \rangle(n0)$; Fusion $R\langle 0, 1 \rangle(n0)$
$\langle 0, 2 \rangle$	Création $R\langle 0, 2 \rangle(n2)$ Non-changement $R\langle 0, 2 \rangle(n0)$ Non-changement $R\langle 0, 2 \rangle(n1)$
$\langle 0, 1, 2 \rangle$	Fusion $R\langle 0, 1, 2 \rangle(n0)$

TABLE 3.2 – Extrait des évènements détectés dans la règle d'insertion d'une arête

Notons que dans le tableau 3.2, nous détectons que l'orbite $L\langle 0, 1 \rangle(n0)$ est scindée par une suppression des 1-arcs implicite, mais aussi que l'orbite $R\langle 0, 1 \rangle(n0)$ est explicitement fusionnée par un 0-arc. Ces deux cas sont mutuellement exclusifs. Dans le premier cas, les deux nœuds d'accroche filtrent deux sommets de la même face, tandis que dans l'autre, ils filtrent des sommets appartenant à deux faces distinctes.

5 Conclusion

Dans ce chapitre, nous exploitons les formalismes des cartes généralisées associés à celui des règles Jerboa afin de caractériser un ensemble d'évènements pouvant avoir lieu au cours de la transformation d'une carte généralisée. Nous avons étudié sept évènements (création, suppression, scission, fusion, non-changement, modification, sans effet) pouvant intervenir lors de la transformation d'un objet et avons présenté nos méthodes afin de les détecter. Ces détections sont essentielles, car dans les chapitres suivants, elles nous permettent d'enregistrer et de suivre avec précision les évolutions d'entités topologiques. Ce suivi est ainsi important pour proposer des appariements d'entités lors de la réévaluation d'un modèle.

Nous avons tout d'abord présenté ces évènements de manière générale au travers de transformations directes de cartes généralisées, c'est-à-dire avant et après leurs transformations. Nous avons ensuite proposé plusieurs méthodes pour formaliser chacun de ces évènements grâce à une analyse des règles de transformation de graphes. Ces analyses nous permettent de détecter des évènements de manière statique, c'est-à-dire sans avoir besoin de les appliquer. Enfin, nous avons étendu nos formalisations d'évènements aux règles Jerboa qui exploitent pleinement le potentiel des orbites dans les cartes généralisées. À l'aide de ce formalisme, nous pouvons déterminer l'ensemble des évènements appliqués sur un objet sans qu'il soit nécessaire de procéder à l'analyse globale de l'objet, sans ajout de code dans les opérations et sans avoir à appliquer les règles. Enfin, les règles Jerboa peuvent être conçues de diverses manières. Or, l'analyse d'une règle dépend de comment celle-ci est écrite. Ainsi, les cas limites de nos méthodes de détection sont définis. Cela nous permet de mettre en œuvre des vérifications locales dans l'objet, afin de déterminer si l'évènement détecté a bien eu lieu ou non.

CHAPITRE 4

Réévaluation à base de règles

Sommaire

1 Introduction

Dans le chapitre 2, nous avons examiné divers systèmes de modélisation. Nous avons étudié leurs méthodes pour réévaluer les spécifications paramétriques ainsi que les différentes solutions qu'ils proposent pour la nomination persistante des entités topologiques. Nous avons constaté que la plupart de ces systèmes sont adaptés uniquement à la conception et à la réévaluation de modèles paramétriques dans une dimension donnée, généralement en 3D. De plus, ils définissent des mécanismes de nommage hétérogènes selon la dimension des cellules.De ce fait, leurs méthodes ne sont définies que pour un certain ensemble d'entités topologiques. Ces méthodes sont ainsi difficiles à étendre vers d'autres dimensions et, par conséquent, ne sont pas généralisables.

Dans ce chapitre, nous nous appuyons sur les travaux de Cardot [Car19] et les concepts qu'elle propose pour mettre en œuvre un système de réévaluation générique grâce à l'utilisation des cartes généralisées et des règles Jerboa. En particulier, nous nous appuyons sur ses contributions pour l'implantation d'un système de nomination persistante en enrichissant la reconstitution des historiques d'évolution des orbites à l'aide du suivi des orbites et de leurs origines. Cette extension permet d'améliorer la procédure d'appariement des entités topologiques ainsi que de définir des stratégies accessibles aux utilisateurs pour la réévaluation.

Nous commençons par présenter deux cas d'études de réévaluation dans la section 2. Nous décrivons ensuite notre approche de nomination persistante, en abordant d'une part la nomination des brins à l'aide des règles Jerboa (section 3), et d'autre part, la nomination des orbites à travers la formalisation des évènements présentée précédemment au chapitre 3 (section 4). Puis, nous présentons, dans la section 5 notre mécanisme d'appariement des orbites lors du processus de réévaluation et proposons un système de stratégie pour paramétrer la réévaluation d'une spécification paramétrique. Enfin, dans la section 6, nous donnons un état des lieux de ce qui a été implanté avant de conclure le chapitre.

2 Cas d'études

Tout au long de ce chapitre, nous présentons nos différentes approches pour la réévaluation, notamment la nomination persistante des entités topologiques et leurs appariements. Nous nous appuyons, pour cela, sur deux exemples de constructions et leurs réévaluations, qui nous servent de cas d'études. Pour chacune de ces constructions, nous présentons les représentations géométriques (par étape de construction) accompagnées de leurs spécifications paramétriques.

2.1 Construction nº 1

La première construction, illustrée figure 4.1, est l'exemple principal utilisé pour ce chapitre. Celui-ci va nous permettre d'illustrer nos méthodes de nomination persistante et de réévaluation. Nous détaillons, ci-après, sa spécification paramétrique contenue dans le listing 4.1.

L'application de l'opération, `1-creationCarre`, crée une face carrée (figure 4.1a). L'opération, `2-extrusionFace`, est appliquée sur cette face, incidente au brin 6, et l'extrude en un cube (figure 4.1b). L'opération, `3-insertionSommet`, est appliquée sur l'arête incidente au brin 46 et y insère un sommet (figure 4.1c). L'opération, `4-extrusionFaceVolume`, est appliquée

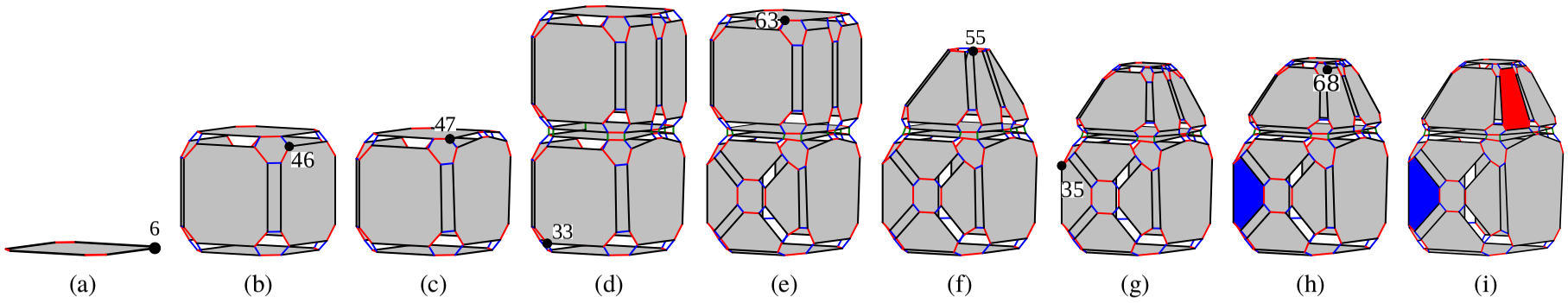


FIGURE 4.1 – Évaluation cas nº 1 : (a) `1-creationCarre(pos)` ; (b) `2-extrusionFace(PN`$_6$`,vec)` ; (c) `3-insertionSommet(PN`$_{46}$`)` ; (d) `4-extrusionFaceVolume(PN`$_{47}$`,vec)` ; (e) `5-triangulationFace(PN`$_{33}$`)` ; (f) `6-contractionFace(PN`$_{63}$`)` ; (g) `7-chanfreinSommet(PN`$_{55}$`)` ; (h) `8-colorationFace(PN`$_{35}$`,Bleu)` ; (i) `9-colorationFace(PN`$_{68}$`, Rouge)`

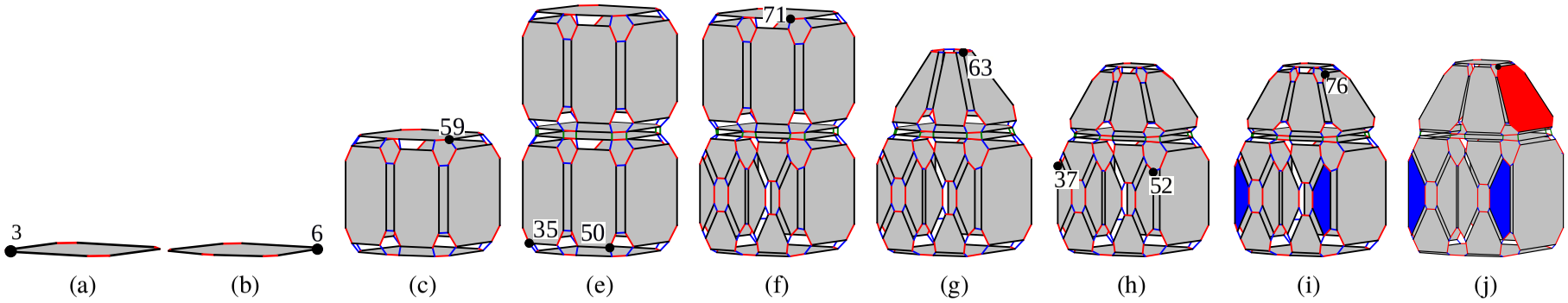


FIGURE 4.2 – Réevaluation cas nº 1 : (a) `1-carre(pos)` ; (b) `ADD 1-insertionSommet(3)` ; (c) `2-extrusionFace(PN`$_6$`,vec)` ; `DELETE 3-insertionSommet(PN`$_{46}$`)` ; (e) `4-extrusionFaceVolume(PN`$_{47}$`,vec)` ; (f) `5-triangulationFace(PN`$_{33}$`)` ; (g) `6-contractionFace(PN`$_{63}$`)` ; (h) `7-chanfreinSommet(PN`$_{55}$`)` ; (i) `8-colorationFace(PN`$_{35}$`, Bleu)` ; (j) `9-colorationFace(PN`$_{68}$`, Rouge)`

sur la face du dessus, incidente au brin 47, et l'extrude en un autre cube (figure 4.1d). L'opération, `5-trianguleFace`, est appliquée sur la face avant, incidente au brin 33, et la triangule (figure 4.1e). L'opération, `6-contractionFace`, est appliquée sur la face incidente au brin 63, et la contracte en un sommet (figure 4.1f). L'opération, `7-chanfreinSommet`, est appliquée sur ce même sommet, incident au brin 55, et fait un chanfrein dessus (figure 4.1g). L'opération, `8-colorationFace`, colore la face incidente au brin 35 (figure 4.1h). Enfin, la dernière opération, `9-colorationFace`, colore la face incidente au brin 68 (figure 4.1i).

```
1 1-creationCarre(pos)
2 2-extrusionFace(PN6=[1n6], vec)
3 3-insertionSommet(PN46=[1n5;2n5])
4 4-extrusionFaceVolume(PN47=[1n5;2n6;3n0], vec)
5 5-triangulationFace(PN33=[1n3;2n2])
6 6-contractionFace(PN63=[1n4;2n6;4n6])
7 7-chanfreinSommet(PN55=[1n5;2n6;3n0;4n4;6n2])
8 8-colorationFace(PN35=[1n3;2n4;5n0], Bleu)
9 9-colorationFace(PN68=[1n5;2n6;3n0;4n4;6n2;7n1], Rouge)
```

Listing 4.1 – Spécification paramétrique initiale du cas nº 1

Nous procédons ensuite à l'édition de cette spécification paramétrique, dont la représentation géométrique, à chaque étape, est illustrée dans la figure 4.2. Cette édition consiste à ajouter une opération d'insertion de sommet `9-insertionSommet(3)` (figure 4.2b) et à supprimer l'opération `3-insertionSommet` afin que celle-ci ne soit pas réévaluée.

```
1 1-carre(pos)
2 ADD 9-insertionSommet(3)
3 2-extrusionFace(PN6=[1n6], vec)
4 DELETE 3-insertionSommet(PN46=[1n5;2n4])
5 4-extrusionFaceVolume(PN47=[1n5;2n6;3n0], vec)
6 5-triangulationFace(PN33=[1n3;2n2])
7 6-contractionFace(PN63=[1n4;2n6;4n6])
8 7-chanfrein(PN55=[1n5;2n6;3n0;4n4;6n2])
9 8-coloration(PN35=[1n3;2n4;5n0], Bleu)
10 9-coloration(PN68=[1n5;2n6;3n0;4n4;6n2;7n1],Rouge)
```

Listing 4.2 – Spécification paramétrique éditée du cas nº 1

2.2 Construction nº 2

La seconde construction est illustrée figure 4.3. Celle-ci va nous permettre de préciser la définition de *stratégies de réévaluation*, notamment pour la suppression d'opérations. Nous détaillons, ci-après, sa spécification paramétrique contenue dans le listing 4.3.

La construction de cet objet est semblable à la construction nº 1. Le processus de modélisation commence avec l'application `1-creationCarre` qui crée une face. L'application suivante `2-extrusionFace` extrude la face en un cube. La troisième application `3-contractionFace` contracte la face supérieure du cube en un sommet. Ce sommet est ensuite chanfreiné par la quatrième application `4-chanfreinSommet`. L'application `5-triangulationFace` triangule la face frontale de l'objet. Enfin, les applications `6-colorationFace` et `7-colorationFace` colorient respectivement deux faces de l'objet en rouge et en bleu. Notons que la face coloriée en rouge est la face issue du chanfreinage (`4-chanfreinSommet`).

(a) (b) (c) (d) (e) (f) (g)

FIGURE 4.3 – Évaluation cas nº 2 : (a) `1-carre(pos)` ; (b) `2-extrusion(PN6,vec)` ; (c) `3-contractionFace(PN47)` ; (d) `4-chanfrein(PN40)` ; (e) `5-triangulationFace(PN33)` ; (f) `6-coloration(PN12,Rouge)` ; (g) `7-coloration(PN57,Bleu)`

(a) (b) (c) (e)

FIGURE 4.4 – Réévaluation cas nº 2 : (a) `1-carre(pos)` ; (b) `2-extrusion(PN6,vec)` ; (c) `3-contractionFace(PN47)` ; `DELETE 4-chanfrein(PN40)` ; (e) `5-triangulationFace(PN33)` ; `6-coloration(PN12,Rouge)` ; `7-coloration(PN57,Bleu)`

```
1-creationCarre()
2-extrusionFace(PN6=[1n6])
3-contractionFace(PN47=[1n5; 2n6])
4-chanfreinSommet(PN40=[1n4; 2n4; 3n2])
5-triangulationFace(PN33=[1n3; 2n2])
6-colorationFace(PN12=[1n4; 2n4; 3n2; 4n2], Rouge)
7-colorationFace(PN57=[1n4; 2n4; 3n2; 4n1; 5n2], Bleu)
```

Listing 4.3 – Spécification paramétrique initiale du cas nº 2

La version éditée de la spécification paramétrique consiste à supprimer la quatrième opération, comme l'illustre le listing 4.4. L'application `4-chanfreinSommet` ne doit donc pas être réévaluée. Les applications `6-colorationFace` et `7-colorationFace`, quant à elles, sont indirectement annulées, car les faces qu'elles doivent colorer ne sont pas recréées ainsi que l'atteste la figure 4.4. En effet, l'application `6-colorationFace` référence la face créée par `4-chanfreinSommet` qui n'existe plus dans la réévaluation. De la même manière, la face initialement coloriée par `7-colorationFace` n'est plus créée lors de la triangulation et ne peut donc pas être coloriée dans la réévaluation.

```
1-creationCarre()
2-extrusionFace(PN6=[1n6])
3-contractionFace(PN47=[1n5; 2n6])
DELETE 4-chanfreinSommet(PN40=[1n4; 2n4; 3n2])
5-triangulationFace(PN33=[1n3; 2n2])
6-colorationFace(PN12=[1n4; 2n4; 3n2; 4n2], Rouge)
7-colorationFace(PN57=[1n4; 2n4; 3n2; 4n1; 5n2], Bleu)
```

Listing 4.4 – Spécification paramétrique éditée du cas nº 2

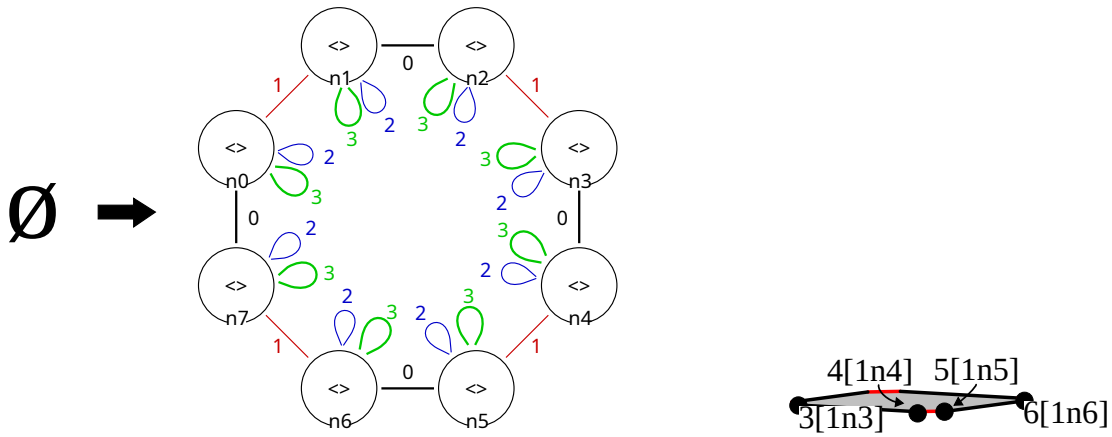
3 Nomination persistante des brins

L'étude des travaux antérieurs a révélé que la nécessité de créer et d'attribuer des noms persistants ne concerne que les entités topologiques spécifiquement référencées dans une spécification paramétrique, c'est-à-dire les paramètres topologiques associés aux opérations. Les travaux de Cardot [Car19] ont montré que, dans un système basé sur des règles, chaque brin peut être identifié de manière unique par un historique de construction, qui retrace l'ensemble des étapes ayant conduit à sa création. Ces travaux utilisent *un ensemble de brins* pour représenter de manière exhaustive les transformations appliquées sur une orbite. Toutefois, dans notre approche, nous estimons qu'utiliser plusieurs brins pour un nom persistant n'est pas nécessaire pour représenter la totalité des changements impactant une orbite. En effet, si une orbite est modifiée par plusieurs règles, alors elle est identifiée par autant de noms persistants. Nous proposons donc, dans un premier temps, de désigner une orbite à l'aide de l'historique d'un *unique* brin, ce brin étant celui sélectionné explicitement lors de la construction pour désigner l'orbite à transformer. Dans un second temps, nous procédons à la reconstitution de l'historique des évolutions de l'orbite désignée pour l'identifier formellement par son historique de construction.

Commençons par la construction des noms persistants des brins sélectionnés lors de la construction de l'objet nº 1 (figure 4.1 et listing 4.1)

Remarque : *Les figures suivantes (figures 4.5 à 4.13) représentent (a) des règles et (b) les résultat de leurs applications sur les instances correspondantes de la spécification paramétrique.*

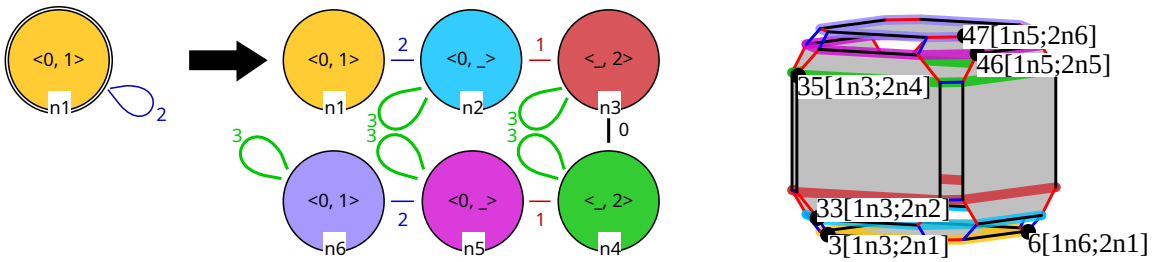
Afin de rendre le résultat plus lisible, seuls quelques brins sont explicitement décrits avec leurs numéros et, entre crochets, leurs historiques.



(a) (b)

FIGURE 4.5 – `1-creationCarre()`

L'application `1-creationCarre()` (figure 4.5) de la règle figure 4.5a crée *ex nihilo* une face carrée (figure 4.5b). Plus précisément, chacun des huit nœuds de cette règle crée un unique brin. Par exemple, les nœuds n_3, n_4, n_5 et n_6, dont les étiquettes sont vides, créent respectivement les brins 3, 4, 5 et 6. Rappelons que nous représentons l'historique de chaque brin avec l'ensemble des numéros d'application et les nœuds qui les créent ou filtrent. Ainsi, l'historique du brin 3 (respectivement des brins 4, 5 et 6) est $[1n_3]$ (respectivement $[1n_4]$, $[1n_5]$, $[1n_6]$).

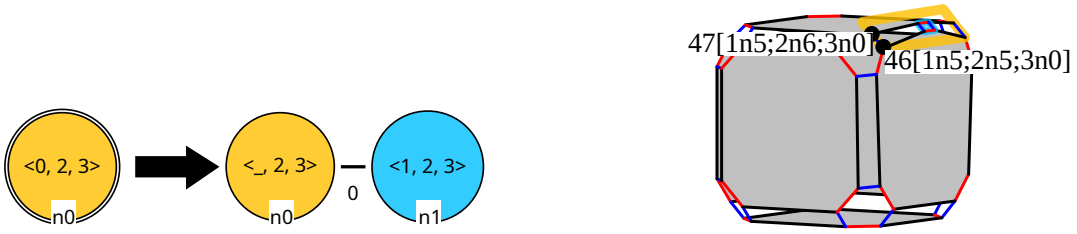


(a) (b)

FIGURE 4.6 – `2-extrusionFace(PN`$_6$`=[1n6])`

L'application `2-extrusionFace(PN`$_6$`=[1n6])` (figure 4.6) est ensuite appliquée sur le brin 6. Puisque le nœud d'accroche de cette règle (figure 4.6a) est de type `face` $\langle 0, 1\rangle$, cela revient à utiliser la face incidente à ce brin comme paramètre de d'application. Toutefois, le numéro d'un brin n'est pas une information robuste et son historique doit évoluer au fur et à mesure de la construction. Pour cette raison, au moment où une règle est appliquée, nous enregistrons l'historique du brin, plutôt que son simple numéro, en tant que nom persistant dans la spécification paramétrique. Cela équivaut, en quelque sorte, à capturer une « photographie » de l'historique des brins désignés, que l'on enregistre ensuite dans la spécification paramétrique. Ainsi, si les numéros de brins venaient à être différents au moment de la réévaluation de la deuxième application, le brin sélectionné ne pourrait être différent de celui créé par le nœud n_6 de `1-creationCarre()`.

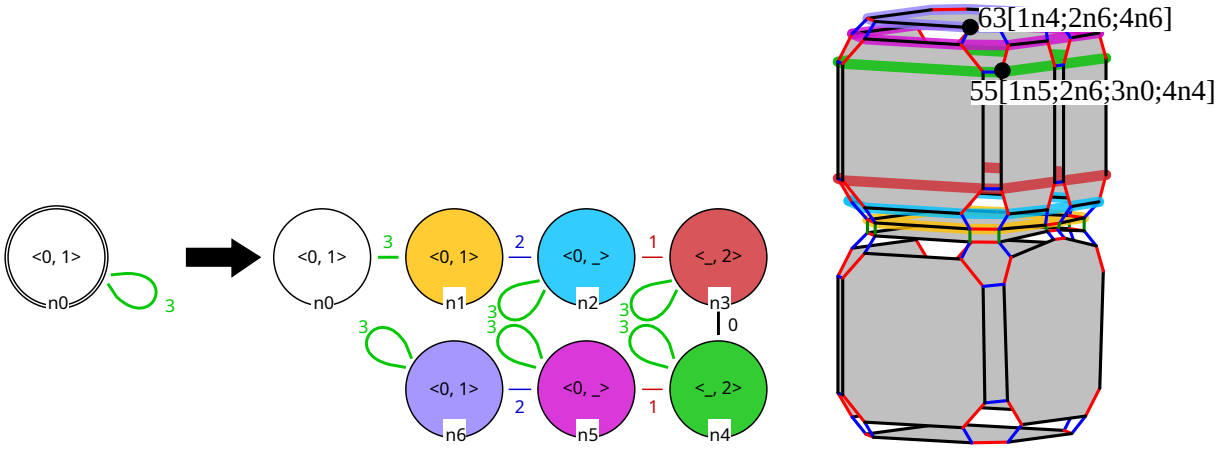
Au fil du processus de modélisation, l'historique d'un brin évolue à chaque fois que celui-ci est filtré par une règle. Si nous observons le cube issu de l'extrusion, nous constatons que les brins filtrés par la règle sont préservés, parce que le nœud qui les filtre (n_1) est lui-même préservé dans la règle (pour rappel, un nœud dans la partie gauche d'une règle est préservé si on le retrouve dans la partie droite). Pour les brins préservés par n_1, leurs historiques sont mis à jour par l'ajout d'un couple `2n1`. Ainsi, l'historique du brin 3 (respectivement des brins 4, 5 et 6) devient `[1n3;2n1]` (respectivement `[1n4;2n1]`, `[1n5;2n1]`, `[1n6;2n1]`). Pour les autres brins, ceux qui sont créés, rappelons que ce sont des copies des brins filtrés. Ainsi, en tant que copies, ces brins portent dans leurs historiques respectifs celui du brin à partir duquel ils sont copiés. Par exemple, les brins 33 et 35, créés par les nœuds n_1 et n_4 respectivement, sont des copies du brin 3. Leurs historiques sont `[1n3;2n1]` et `[1n3;2n4]` respectivement. De la même manière, les brins 46 et 47, créés par les nœuds n_5 et n_6, sont des copies du brin 5. Leurs historiques sont `[1n5;2n5]` et `[1n5;2n6]` respectivement (figure 4.6b).



(a) (b)

Figure 4.7 – `3-insertionSommet(PN`$_{46}$`=[1n5;2n5])`

L'application `3-insertionSommet(PN`$_{46}$`=[1n5;2n4])` (figure 4.7) est appliquée sur l'arête désignée par le brin 46. Comme pour l'application précédente, le nom persistant enregistré est une photographie de l'historique du brin 46, c'est-à-dire `[1n5;2n5]`. Suite à l'application, les historiques des brins 46 et 47, tous deux filtrés par le nœud n_0 (figure 4.7a), sont mis à jour en `[1n5;2n5;3n0]` et `[1n5;2n6;3n0]` respectivement (figure 4.7b).

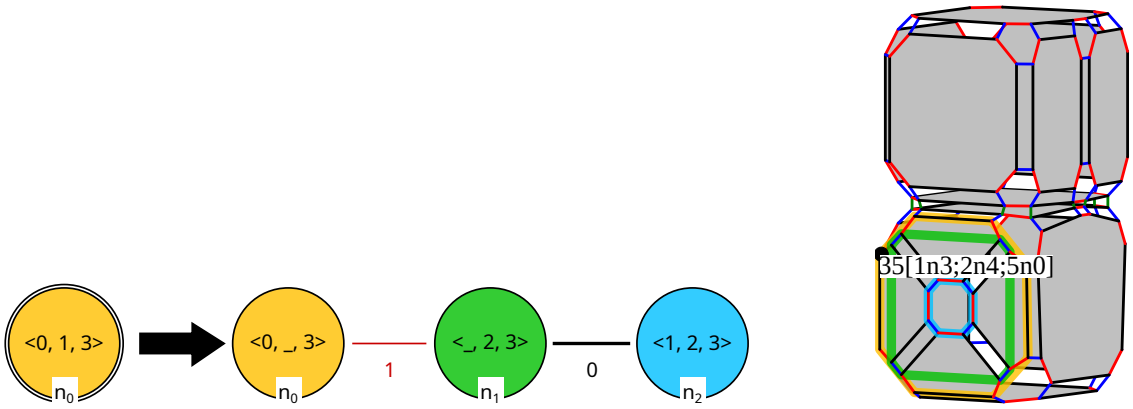


(a) (b)

Figure 4.8 – `4-extrusionFaceVolume(PN`$_{47}$`=[1n5;2n6;3n0])`

L'application `4-extrusionFaceVolume(PN`$_{47}$`=[1n5;2n6;3n0])` crée un nouveau cube en extrudant la face supérieure de l'objet (figure 4.8). Cette face est désignée par le brin 47 et le

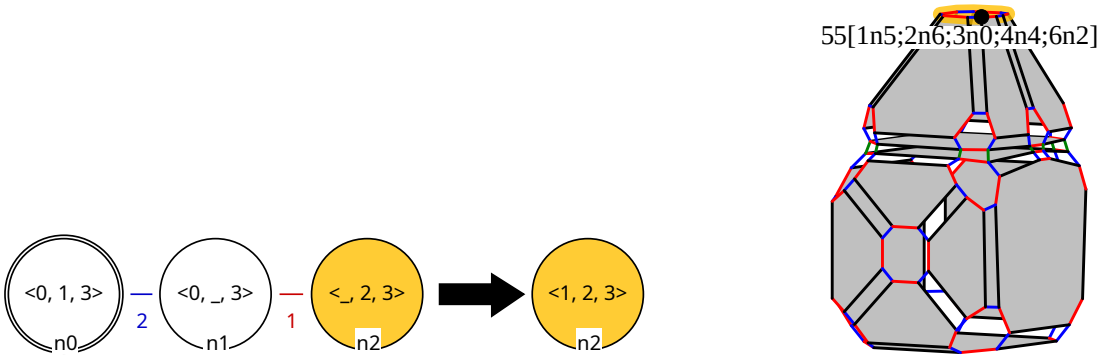
nom persistant enregistré est donc `[1n5;2n6;3n0]`. Les brins 55 et 63 sont créés par les nœuds n_4 et n_6 respectivement (figure 4.8b) et leurs historiques sont `[1n5;2n6;3n0;4n4]` et `[1n4;2n6;4n6]` respectivement. Notons que la création du brin 63 est indépendante de la troisième application. Elle n'apparaît donc pas dans son historique.



(a) (b)

FIGURE 4.9 – `5-triangulationFace(PN33=[1n3;2n2])`

L'application `5-triangulationFace(PN33=[1n3;2n2])` (figure 4.9) triangule la face désignée par le brin 33 et le nom persistant `[1n3;2n2]` est enregistré dans la spécification paramétrique. Les brins 33 et 35 sont filtrés par le nœud n_0 (figure 4.9a), leurs historiques évoluent donc en `[1n3;2n2;5n0]` et `[1n3;2n4;5n0]` respectivement (figure 4.9b).

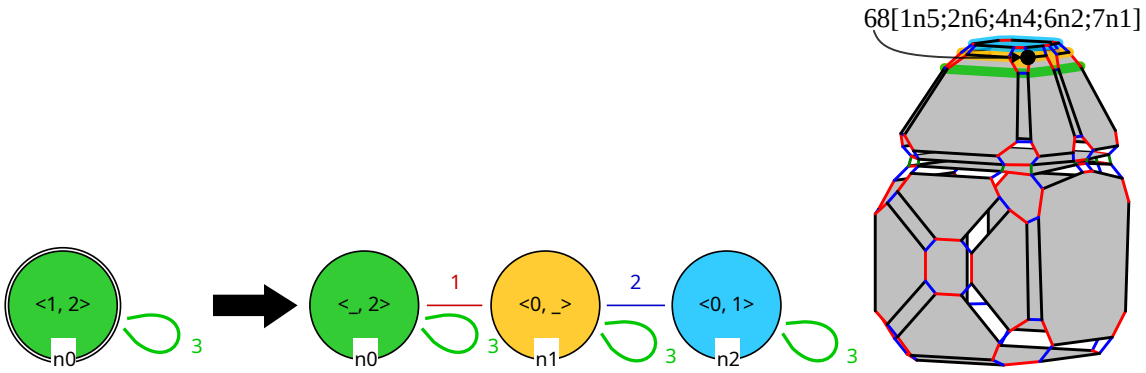


(a) (b)

FIGURE 4.10 – `6-contractionFace(PN63=[1n4;2n6;4n6])`

L'application `6-contractionFace(PN63=[1n4;2n6;4n6])` (figure 4.10) contracte la face supérieure du cube du haut désignée par le brin 63 en un sommet (figure 4.10b). Le nom persistant enregistré dans la spécification paramétrique est `[1n4;2n6;4n6]`. Puisque toute la face est filtrée par le nœud n_0, lequel est supprimé dans la règle (figure 4.10a), le brin 63 est lui-même supprimé. Les historiques des brins filtrés par $n2$, qui composent maintenant le sommet issu de la contraction, sont mis à jour. En particulier, l'historique du brin 55 devient `[1n5;2n6;3n0;4n4;6n2]` (figure 4.10b).

L'application `7-chanfreinSommet(PN55=[1n5;2n6;3n0;4n4;6n2])` (figure 4.11) est appliquée sur le sommet issu de l'application précédente, désigné par le brin 55, lui-même filtré par le nœud $n0$ (figure 4.11a). Comme dans les étapes précédentes, l'historique `[1n5;2n6;3n0;`



(a) (b)

FIGURE 4.11 – 7-chanfreinSommet(PN$_{55}$=[1n5;2n6;3n0;4n4;6n2])

4n4;6n2], avant sa mise à jour par la règle, est enregistré dans la spécification paramétrique. L'application crée une face à partir du chanfreinage du sommet désigné. Parmi les brins créés, nous trouvons notamment le brin 68 dont l'historique est [1n5;2n6;3n0;4n4;6n2;7n1] (figure 4.11b) .

(a) (b)

FIGURE 4.12 – 8-colorationFace(PN$_{35}$=[1n3;2n4;5n0])

(a) (b)

FIGURE 4.13 – 9-colorationFace(PN$_{68}$=[1n5;2n6;3n0;4n4;6n2;7n1])

Enfin, les deux dernières applications enregistrées, 8-colorationFace(PN$_{35}$=[1n3;2n4;5n0]) et 9-colorationFace(PN$_{68}$=[1n5;2n6;3n0;4n4;6n2;7n1]) (figures 4.12 et 4.13), colorient les faces désignées par les brins 35 et 68 (figures 4.12b et 4.13b), filtrés par l'unique

nœud n_0 de chacune des deux règles de coloration (figures 4.12a et 4.13a, seul le plongement de couleur change). L'historique du brin 35 (respectivement 68) est enregistré dans la spécification paramétrique comme nom persistant de la huitième application (respectivement la neuvième), puis mis à jour en `[1n3;2n4;5n0;8n0]` (respectivement `[1n5;2n6;3n0;4n4;6n2;7n1;9n0]`).

Le processus de création de chacun des noms persistants enregistrés dans la spécification paramétrique du cas d'étude nº 1, que nous venons de détailler, nous permet de mettre deux propriétés en évidence. La première est que la nomination des brins est, par construction, unique. Un tel historique construit pas à pas, *via* les règles appliquées et de manière explicite, un chemin qui relie un brin et son ancêtre le plus ancien (créé *ex nihilo*). La seconde propriété est que la nomination persistante via l'historique des brins est, à elle-seule, insuffisante, car l'historique d'évolution d'un brin ne permet pas de caractériser l'historique des évènements qui ont impacté une orbite. De manière plus évidente, en omettant le cas où un nœud d'accroche désigne explicitement un brin, un brin n'est pas une orbite. Cette nomination, seule, n'est donc pas suffisante, car non représentative des transformations subies par une orbite. Dans notre approche, ces deux propriétés renforcent l'idée que représenter l'historique des évolutions d'une orbite est nécessaire pour permettre son appariement lors de la réévaluation.

4 Nomination persistante des orbites

Pour compléter la nomination des entités topologiques, nous construisons, pour chaque nom persistant de la spécification paramétrique, un arbre d'évaluation, ou *DAG d'évaluation*. Nous allons montrer que l'utilisation combinée de ces noms persistants avec la détection automatique des évènements, obtenus par l'analyse syntaxique des règles (chapitre 3), permet de reconstruire les historiques d'évolution des orbites, garantissant ainsi les bases d'une nomination persistante.

4.1 Reconstitution de l'historique des évolutions d'une orbite

Pour expliciter les mécanismes mis en œuvre, nous détaillons la construction du DAG d'évaluation pour le paramètre topologique de l'application `8-colorationFace(PN`$_{35}$`=[1n3;2n4;5n0], Bleu)`, dans la construction nº 1. La première étape consiste à identifier le type d'orbite

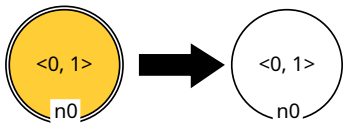


FIGURE 4.14 – Entrée de la règle de coloration de face

filtré par la règle. Comme expliqué dans les chapitres précédents, cette information est représentée dans l'étiquette du nœud d'accroche de la règle. Pour notre exemple, le type d'orbite qui nous intéresse, $\langle 0, 1 \rangle$ en l'occurrence, est déterminée par l'étiquette du nœud n_0 affichée en **jaune** dans la figure 4.14. À partir de cette information, nous pouvons commencer à remonter l'historique des évolutions de l'orbite sur laquelle l'opération de coloration a été appliquée. Autrement dit, l'historique que nous reconstituons prend la forme d'un DAG qui se construit de

manière rétrograde, partant de l'état le plus récent de l'orbite et remontant jusqu'à ses ancêtres les plus anciens. Ce type d'orbite est qualifié de *type d'orbite de suivi*. Il représente le premier élément que nous intégrons dans le DAG en tant que racine (figure 4.15).

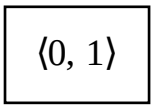


Figure 4.15 – Racine du DAG d'évaluation

La suite de la reconstitution consiste à remonter l'historique du nom persistant du brin, application par application. Dans le DAG d'évaluation, une application est représentée par un couple de niveaux : (1) un *niveau d'orbite* qui contient l'ensemble des types d'orbites suivis dans l'application courante ; (2) un *niveau d'événements* qui liste les évènements qui ont impacté les orbites suivies. Reprenons notre exemple. La deuxième étape de la reconstitution d'historique

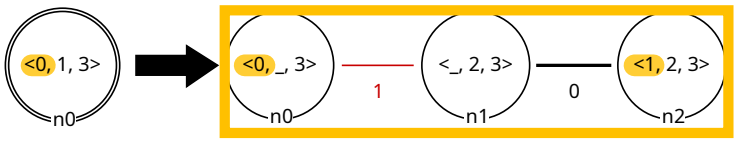


Figure 4.16 – Analyse de la règle de triangulation

commence avec la dernière application enregistrée dans le nom persistant du brin 35, c'est-à-dire l'application `5-triangulationFace` et son nœud n_0. Grâce à ces informations, nous pouvons procéder à l'analyse de la règle de triangulation (figure 4.16) et plus particulièrement à la face incidente à $n0$, $\langle 0,1\rangle(n_0)$. L'étude de cette orbite, comme nous l'avons vu dans la section 3.2, montre que l'orbite $L\langle 0,1\rangle(n_0)$ est scindée du fait de la réécriture de tous les 1-arcs implicites dans des dimensions hors du type $\langle 0,1\rangle$, voire de leur suppression. De plus, nous pouvons voir dans l'orbite $R\langle 0,1\rangle(n_0)$ que les 0 et 1-arcs implicites, à droite en **jaune**, sont des réécritures d'un 0-arc implicite dans le nœud d'accroche, à gauche aussi en **jaune**. Ainsi, l'orbite que nous

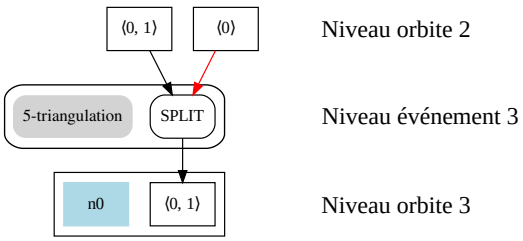


Figure 4.17 – Étape 2 `[1n3;2n4;5n0]` de la construction du DAG

suivons, d'une part, est scindée, et, d'autre part, a pour origine une arête de type $\langle 0\rangle$. Ces informations, acquises lors de l'analyse statique de la règle, sont reportées dans le DAG d'évaluation de la manière suivante :

1. la racine du DAG est intégrée dans un niveau d'orbite qui est annoté par le nœud n_0 issu du nom persistant ;
2. un niveau d'événement, annoté par le numéro d'application et le nom de la règle, est ensuite créé et contient un nœud d'événement correspondant à la scission détectée pour l'orbite $\langle 0,1\rangle(n_0)$;

3. une flèche **noire** est créée entre le nœud d'événement [SPLIT] et le nœud d'orbite $\boxed{\langle 0,1\rangle}$ pour marquer une *relation de suivi* ;
4. les nœuds d'orbites de suivi $\boxed{\langle 0,1\rangle}$ et d'origine $\boxed{\langle 0\rangle}$ sont ajoutés au DAG, au-dessus du niveau de l'événement ;
5. enfin, le nœud d'événement [SPLIT] est relié au nœud d'orbite $\boxed{\langle 0,1\rangle}$ par une relation de suivi et au nœud d'orbite $\boxed{\langle 0\rangle}$ par une *relation d'origine* (représentée par une flèche **rouge**).

Au terme de cette étape, le DAG (figure 4.17) se lit de bas en haut de la manière suivante : l'orbite face $\langle 0,1\rangle$ provient de la scission, par triangulation, d'une autre face $\langle 0,1\rangle$ et a pour origine une arête $\langle 0\rangle$. Notons que les nœuds d'orbites ne sont, à ce stade, pas encore intégrés dans un niveau d'orbite car ils sont issus d'une application antérieure à l'application courante dans l'historique.

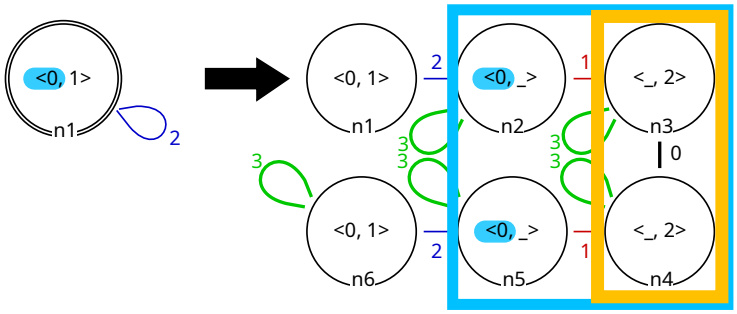


FIGURE 4.18 – Analyse de la règle d'extrusion d'une face

À l'étape suivante, nous remontons l'historique du nom persistant du brin 35 d'un niveau. Cette fois, l'application concerne la règle d'extrusion d'une face et son nœud n_4. Comme pour la triangulation, nous procédons à l'analyse de la règle à la différence près que cette fois-ci, nous suivons deux types d'orbites : une face $\langle 0,1\rangle$ (l'orbite encadrée en **bleu**) et une arête $\langle 0\rangle$ (l'orbite encadrée en **jaune**), toutes deux incidentes au nœud n_4 (figure 4.18). L'analyse révèle que la face et l'arête sont toutes les deux créées, car les orbites sont constituées uniquement de nœuds créées. La face a pour origine une arête ($\langle 0\rangle$), car les 0-arcs implicites (surlignés en **bleu**) sont réécrits à partir d'un 0-arc dans le nœud d'accroche. L'arête à pour origine un brin ($\langle\rangle$) du fait de l'absence d'arc implicite réécrit en 0 dans son orbite.

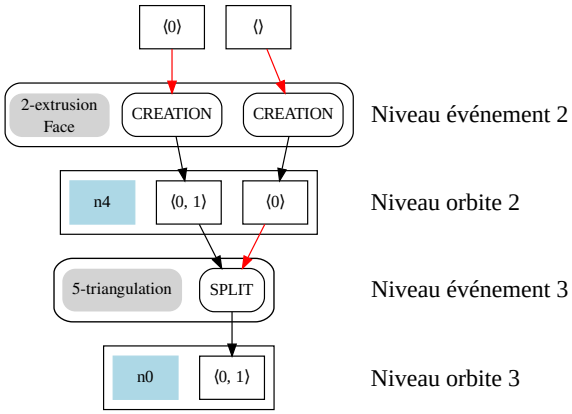


FIGURE 4.19 – Étape 3 `[1n3;2n4;5n0]` de la construction du DAG

Encore une fois, les informations issues de l'analyse de la règle sont reportées dans le DAG :

1. nous créons un niveau d'orbite, annoté par le nœud n_4, qui contient les nœuds d'orbites de suivi, calculés à l'étape précédente ;
2. nous créons un niveau d'événement, annoté par le numéro d'application et le nom de la règle, qui contient deux nœuds d'événements `CREATION` (un pour chaque nœud d'orbite) ;
3. les nœuds d'origines sont ajoutés au DAG ;
4. chacun des deux nœuds d'événement est relié par une relation de suivi aux nœuds d'orbites de suivis ;
5. enfin, chacun des nœuds d'orbite d'origine est relié au nœud d'événement qui convient, c'est-à-dire, l'orbite origine $\langle 0 \rangle$ (respectivement $\langle \rangle$) est connectée à un nœud d'évènement `CREATION` qui est lui-même connecté à $\langle 0, 1 \rangle$ (respectivement $\langle 0 \rangle$).

Ce niveau du DAG (figure 4.19) se lit ainsi : l'orbite face incidente à n_4 est créée par l'extrusion d'une arête et l'arête incidente à n_4 est créée par l'extrusion d'un brin.

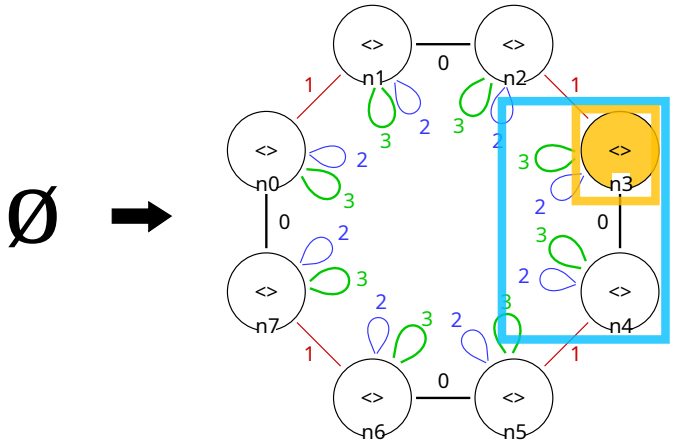


Figure 4.20 – Analyse de la règle d'extrusion d'une face

Nous remontons encore l'historique d'un cran et arrivons à la dernière étape de cette reconstitution de l'historique des évolutions de notre orbite de suivi. L'application désignée est la toute première enregistrée dans la spécification, c'est-à-dire la règle de triangulation et son nœud n_3. L'analyse de la règle révèle que l'arête (encadrée en **bleu**) et le brin (encadré en **jaune**) sont créés *ex nihilo* par la règle, car son membre gauche est vide. Comme pour les étapes précédentes, nous créons les niveaux d'orbite et d'événement qui correspondent à l'application courante et les remplissons avec les nœuds d'orbites issus de l'étape précédente et les événements que nous avons détectés.

Dans cet exemple, nous avons illustré la reconstitution de l'historique des évolutions d'une orbite à l'aide de relations de suivi et d'origine. Ces relations nous permettent de représenter des orbites par leur historique d'évolution complet et ainsi de les distinguer les unes des autres lorsqu'elles partagent le même processus de construction.

4.2 Intégration d'un chemin d'origine

Reprenons la notion d'orbite d'origine d'après la définition 30 (page 75). Du fait de la propriété d'isomorphisme des nœuds dans les règles Jerboa, un type d'origine est calculé uniquement à partir de la réécriture des arcs implicites d'un nœud d'accroche et d'un type d'orbite $\langle \omega \rangle$.

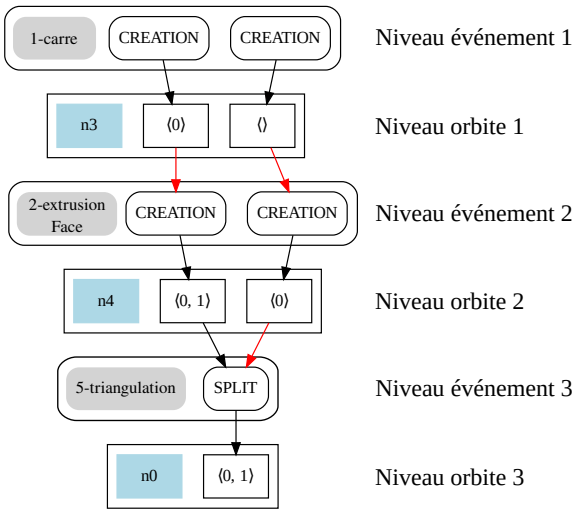


FIGURE 4.21 – Étape 4 `[1n3;2n4;5n0]` de la construction du DAG

Malgré le fait que le type d'une telle orbite soit calculé à partir d'un nœud d'accroche, celui-ci ne filtre pas toujours l'orbite d'origine, elle peut simplement se trouver dans sa composante connexe.

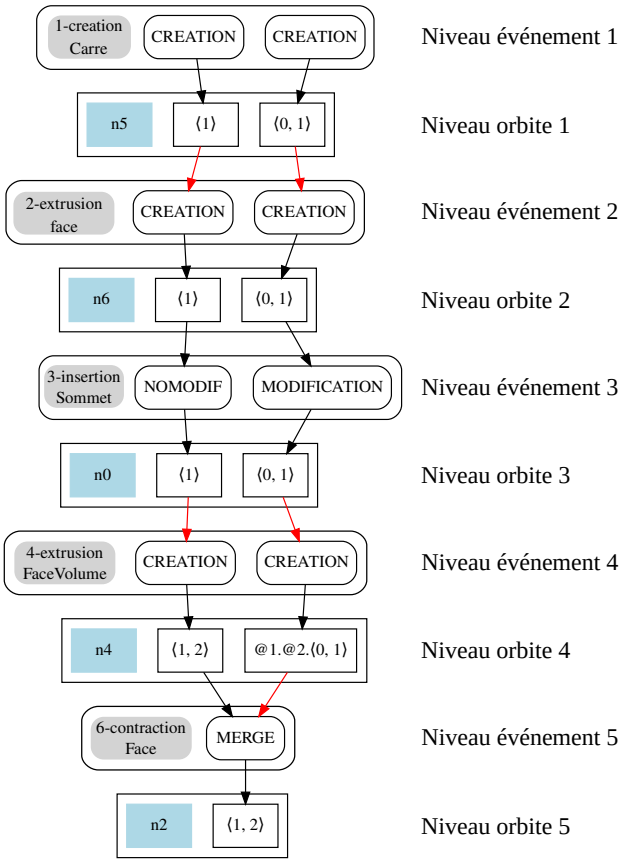


FIGURE 4.22 – DAG d'évaluation du PN_{55}

Comme nous venons de le voir, un DAG est construit à partir de l'historique d'un brin. Ainsi, chaque niveau d'orbite contient un nœud de règle dont l'association à un nœud d'orbite désigne une orbite dans une règle Jerboa. Or, dans le processus de construction, l'orbite associée à ce

nœud est elle-même filtrée par le nœud d'accroche de la règle suivante. Par conséquent, l'association entre un nœud de règle et un nœud d'origine peut désigner une orbite qui n'est pas une orbite d'origine. Pour cette raison, nous intégrons dans les nœuds d'orbite une information complémentaire sous la forme d'un chemin. Ce chemin permet, dans la règle, de remonter jusqu'au nœud de l'orbite d'origine à partir du nœud enregistré dans le DAG.

Dans l'exemple que nous venons de traiter, les orbites d'origines sont systématiquement filtrées par le nœud d'accroche. En effet, dans l'extrusion (respectivement la triangulation), l'arête (respectivement la face) est créée à partir d'un brin (respectivement d'une arête) filtré(e) par le nœud d'accroche. Ainsi, puisque l'origine est dans le nœud d'accroche, ces chemins sont nuls.

Considérons le DAG d'évaluation du PN_{55} (figure 4.22) pour illustrer le calcul et l'utilisation d'un chemin pour compléter les orbites d'origines. Ce DAG décrit l'historique des évolutions d'un sommet issu d'une fusion provoquée par l'application `6-contractionFace`. L'analyse de

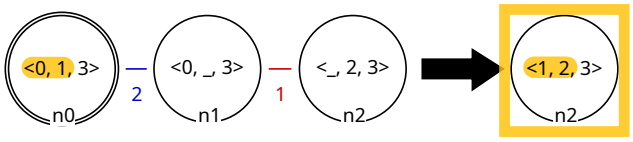


Figure 4.23 – Analyse de la règle de contraction

la règle de contraction nous apprend que l'origine de cette fusion est une face, comme nous pouvons aussi le voir dans le DAG, mais que cette face est filtrée par le nœud d'accroche n_0 (figure 4.23). Or, le sommet $R\langle 1,2\rangle(n_2)$ n'est créé ni par copie ni par la réécriture des arcs de la face incidente à n_0. Au contraire, la suppression de cette face provoque la fusion de ses sommets incidents en un seul sommet qui est celui désigné par le DAG. Ainsi, la face supprimée est à l'origine de la fusion, mais elle ne permet pas, seule, de représenter les sommets fusionnés qui sont filtrés par l'orbite $L\langle 1,2\rangle(n_2)$. Nous devons alors compléter la notion d'origine à l'aide d'un contexte qui permet d'atteindre le nœud d'accroche de la règle. Ce contexte prend la forme

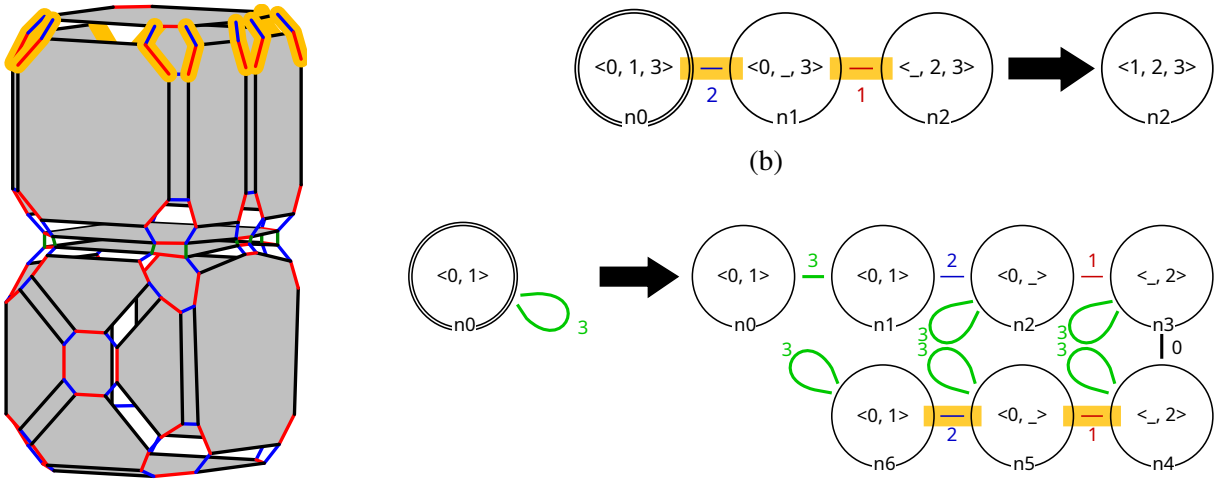


Figure 4.24 – Chemin @1.@2 dans l'objet (a) ; règle de contraction (b) ; la règle d'extrusion (c)

d'un chemin entre les nœuds. Pour calculer un tel chemin, nous effectuons un parcours depuis le nœud enregistré par l'application jusqu'au nœud d'accroche (voir le chemin en **jaune** dans

la figure 4.24b et le filtrage associé dans l'objet figure 4.24a). Par exemple, dans la règle de contraction, pour n_2 comme nœud de départ, ce chemin est constitué des arcs explicites 1 et 2. Dans le DAG, il est ajouté au nœud de l'orbite face qui devient $\boxed{@1.@2\langle 0,1\rangle}$.

En effet, le DAG est construit à partir du nom persistant du brin 55 (PN_{55}=[1n5 ;2n6 ;3n0 ; 4n4 ;6n2]). Comme vu précédemment, les nœuds apparaissant dans le DAG sont bien ceux du nom persistant. Un simple calcul d'origine sans prise en compte du chemin indiquerait que le sommet désigné par $\langle 1,2\rangle(n_2)$ aurait pour origine une face directement incidente au nœud n_4, nœud provenant du membre droit de la règle d'extrusion. Or, en analysant cette règle, nous voyons que la face incidente à n_4 correspond à une face latérale et non à la face supérieure du cube créé par extrusion. En revanche, nous pouvons observer que l'utilisation du chemin depuis n_4 (le parcours des arcs explicites 1 et 2 à partir du nœud n_4) permet bien de désigner le nœud n_6, et donc la face supérieure de ce cube (figure 4.24c). L'annotation du nœud avec un chemin nous permet donc de remonter jusqu'au nœud désignant la face à l'origine de la fusion de ses sommets incidents lors de l'application suivante.

4.3 Résumé

La mise en œuvre de cette procédure, pour reconstituer l'historique des évolutions d'une orbite, ne nécessite qu'un nom persistant de brin et les règles enregistrées dans une spécification paramétrique. Comme illustré dans l'exemple, combiner l'analyse statique des règles aux noms persistants des brins enregistrés dans la spécification paramétrique nous permet de calculer et représenter les évolutions d'une orbite jusqu'à ses origines. Avec cette approche, il n'est alors pas nécessaire de réappliquer les règles pour déterminer les évènements au fur et à mesure de l'évolution du modèle. Ce mécanisme est exécuté uniquement pour les paramètres topologiques, c'est-à-dire les entités topologiques enregistrées dans la spécification paramétrique.Il combine ainsi les avantages de l'analyse statique des règles, qui permet une détection des événements à faible coût en calcul, avec la nomination d'un échantillon d'orbites restreint à celles explicitement désignées par l'utilisateur lors du processus de construction d'un objet. Cela réduit considérablement les calculs à effectuer

5 Réévaluation

Désormais, lorsque la spécification paramétrique est éditée puis réévaluée, les DAG d'évaluation peuvent être utilisés pour réaliser un appariement. Pour cela, à partir des DAG d'évaluation et au fur et à mesure de la réévaluation, nous construisons des *DAG de réévaluation* qui intègrent les modifications apportées à la spécification paramétrique. Comme nous allons le voir, l'enregistrement dans les DAG d'évaluation du suivi des entités mais aussi de leurs origines permet, d'une part, un appariement robuste et autorise, d'autre part, la mise en œuvre de stratégies de réévaluation configurables par l'utilisateur.

5.1 Appariement des orbites

La construction d'un DAG de réévaluation consiste à utiliser, étape par étape, le DAG d'évaluation initial et à intégrer les modifications apportées dans la spécification paramétrique éditée.

Chaque application est associée à un *type* spécifique : `INIT` si elle provient de l'évaluation **initiale**, ou bien `ADD`, `DELETE`, ou `MOVE`, selon qu'elle est respectivement **ajoutée**, **supprimée**, ou **déplacée** dans la spécification paramétrique éditée. Enfin, et à l'inverse de la construction d'un DAG d'évaluation, un DAG de réévaluation est construit dans l'ordre d'application des règles.

L'objectif de l'appariement est d'identifier dans un modèle réévalué les orbites utilisées comme paramètres dans une spécification paramétrique lors de l'évaluation initiale. Cette procédure a lieu en cours de réévaluation en utilisant directement les brins de l'instance courante du modèle et non les nœuds de règles Jerboa. Ceux-ci peuvent en effet filtrer plusieurs instances de brins et conduire à des erreurs d'appariement. Les DAG de réévaluation enregistrent donc directement des brins. Ainsi, la méthode consiste, pour chaque application d'un type donné, à :

`INIT` **:** mettre à jour les évènements initialement détectés lorsque nécessaire ;

`ADD` **:** intégrer des niveaux d'applications supplémentaires ;

`DELETE` **:** gérer l'annulation des évènements initialement détectés, voire interrompre l'appariement ;

`MOVE` **:** gérer les conséquences d'un déplacement, c'est-à-dire si l'opération déplacée ou les suivantes sont toujours applicables.

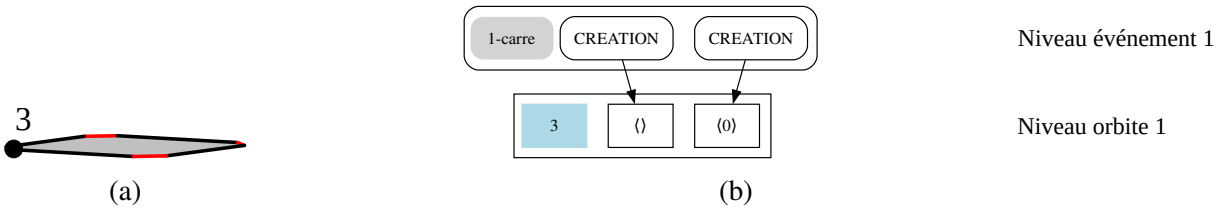


Figure 4.25 – Appariement des orbites issues de `INIT 1-creationCarre()`

Le nom persistant PN_{35} du cas d'étude et son DAG d'évaluation (figure 4.21), sert de point de départ pour illustrer la création d'un DAG de réévaluation et les différentes étapes d'appariement des paramètres topologiques. Pour rappel, la spécification paramétrique éditée était présentée dans le listing 4.2 et le résultat de la réévaluation était illustré sur la figure 4.2.

La première étape d'appariement concerne l'application `INIT 1-creationCarre` illustrée figure 4.25. Pour commencer, nous vérifions que la règle est toujours applicable. Puisque `1-creationCarre` ne référence aucun paramètre topologique, elle est réappliquée immédiatement.

Nous pouvons alors reproduire dans le DAG de réévaluation le premier niveau d'application du DAG d'évaluation qui contient deux nœuds d'évènements `CREATION` et deux nœuds d'orbites brin $\langle\rangle$ et arête $\langle 0\rangle$ incidentes au nœud n_3. Comme évoqué plus tôt, dans un DAG de réévaluation les niveaux d'orbites sont annotés par des brins et non des nœuds. Pour cela, nous récupérons un brin parmi ceux filtrés par le nœud n_3 dans l'application `1-creationCarre`. Puisque n_3 ne crée qu'un unique brin 3 (figure 4.25a), c'est ce brin que nous enregistrons dans le DAG de réévaluation (figure 4.25b).

Lors de l'édition de la spécification paramétrique, nous ajoutons l'application `ADD 10-insertionSommet(3)` (figure 4.26). Puisqu'elle est ajoutée, nous vérifions tout d'abord si elle a un impact sur les évolutions d'orbites représentées dans ce DAG. Pour cela, nous vérifions si la règle filtre tout ou partie d'orbite incidente au brin 3 enregistré dans le DAG. Si ce n'est pas le cas, le niveau d'application inséré est une simple copie du niveau précédent dans lequel les nœuds d'évènements sont tous remplacés par des `NOEFFECT`.

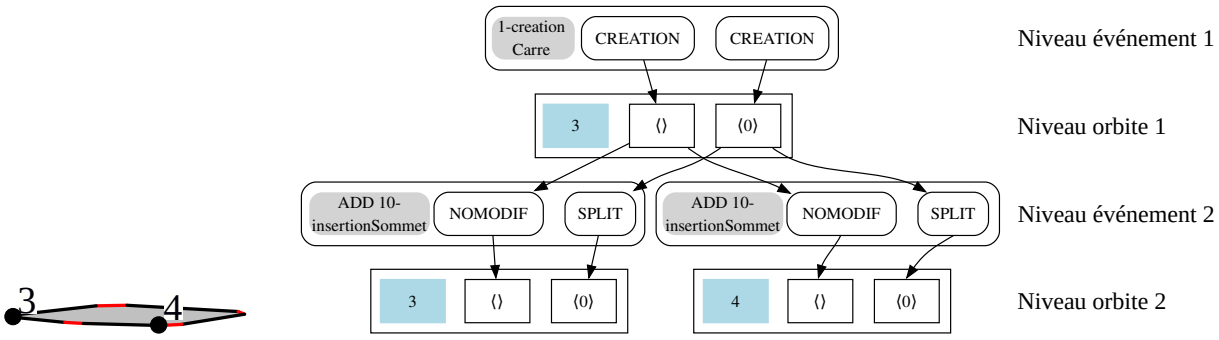


FIGURE 4.26 – Appariement des orbites issues de `ADD 10-insertionSommet`

Dans notre exemple, l'insertion du sommet sur l'arête incidente au brin 3 scinde cette arête en deux. La détection d'une scission, résultant d'une application ajoutée, entraîne le *dédoublement* de la branche de suivi. Cela permet de suivre l'évolution de chacune des entités générées par la scission. Nous avons alors deux branches de suivi avec chacune un niveau d'évènement qui contient un nœud d'évènement `NOMODIF`, car l'orbite $\langle\rangle(3)$ (le brin à l'extrémité de l'arête) est inchangée, et un nœud d'évènement `SPLIT`, car l'orbite $\langle 0\rangle(3)$ (l'arête de face) est scindée en deux (figure 4.26a). De même, chaque branche contient un niveau d'orbites avec les nœuds d'orbites $\langle\rangle$ et $\langle 0\rangle$. Les branches sont distinguées par leurs brins respectifs. En effet, une branche suit désormais l'évolution des orbites incidentes au brin 3 tandis que l'autre suit les orbites incidentes au brin 4 (figure 4.26b).

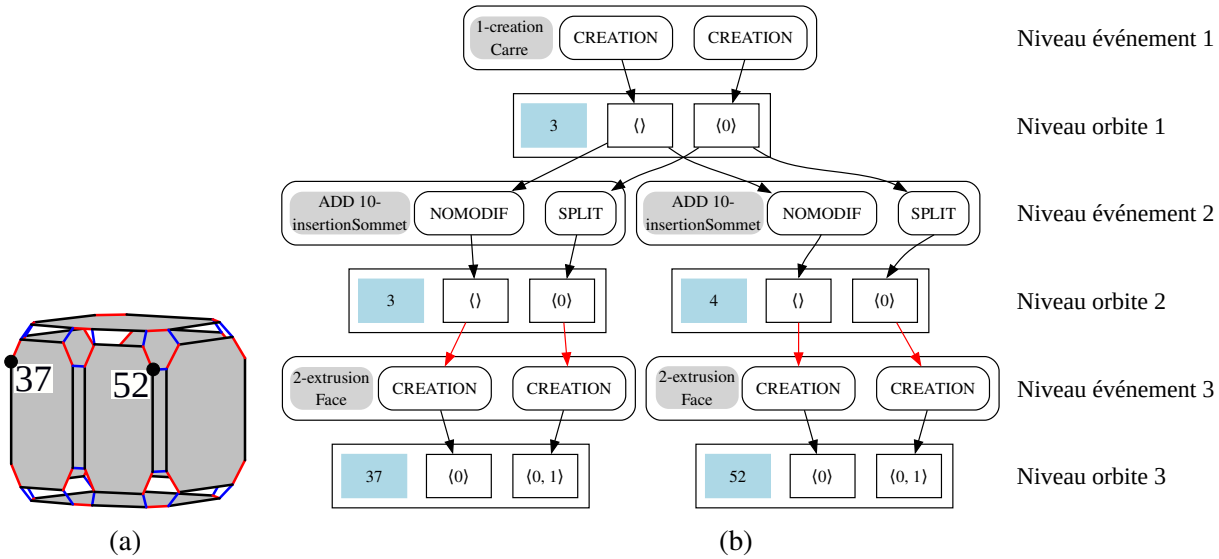


FIGURE 4.27 – Appariement des orbites issues de `INIT 2-extrusionFace`

Suite à la scission ajoutée à l'étape précédente, nous suivons l'impact de l'application suivante dans la spécification paramétrique éditée, `INIT 2-extrusionFace`, pour chaque branche du DAG (figure 4.27). Comme lors du jeu initial, l'application de la règle crée des arêtes par extrusion de brins et des faces par extrusion d'arêtes. Nous enregistrons donc deux nœuds d'évènements `CREATION` et deux nœuds d'orbites $\langle 0\rangle$ et $\langle 0, 1\rangle$ dans chaque branche du DAG (figure 4.27b). Le nœud de règle initialement enregistré, n_4, crée un ensemble de brins, mais seuls

deux d'entre eux sont directement issus des brins 3 et 4. Ce sont respectivement les brins 37 et 52 (figure 4.27a), que nous enregistrons dans les niveaux d'orbites du DAG.

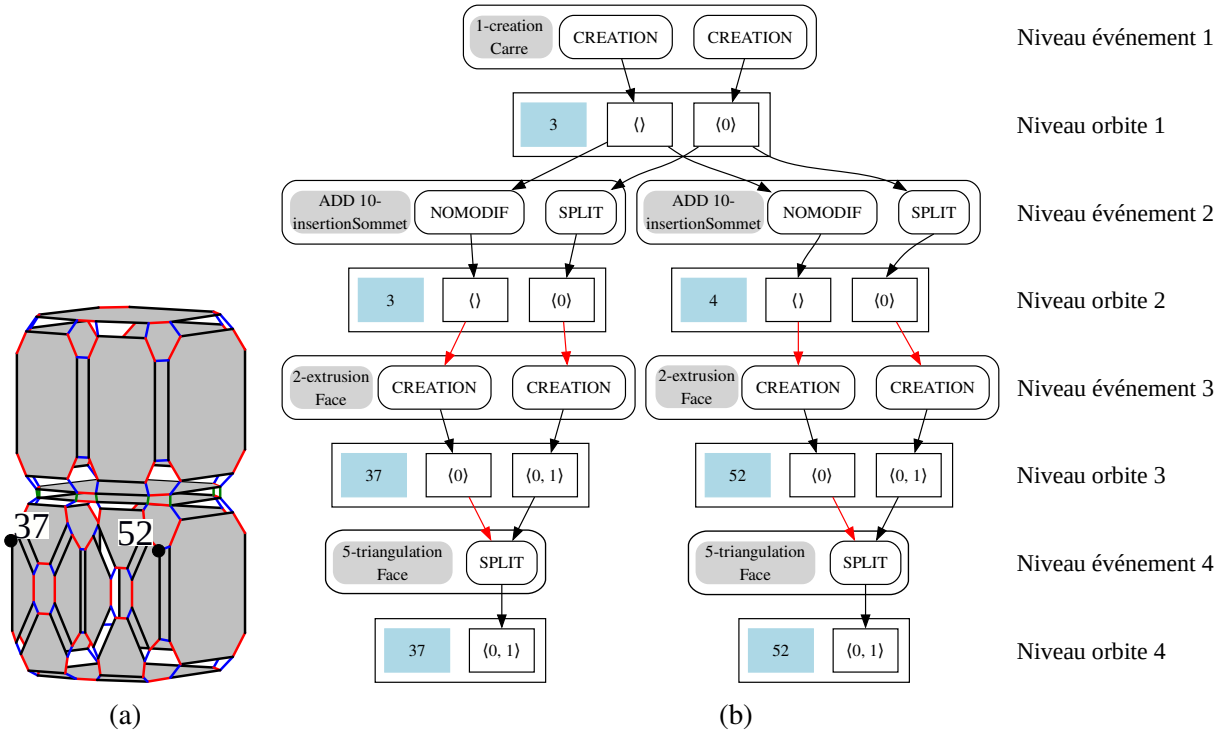


Figure 4.28 – Appariement des orbites issues de `INIT 5-triangulationFace`

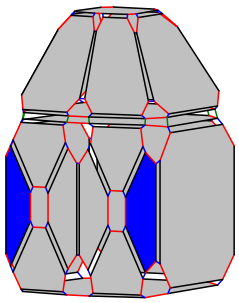


Figure 4.29 – Fin de l'appariement du PN_{35}

La dernière étape pour l'appariement du PN_{35} est l'application `INIT 5-triangulation-Face` (figure 4.28). Lors de cette scission, les faces suivies dans chacune des branches sont scindées, mais puisque cet évènement était prévu dans l'évaluation initiale et que seule la face ayant pour origine $\langle 0 \rangle(n_4)$ était suivie, alors aucun dédoublement de branche n'est nécessaire. Seules les faces ayant pour origine $\langle 0 \rangle(37)$ et $\langle 0 \rangle(52)$ seront suivies dans le DAG de réévaluation. Nous enregistrons donc un nœud d'évènement `SPLIT` ainsi qu'un nœud d'orbite $\boxed{\langle 0, 1 \rangle}$ dans les niveaux d'évènements et d'orbites de chaque branche (figure 4.28b). Dans cette règle, les brins 37 et 52 ont été filtrés et préservés par le nœud $n0$ de la règle de triangulation. Ainsi, nous enregistrons ces deux brins dans les niveaux d'orbites du DAG.

À ce stade de la réévaluation, la procédure d'appariement pour le paramètre topologique de l'application `INIT 8-coloration Face(`PN_{35}`, Bleu)` est terminée. Le DAG de réévaluation indique que deux orbites peuvent être sélectionnées par l'utilisateur pour la réévaluation : les

faces $\langle 0,1\rangle(37)$ et $\langle 0,1\rangle(52)$ (figure 4.28a). Dans cet exemple, nous avons fait le choix de réappliquer la coloration sur les deux faces désignées par le DAG (figure 4.29). Cependant, il est tout à fait possible d'opter pour une approche différente, en paramétrant le comportement du mécanisme de réévaluation.

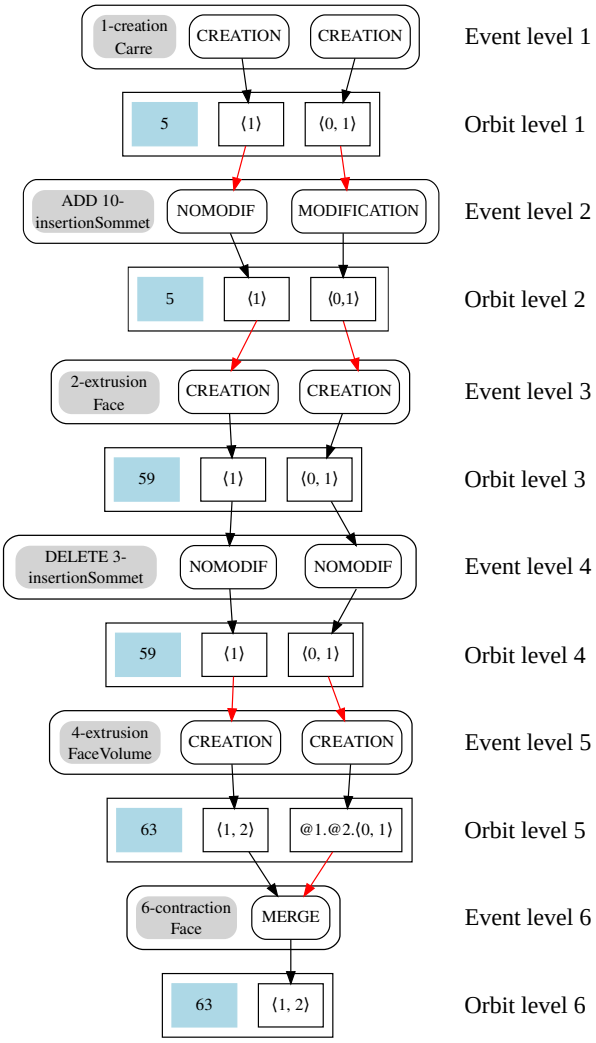


FIGURE 4.30 – Appariement du PN_{55}

Dans le cas du DAG de réévaluation du PN_{55} (figure 4.30), la mise à jour de l'historique d'évolutions ne provoque pas la création de nouvelle branche et identifie un unique sommet pour réappliquer la règle de chanfreinage. L'application `1-creationCarre` produit une face carrée et identifie les orbites $\langle 1\rangle(5)$ et $\langle 0,1\rangle(5)$. L'application ajoutée `ADD 10-insertionSommet` filtre partiellement ces deux orbites donc nous n'enregistrons pas de `NOEFFECT` dans le DAG mais des `NOMODIF`. À l'exception de `DELETE 3-insertionSommet`, les autres applications se déroulent de manière quasi identique à l'évaluation initiale. Contrairement à l'évaluation initiale, où le DAG (figure 4.22) (page 105) enregistre les nœuds d'évènements `MODIFICATION` et `NOMODIF`, nous pouvons voir, lors de la réévaluation, que la suppression de `3-insertionSommet` annule le nœud `MODIFICATION` qui est alors remplacé par `NOMODIF`.

Lors de la réévaluation d'une spécification paramétrique éditée, et plus précisément lorsqu'une application est supprimée, nous gardons dans les DAG une trace des évènements qui

sont annulés. Afin de correspondre à la réalité de la réévaluation, ces évènements sont remplacés par d'autres évènements qui décrivent leurs annulations dans la phase d'appariement. Ainsi un nœud `CREATION` est remplacé par un `NOCREATION` ; `SPLIT` est remplacé par `NOSPLIT` ; `MERGE` est remplacé par `NOMERGE` ; `MODIFICATION` est remplacé par `NOMODIF`. Notons que les nœuds d'évènements `NOMODIF` ne sont pas remplacés. En effet, si l'application d'une règle ne modifie pas une orbite, son annulation ne peut pas la modifier non plus.

Ce suivi des évènements annulés permet de garder trace de toutes les évolutions entre les évènements composant l'histoire d'une orbite du modèle évalué et ceux survenus dans la réévaluation. Cela rend également possible la mise en œuvre de stratégies d'appariement. Par exemple, lorsqu'une orbite référencée dans la spécification paramétrique n'est pas re-créée lors de la réévaluation, nous pouvons tout simplement interrompre l'étape d'appariement et terminer la réévaluation sans erreurs d'appariement. Par exemple, dans le cas d'étude nº 2 (figures 4.3d

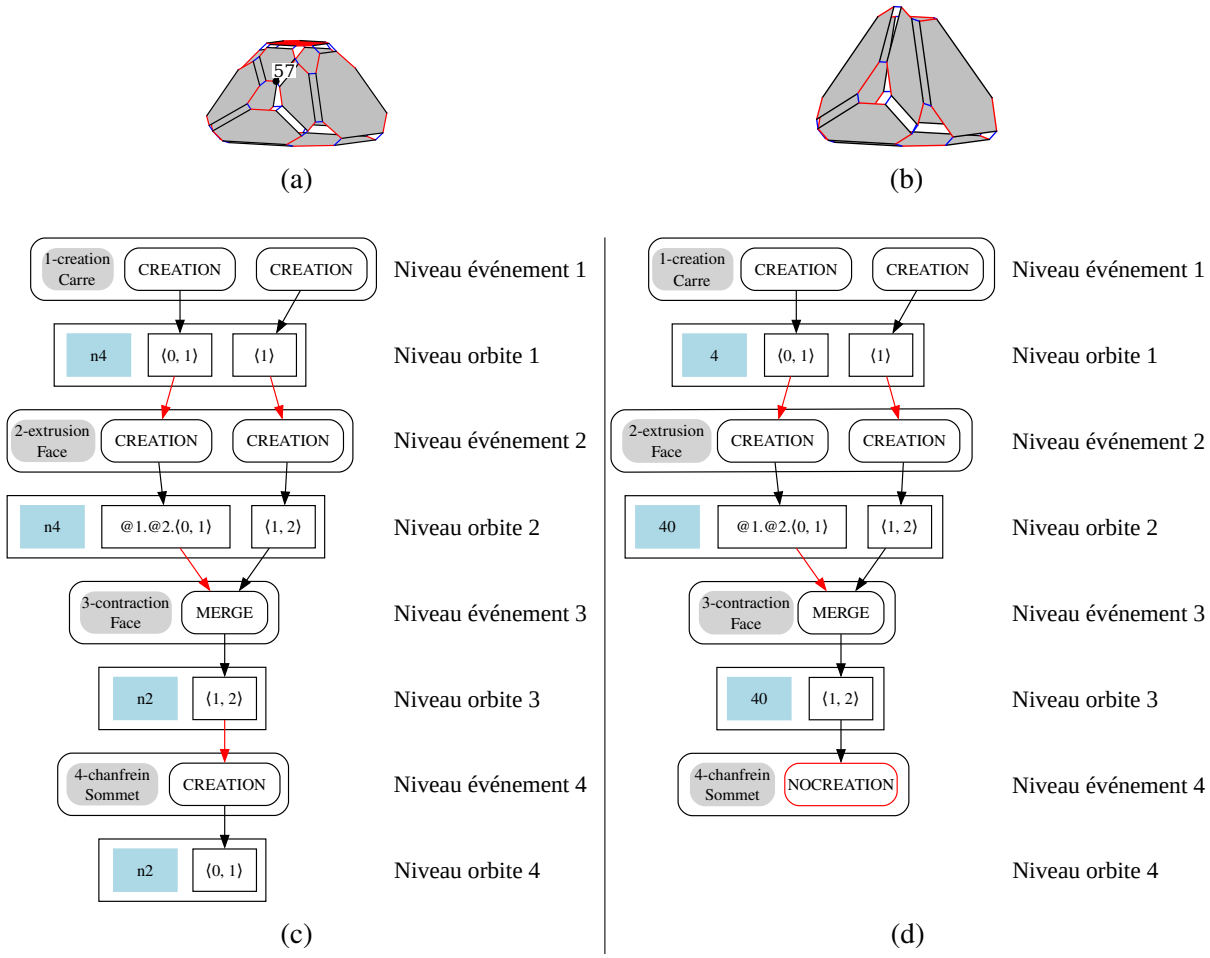


Figure 4.31 – Appariement du PN_{12} du cas d'étude nº 2, objet initial (a), objet re-généré (b), DAG d'évaluation (c) et de réévaluation (d)

et 4.4 page 95), l'opération de chanfrein n'est pas réévaluée et la face à colorier en rouge n'existe plus. Dans ces circonstances, il apparaît contraire à l'intention de conception d'un utilisateur de colorier une autre face qui ne serait pas issue de ce chanfrein. Dans le DAG d'évaluation du

PN_{12} (figure 4.31c), au niveau d'évènement 4, nous observons que l'application `4-chanfrein-Sommet` crée un sommet qui n'est pas recréé dans le DAG de réévaluation (figure 4.31d) ; celui-ci est remplacé un nœud d'évènement `NOCREATION` en rouge. La conséquence immédiate de cet évènement est l'interruption de l'appariement pour ce nom persistant afin de respecter les intentions de conception de l'utilisateur, ce qui donne l'objet illustré dans la figure 4.31b.

5.2 Stratégies de réévaluation

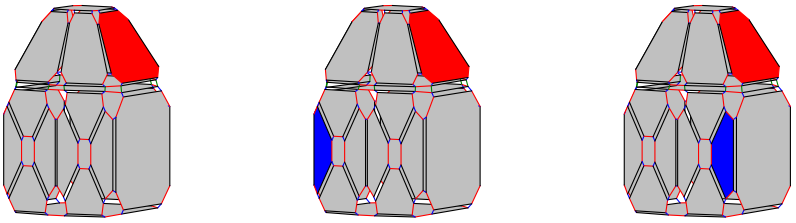
Nous venons de voir que les DAG de réévaluation permettent non seulement de représenter les historiques d'évolutions des orbites, mais aussi les évènements annulés et ajoutés suivant l'édition d'une spécification paramétrique. À l'aide de ces évènements, nous pouvons donc mettre en œuvre des stratégies pour adapter le comportement du mécanisme de réévaluation selon les préférences des utilisateurs. Nous présentons ici plusieurs stratégies de réévaluation qui découlent directement de l'édition d'une spécification paramétrique avec l'ajout et la suppression d'opérations.

Ajout de scission

Nous présentons un premier exemple de scission ajoutée avec le DAG de réévaluation du PN_{35} (figure 4.28). De manière générale, lorsqu'une scission est ajoutée, une branche de suivi d'orbites est ajoutée dans le DAG pour chacune des orbites issues de la scission de l'orbite initialement suivie. Ce comportement nous permet de représenter plusieurs historiques d'évolutions et ainsi proposer plusieurs options de réévaluation aux utilisateurs. C'est à partir de ces choix que nous pouvons définir des stratégies de réévaluations et ainsi permettre aux utilisateurs de configurer le comportement du mécanisme de réévaluation selon leurs projets de modélisation.

Une première stratégie, celle que nous avons utilisée par défaut lors de la réévaluation du PN_{35}, est de réappliquer une opération sur la totalité des orbites désignées par un DAG. Le résultat obtenu est la coloration des deux faces incidentes aux brins 37 et 52 (figure 4.29). Une deuxième stratégie serait de ne réévaluer que l'orbite désignée par la branche principale du DAG. Dans notre exemple, cela reviendrait à ne colorier que la face incidente au brin 37.

Enfin, nous proposons une dernière stratégie qui consiste, lors de la réévaluation, à offrir aux utilisateurs la possibilité de sélectionner explicitement un ensemble d'orbites sur lesquelles réappliquer une opération et de sauvegarder cette configuration. Cette dernière option laisse alors un certain degré de liberté quant à l'exploration dans la conception d'un objet.



(a) Aucune sélection (b) Brin 37 (c) Brin 52

FIGURE 4.32 – Exemples de réévaluation selon la sélection de brins

La figure 4.32 illustre plusieurs exemples de réévaluations selon l'échantillon de brins sélectionnés par l'utilisateur pour l'appariement. Par exemple, aucun brin n'est sélectionné (figure 4.32a), ou seulement l'un des deux (figures 4.32b et 4.32c).

Annulation d'évènements

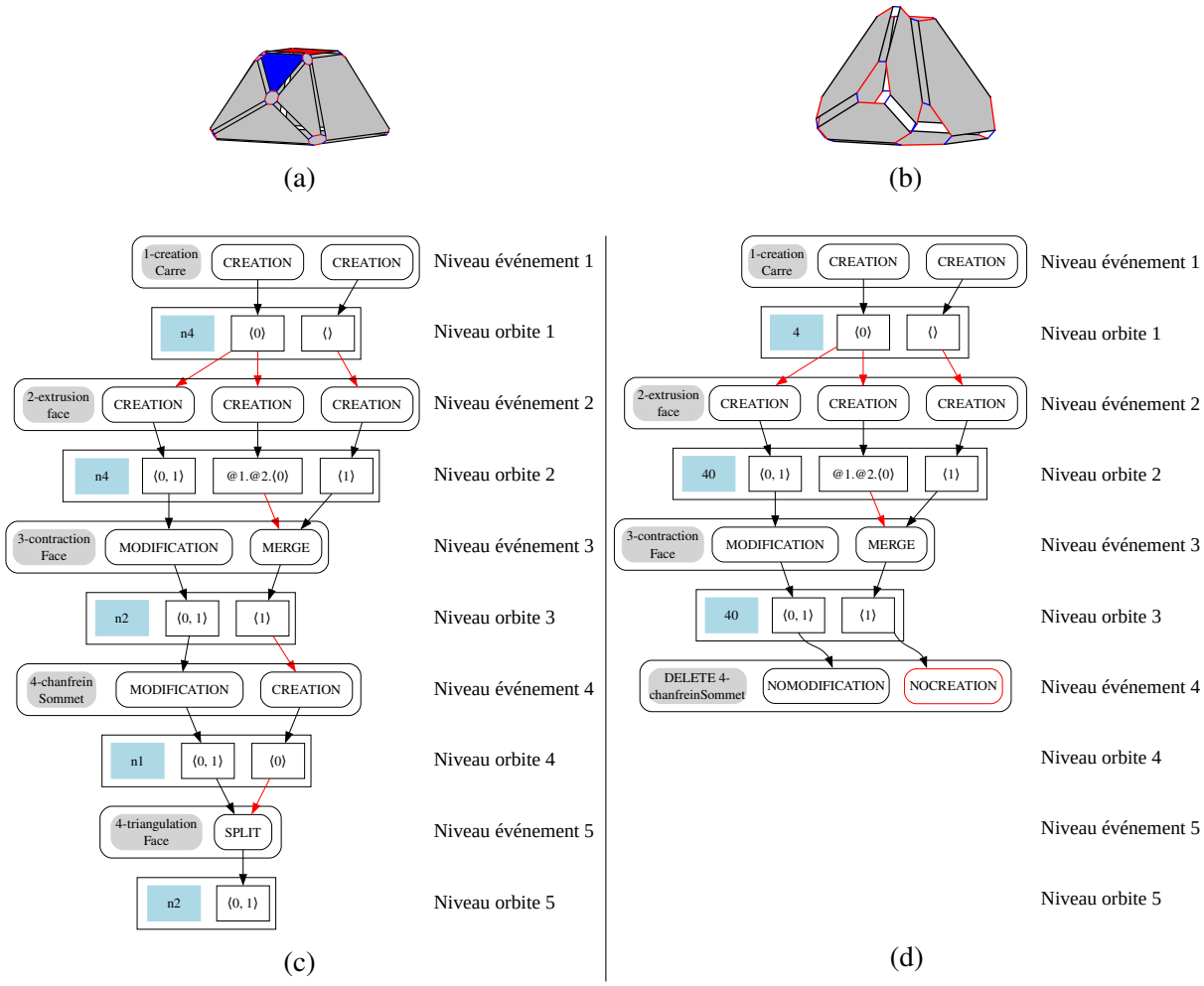


Figure 4.33 – Appariement du PN_{57} du cas d'étude nº 2, objet initial (a), objet re-généré (b), DAG d'évaluation (c) et de réévaluation (d)

Dans l'exemple de l'appariement du PN_{12} (figure 4.31), nous avons montré que le suivi des évènements et de leurs annulations nous permet de prévenir le risque d'un appariement erroné d'orbites et de produire des objets qui respectent les intentions de conception d'un utilisateur. Nous illustrons cette stratégie dans le cas d'une origine qui n'est pas créée et d'une face qui ne peut donc pas être générée à partir de cette origine au rejeu (figure 4.33).

Dans cet exemple, lors de l'évaluation initiale, la face à colorier en bleu provient d'une face scindée et est désignée par une arête (elle-même créée par l'application `4-chanfreinSommet`). Or, dans la réévaluation, cette application est supprimée et l'arête d'origine de la face à colorer n'est pas re-créée. Comme illustré dans la figure 4.33b, la conséquence est que cette face n'est pas re-créée non plus. À la place, nous gardons seulement la trace de l'orbite sommet $\langle 1 \rangle$ à partir

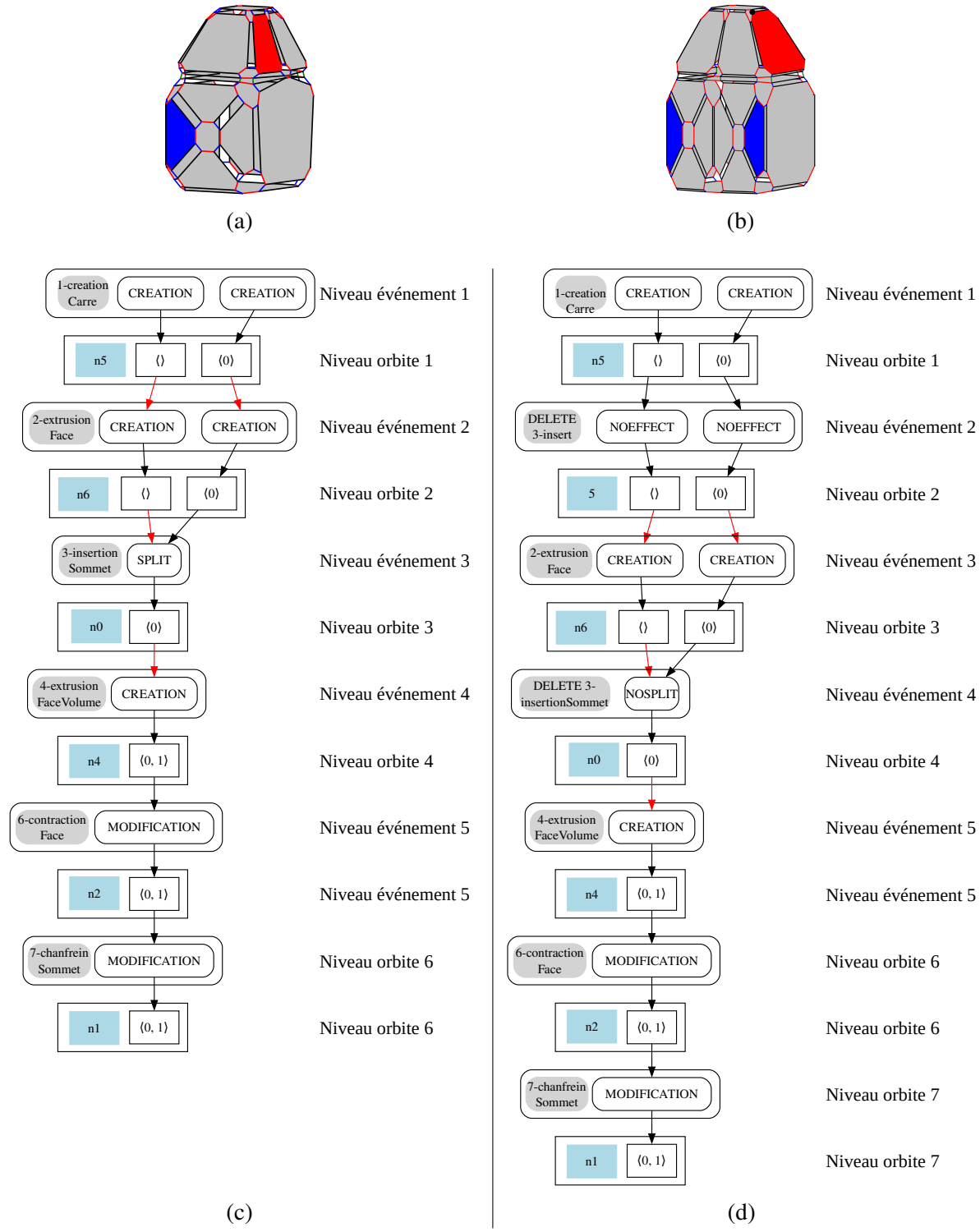


FIGURE 4.34 – Appariement du PN_{68} du cas d'étude nº 1, objet initial (a), objet re-généré (b), DAG d'évaluation (c) et de réévaluation (d)

de laquelle devait être créée l'arête. Or, celle-ci ne correspond pas à une origine de face scindée par la règle de triangulation.

Ainsi, un tel appariement ne correspondrait, *a priori*, en rien aux intentions de conceptions de l'utilisateur. Néanmoins, bien que le type d'orbite sommet de face $\langle 1 \rangle$ n'est pas une origine

convenable, ce sommet peut désigner une face qui possède le même historique de construction que celle qui n'a pas été recréée. Au contraire, une alternative consisterait à demander à l'utilisateur s'il souhaite ou non faire l'appariement sur une telle orbite. Dans le cas de notre exemple, cela reviendrait à colorier la face triangulaire à droite (figure 4.35).

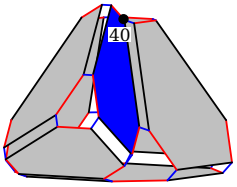


Figure 4.35 – Stratégie d'appariement pour la face non créée

Dans certains cas, la suppression d'une application peut annuler des évènements de scission ou de fusion. Ces suppressions provoquent alors, indirectement, des effets inverses de ceux des évènements annulés. Par exemple, dans l'évaluation de notre cas d'étude nº 1 (figure 4.34), l'application `3-insertionSommet` crée un sommet qui est extrudé en une arête par l'application `4-extrusionFaceVolume`. L'une des faces, le long de cette arête, est ensuite coloriée en rouge par l'application `7-colorationFace` (figures 4.34a et 4.34c). Dans la spécification paramétrique éditée, l'application `3-insertionSommet` est supprimée.

Le sommet n'est donc plus créé ni extrudé dans la réévaluation (figures 4.34b et 4.34d). Ainsi, et de manière indirecte, la suppression de cette application a fusionné deux faces qui étaient distinctes dans l'évaluation initiale. Dans l'hypothèse où les deux faces sont transformées dans l'évaluation initiale, une stratégie pourrait être de ne rien faire et laisser les transformations être résolues de manière séquentielle. Par exemple, si l'autre face avait été coloriée en vert après que la face rouge ait été coloriée, alors la stratégie résoudrait ces deux opérations l'une après l'autre, de sorte qu'à la fin de la modélisation, la face serait coloriée en vert. Au contraire, une alternative consisterait à laisser l'utilisateur choisir les opérations qu'il souhaite réappliquer sur cette face.

6 Implantation

L'approche de réévaluation à base de règles présentée dans les sections précédentes a été partiellement implantée dans Jerboa. Les procédures permettant le support du déplacement d'une opération suite à l'édition d'une spécification paramétrique sont, au moment de la rédaction de ce manuscrit, des travaux en cours. Dans cette section, nous décrivons les implantations des structures de données des DAG d'évaluation et de réévaluation grâce auxquels nous pouvons apparier des orbites.

Pour permettre le mécanisme de réévaluation que nous proposons, nous avons implanté les structures de données illustrées dans les figures 4.36 et 4.37. Les DAG d'évaluation et de réévaluation sont deux structures représentant l'historique d'évolution d'une orbite dans deux contextes différents : l'évaluation et la réévaluation. Ainsi, un DAG trace un historique au travers des opérations enregistrées dans une spécification paramétrique.

Pour l'implantation de ces structures, nous avons fait le choix de représenter une application en deux classes distinctes qui décrivent un niveau d'application. La première classe *EventLevel*

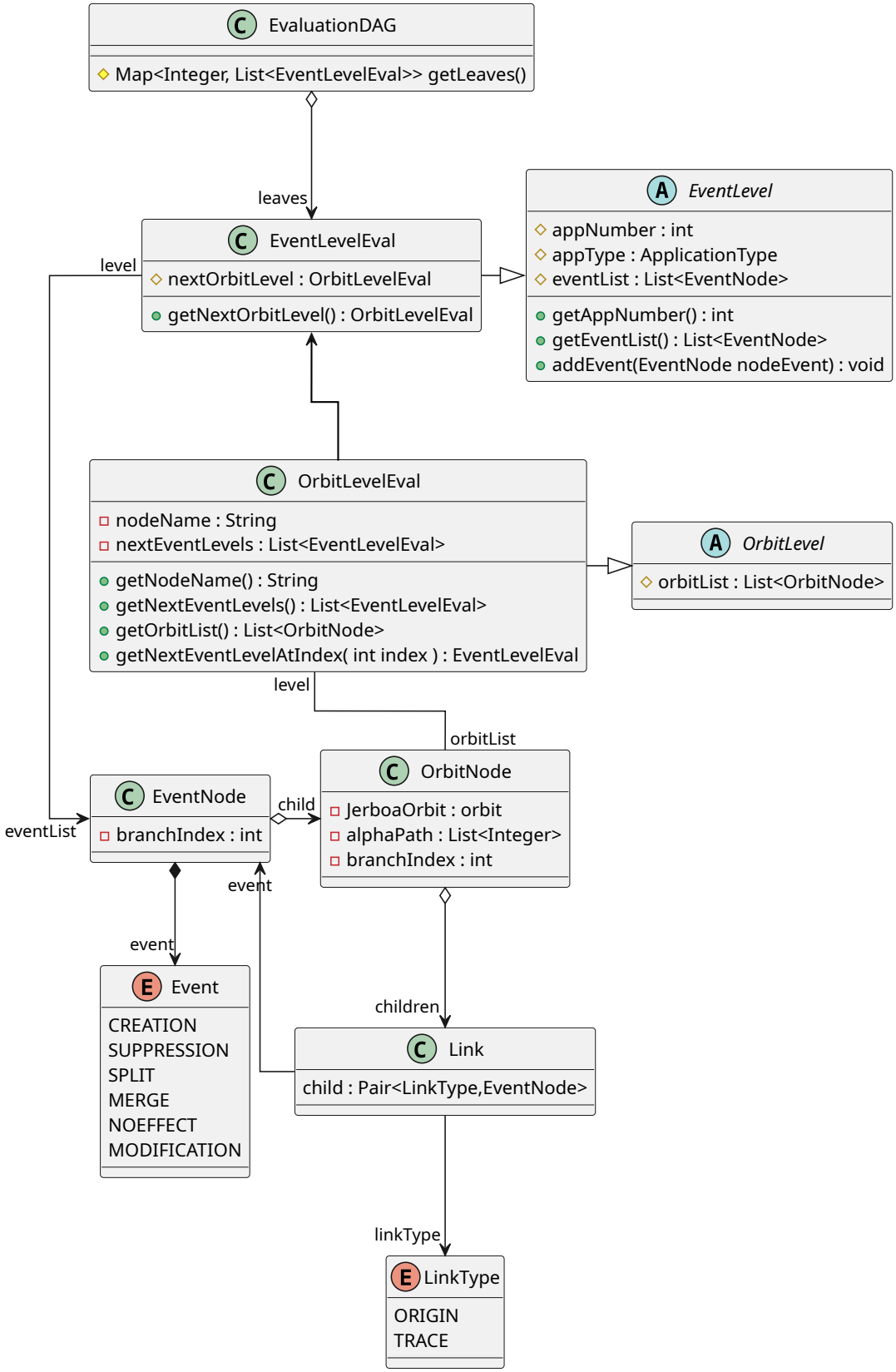


FIGURE 4.36 – Diagramme de la classe *EvaluationDAG*
A : *Abstract class* ; C : *Class* ; E : *Enumeration*

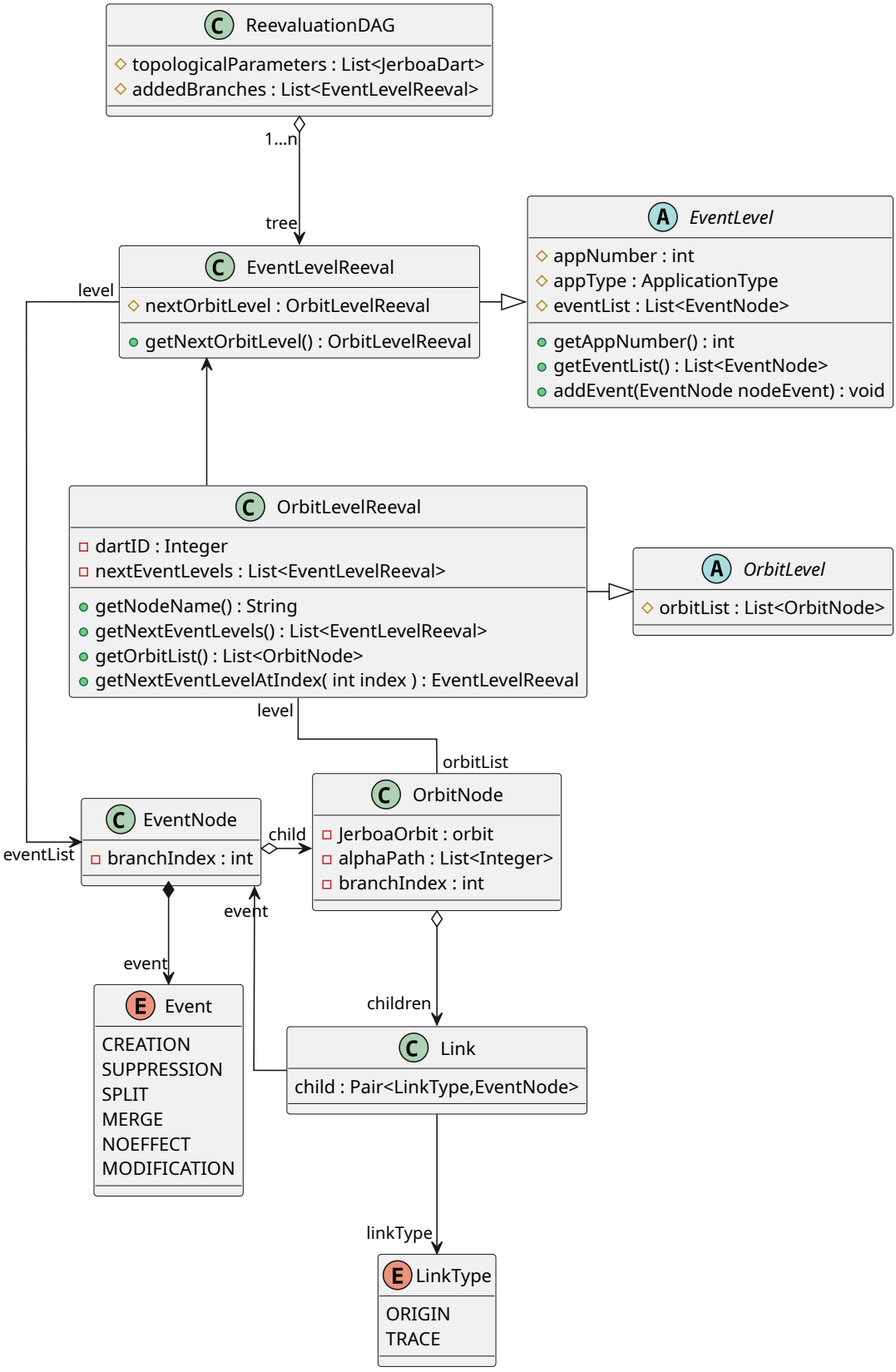


Figure 4.37 – Diagramme de la classe *ReevaluationDAG*
`A` : *Abstract class* ; `C` : *Class* ; `E` : *Enumeration*

décrit une application du point de vue de la règle. Cette classe spécifie qu'un niveau d'évènements détient un numéro d'application *appNum*, un type d'application *appType* et la liste des évènements détectés *eventList* pour une orbite donnée. Notons que lors de l'évaluation initiale, *appType* est initialisée à *INIT*.

La deuxième classe *OrbitLevel* décrit une application du point de vue d'une orbite suivie. En raison de la différence de contexte dans lequel nous utilisons les DAG d'évaluation et de réévaluation, ces deux classes sont implémentées en tant que classes abstraites. Une instance *EventLevelEval* (ou *EventLevelReeval*) détient également une référence vers le niveau d'orbite relatif à la même application. Pour les niveaux d'orbites, rappelons que si à l'évaluation nous enregistrons des noms nœuds afin d'être le plus générique possible, ce n'est pas le cas pour la réévaluation. En effet, dans la phase d'appariement, un DAG de réévaluation enregistre directement des identifiants de brins. Ainsi, une instance de *OrbitLevelEval* détient un nom de nœud *nodeName* et une référence vers le niveau d'évènement de l'application suivante. Au contraire, une instance de *OrbitLevelReeval* détient un identifiant de brin *dartID* et une référence vers le niveau d'évènement de l'application suivante.

À l'intérieur de ces niveaux d'applications, nous organisons en alternance des orbites et des évènements et leurs relations de suivi ou d'origine. Pour représenter les évènements, nous implantons la classe *EventNode* qui détient un tag de type *Event* (une simple énumération des évènements que nous pouvons détecter), un numéro de branche *branchIndex* et une référence vers une unique orbite *child*. Pour les orbites, nous implémentons la classe *OrbitNode* qui détient un type d'orbite *orbitType*, un chemin *alphaPath*, un numéro de branche *branchIndex* et une liste d'enfants *children*. Avec cette implantation, nous exprimons le fait qu'un nœud d'évènement ne peut référencer qu'un seul nœud d'orbite, car tout évènement dans une règle n'est étudié que pour une orbite donnée. Au contraire, une orbite peut être impactée par plusieurs évènements. Notons également que l'enfant d'un nœud d'orbite est une instance de la classe *Link* que nous créons pour associer un nœud d'évènements à un type de relation *LinkType*, c'est-à-dire soit un suivi *TRACE*, soit une origine *ORIGIN*.

Enfin, nous avons décidé d'encapsuler les niveaux d'application dans une liste afin de pouvoir itérer sur le contenu d'un DAG. Toutefois, cela ne considère pas la possibilité que plusieurs branches soient nécessaires pour représenter pleinement l'historique des évolutions d'une orbite. Ainsi, pour les DAG d'évaluation, nous décidons d'encapsuler la structure de données dans une *Map* où chaque niveau d'application est accessible directement à partir de son *appID*. Le choix d'une *Map* présente un aspect pratique, car, au cours de la procédure de réévaluation, nous avons besoin de récupérer des morceaux de DAG d'évaluation niveau par niveau. Au contraire, pour les DAG de réévaluation, la structure de données est simplement encapsulée dans une autre liste, puisque l'accès se fait toujours par rapport au dernier niveau inséré.

7 Conclusion

Dans ce chapitre, nous présentons les structures de données et procédure nécessaires à l'implantation d'un mécanisme de réévaluation dans un système de modélisation à base de règles. La difficulté d'un tel mécanisme est la phase d'appariement entre les entités topologiques du modèle initial et celles du modèle réévalué. Pour cela, nous présentons une approche exploitant les formalismes des cartes généralisées, des règles Jerboa et des évènements que nous avons définis dans le chapitre 3.

Lors de la conception d'un objet, tout brin d'une carte généralisée est représenté par son historique d'évolutions. Celui-ci référence l'ensemble des applications et les nœuds de règles ayant conduit à sa création ou sa modification. Puisqu'un brin est explicitement désigné lors de l'application d'une règle et que son historique est unique, nous utilisons celui-ci comme nom persistant pour désigner une orbite.

Toutefois, la nomination des brins ne suffit pas, car elle ne représente pas explicitement l'historique des évolutions d'une orbite. Nous présentons alors un second mécanisme permettant de reconstituer les historiques d'évolutions des orbites identifiées par des noms persistants. Pour cela, nous enregistrons dans un DAG chacune des applications qui interviennent dans l'évolution d'une orbite. Les évènements sont détectés par analyse statique des règles, ce qui nous permet de représenter à la fois le suivi d'une orbite et le suivi de chacune des origines qui peuvent être détectées. De plus, nous complétons la notion d'origine à l'aide d'un contexte, automatiquement calculé au moment de l'analyse, qui nous permet de garantir la cohérence du suivi d'orbite. Ces deux mécanismes de reconstitution d'historiques pour les brins et les orbites représentent, en réalité, notre mécanisme de nomination persistante. Celui a ainsi l'avantage d'être robuste et efficace dans le sens où seules les orbites disposant d'un nom persistant sont décrites par un historique enrichi.

Enfin, nous avons présenté notre mécanisme d'appariement des entités pour la réévaluation. Dans cette contribution, les DAG d'évaluation nous servent de bases pour créer des DAG de réévaluations, c'est-à-dire des DAG qui prennent en compte les modifications apportées au modèle géométrique en fonction de l'édition d'une spécification paramétrique. Afin d'offrir un certain degré de liberté lors de la réévaluation, nous proposons également des stratégies configurables par les utilisateurs. Ces stratégies leur permettent, entre autre, de sélectionner un ensemble de paramètres sur lesquels réappliquer des opérations lorsque, par exemple, une entité est scindée par une opération ajoutée. D'autres stratégies permettent de prendre en compte la suppression d'orbite ainsi que le déplacement d'opérations sont en cours d'implantation.

CHAPITRE 5

Extension de la réévaluation aux scripts

Sommaire

1 Introduction

Dans les chapitres précédents, nous avons présenté les règles Jerboa et les moyens mis en œuvre pour réévaluer des modèles paramétriques construits à l'aide de ces règles. Cependant, cette approche ne permet pas de créer des opérations complexes avec ces règles, c'est-à-dire des opérations structurées à l'aide de structures de contrôle. Or, le langage Jerboa propose des opérations sous forme de scripts de règles, permettant d'organiser les règles avec des structures de contrôle largement répandues dans la plupart des langages de programmation. L'objectif de ce chapitre est donc d'étendre notre système de réévaluation pour permettre la réévaluation des scripts de règles.

Pour cela, nous proposons d'étendre notre système de réévaluation à des scripts présentant un ensemble limités de structures de contrôles, à savoir des séquences de règles, des itérations sur les sous-orbites d'une orbite ainsi que des alternatives « si… alors… sinon… ». Avec de telles structures, il devient nécessaire d'enregistrer dans les DAG tous les paramètres associés à l'utilisation de ces structures, tels que les conditions, les paramètres d'itération, et autres éléments de contrôle. Si, par exemple, une boucle était utilisée pour appliquer une opération sur chacun de ses sommets, il est essentiel d'enregistrer dans les DAG l'historique de la face parcourue. En effet, si cette face venait à disparaître, la boucle ne pourrait pas être rejouée, ce qui empêcherait l'application des opérations sur les sommets. Cela affecterait directement les appariements liés à ces opérations, compromettant ainsi la cohérence du modèle et de son réévaluation.

Ce chapitre est structuré comme suit. Nous commençons par une description du langage de script Jerboa dans la section 2. Nous y détaillons les fonctionnalités que nous retenons dans le cadre de l'étude de la réévaluation de scripts. Nous présentons ensuite, dans la section 3, les séquences de règles et les problématiques liées à l'enregistrement des opérations contenues dans un script. Par la suite, dans les sections 4 et 5, nous détaillons le support des structures itératives puis alternatives et nous voyons la façon dont ces structures sont intégrées dans les DAG d'évaluation et de réévaluation pour permettre les appariements. Nous illustrons ensuite notre approche dans la section 6 en présentant des exemples de réévaluations ou nous combinons les différentes structures de contrôle. Enfin, nous proposons un état des lieux de l'implantation réalisée pour la réévaluation avec des scripts avant de conclure dans la section 7.

2 Scripts

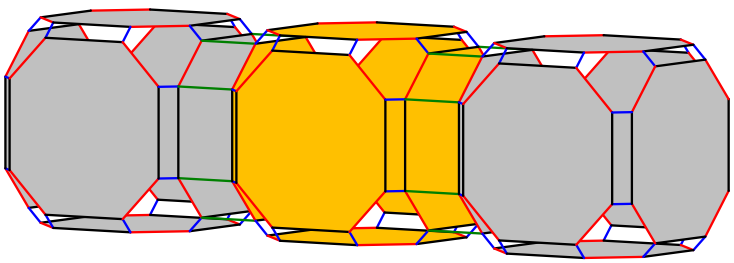


Figure 5.1 – Exemple de volume à découdre par Gauthier

Les règles Jerboa nous permettent de représenter diverses opérations de modélisations plus ou moins complexes. Toutefois, comme nous l'avons vu dans le chapitre 1 avec la définition 15

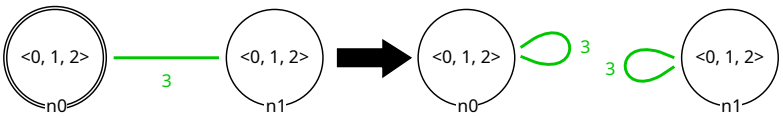


FIGURE 5.2 – Règle de décousure d'un volume par ses 3-liaisons

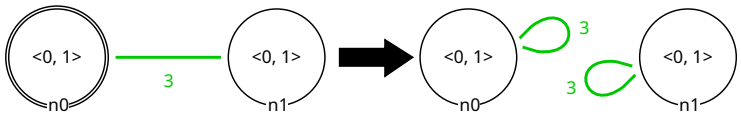


FIGURE 5.3 – Règle de décousure d'une face par ses 3-liaisons

(page 21), les orbites filtrées par la partie gauche d'une règle doivent être isomorphes à renommage près. La figure 5.1 illustre un exemple où la décousure d'un volume par ses 3-liaisons (figure 5.2) est impossible, car certaines des faces du volume sont libres. Elles ne possèdent pas de 3-liaisons à découdre et le volume **jaune** ne peut pas être déconnecté du reste de la composante connexe. Pour déconnecter ce volume, il faudrait appliquer la règle de décousure de face (figure 5.3) sur chacune des faces incidentes à un autre volume.

2.1 Scripts Jerboa

Les scripts Jerboa [Bel+14 ; Gau19] permettent, à l'aide d'un langage de script, de construire des opérations avec des comportements plus complexes que les règles. Un script peut faciliter, par exemple, le processus de décousure du volume **jaune** en évitant de réappliquer manuellement la règle de décousure de face (figure 5.1). Le langage utilisé pour écrire les scripts Jerboa définit l'ensemble des instructions nécessaires pour manipuler les G-cartes et leurs orbites. Les structures de contrôles classiques de Java et du C++ sont disponibles, à savoir les boucles (`for`, `foreach`, `while`) et les alternatives. Plus particulièrement, ces scripts peuvent appeler d'autres opérations. Ces appels sont les instructions élémentaires d'un script qui, dans sa forme la plus simple, est une séquence d'appels de règles.

FIGURE 5.4 – Motif gauche du script de décousure de volume

```
for(JerboaDart face : <0,1,2>_<0,1>(volume#0))
{
    try
    {
        @rule<UnsewFace01A3>(face);
    } catch (JerboaException e){
        // echec de l'application
    }
}
return null;
```

Listing 5.1 – Script de décousure d'un volume

Les scripts, comme les règles, sont des opérations. À ce titre, elles disposent de paramètres en entrée qui peuvent être des paramètres topologiques ou de plongement (géométrie, couleur, etc.). En pratique, ces paramètres sont utilisés différemment. Observons le motif gauche du script qui découd un volume de sa composante connexe (figure 5.4). Ce motif est composé d'un unique nœud d'accroche appelé `volume`. Son étiquette désigne un brin ($\langle\rangle$) ainsi qu'une information de *multiplicité* (`1`) qui indique le nombre d'orbites sur lesquelles ce script peut être appliqué ; ici, une seule orbite. Une différence notable, entre les motifs gauches des règles et de scripts, est que les motifs de scripts ne filtrent pas d'orbites. Ceux-ci récupèrent simplement une liste de brins, un par orbite, et délèguent le filtrage aux règles qu'ils appellent. Étudions maintenant le script (liste 5.1). La principale structure de ce script est une boucle qui itère sur chacune des faces d'un volume incident à un brin obtenu par le nœud `volume`. L'itération est formalisée par l'instruction `<0,1,3>_<0,1>(volume#0)` qui retourne une liste de brins à raison d'un brin par face. Notons, d'une part que `volume#0` est le premier (et unique brin) contenu dans `volume` et, d'autre part, que la variable `face` est effectivement un brin de type `JerboaDart`. Dans cette boucle, un bloc `try-catch` contient un appel de la règle `UnsewFace01A3` qui doit alors être appliquée à chacune des faces du volume. Celui-ci empêche une sortie prématurée du programme lorsque la règle échoue suite à son application sur une face libre. Enfin, notons que, dans les règles, le motif droit construit automatiquement la sortie de son application, c'est-à-dire un tableau listant tous les brins préservés ou créés par les nœuds du motif. Pour exploiter le résultat d'application d'une règle, celui-ci doit être enregistré dans une variable de type `JerboaRuleResult`. Au contraire, pour les scripts, une telle sortie n'est pas créée automatiquement. Ainsi, si elle existe, elle doit être explicitement construite par le développeur et retournée.

Ainsi, les scripts Jerboa permettent d'écrire des opérations plus complexes que les règles puisqu'ils proposent d'organiser des opérations à l'aide de structures de contrôle. Une opération, voire une séquence d'opérations, peut être répétée un certain nombre de fois à l'aide de boucles. Une application peut être soumise à condition et le script peut contenir une alternative selon l'évaluation de la condition. Enfin, puisqu'une règle et un script sont tous deux des opérations, un script peut adopter un comportement d'autant plus complexe qu'il peut appeler un autre script.

2.2 Intégration ouverte ou fermée dans une spécification paramétrique

Afin d'étendre notre système de réévaluation avec le support des scripts de règle, nous réutilisons une partie du langage de scripts Jerboa. Ainsi, dans ce chapitre, nous étudions la réévaluation de scripts construits avec un sous-ensemble des structures de contrôles restreint aux séquences, aux boucles `foreach` et aux alternatives de règles.

Une première question se pose sur l'intégration des scripts dans notre système de réévaluation : comment représenter un script dans une spécification paramétrique ? À travers cette question, nous nous intéressons aux changements qui doivent être apportés à la spécification paramétrique ainsi qu'aux DAG d'évaluation et de réévaluation.

Listing 5.2 – Exemple de spécification « boîte fermée »

```
1 1-uneRegle();
2 2-unScript(PN1);
3 3-uneAutreRegle(PN2);
```

Une première possibilité consiste à représenter un script uniquement par ses entrées et ses sorties, c'est-à-dire selon selon un modèle de boîte fermée (ou *black box* en anglais) comme présenté

dans le listing 5.2. Étant donné qu'il s'agit d'une boîte fermée, le système de réévaluation ne pourra ni connaître l'ensemble des règles évaluées, ni analyser leur déroulement pour suivre l'évolution des orbites. Dans ce cas, il revient au développeur de définir précisément et de manière persistante les paramètres de sortie du script, en particulier lorsque celui-ci est réévalué dans un contexte différent. Imaginons, par exemple, que dans le listing 5.2, `3-uneAutreRegle(`$\texttt{PN}_2$`)` soit appliquée sur un sommet retourné par l'application `2-unScript(`$\texttt{PN}_1$`)`. La boîte du script étant fermée, il revient aux développeurs de s'assurer que, dans un contexte de réévaluation différent, le sommet retourné représente toujours l'intention de conception. Cela revient donc à demander au développeur du script de résoudre manuellement le problème de la nomination persistante. Cette approche est évidemment limitée, car un script complexe peut créer, supprimer ou modifier de nombreuses orbites. Cependant, il peut être pertinent de définir explicitement quelques orbites caractéristiques comme sorties du script.

Une seconde possibilité consiste à utiliser une boîte ouverte (ou *glass box*). Dans ce cas, l'ensemble des orbites créées, supprimées ou modifiées à l'intérieur du script restent accessibles. Elles doivent pouvoir être suivies, nommées de manière persistantes et appareillées de façon automatique, au même titre que les orbites gérées par de simples règles. Dans les sections suivantes, nous étudions en détail l'utilisation de boites ouvertes. Nous analysons les différentes structures de contrôle (la séquence, la boucle et la conditionnelle) ainsi que leur intégration dans la spécification paramétrique et dans les DAG d'évaluation, avant de voir comment elles sont utilisées dans les DAG de réévaluation.

3 Structure de séquence

La première structure contenue dans un script, que nous étudions est la séquence de règles.

```
1 @rule<ruleA>(brin);
2 @rule<ruleB>(brin);
3 @rule<ruleC>(brin);
```

Une séquence permet d'appeler successivement plusieurs opérations. Le script peut donc dans ce cas être comparé à une spécification paramétrique qui appelle successivement les règles qui la compose. En adoptant une approche en boîtes ouvertes, les règles contenues dans le script et leurs paramètres sont directement enregistrés dans la spécification paramétrique. Cela permet de suivre les évolutions des paramètres de manière très similaire à ce qui était réalisé dans le chapitre 4 précédent, grâce aux noms persistants des brins et aux DAG d'évaluation.

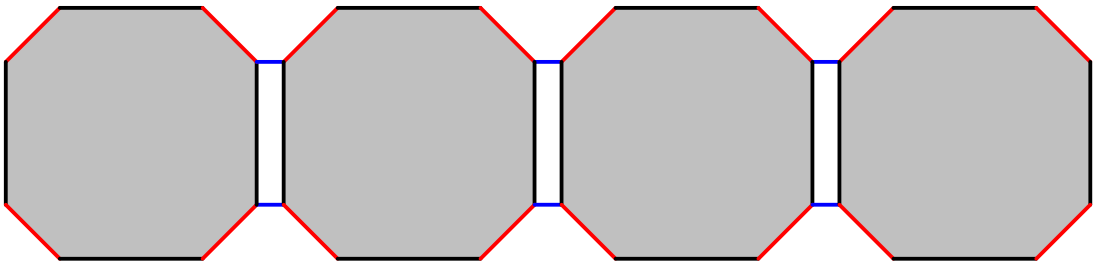


FIGURE 5.5 – Ruban

```
JerboaRuleResult face2 = @rule<extrusionAreteFace>(face#0);
JerboaRuleResult face3 = @rule<extrusionAreteFace>(face2#4#0);
@rule<extrusionAreteFace>(face3#4#0);
```

Listing 5.3 – Script `S_extrusionRuban`

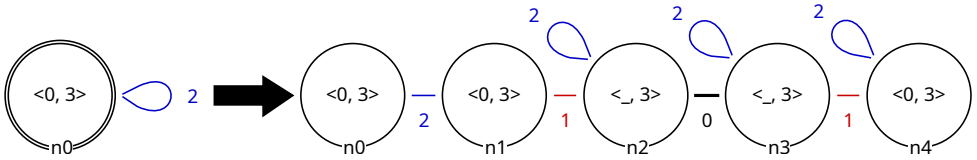


Figure 5.6 – Règle d'extrusion d'une arête de face

Prenons le script (listing 5.3) qui construit la rangée de faces illustrée dans la figure 5.5. Ce script prend une face en entrée et contient trois appels de la règle `extrusionAreteFace`. Chaque appel crée une nouvelle face dans l'objet par l'extrusion d'une arête. Le premier appel prend en paramètre le premier brin filtré par le nœud d'accroche (appelé `face`). Ce brin, `face#0`, désigne l'arête à extruder. La sortie de la règle, c'est-à-dire les brins filtrés par son motif droit, est ensuite enregistrée dans la variable `face2` de type `JerboaRuleResult`. Le deuxième appel prend en entrée un brin enregistré dans le résultat `face2`. Il s'agit du premier brin filtré par le nœud $n4$ de la règle d'extrusion illustrée dans la figure 5.6. Ce brin est noté `face2#4#0`. Enfin, le troisième appel est exécuté de la même manière.

3.1 Extension de la spécification paramétrique

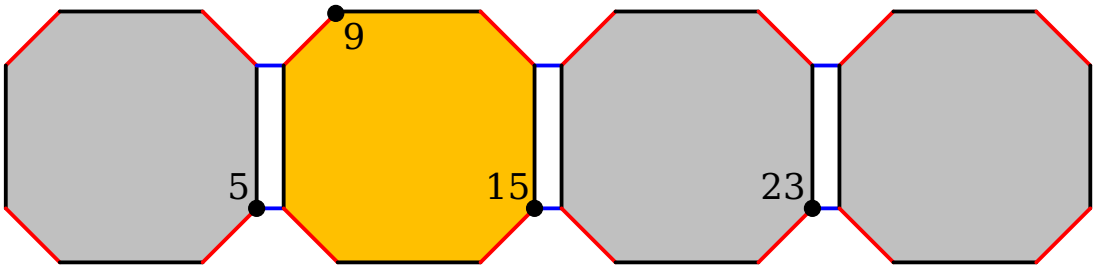


Figure 5.7 – Ruban avec coloration d'une face

Pour assurer une nomination persistante des orbites utilisées comme paramètres à l'intérieur des scripts, il est nécessaire d'enregistrer explicitement, dans la spécification paramétrique, chacune des règles invoquées ainsi que leurs paramètres.

```
1-creationCarre()
2-S_extrusionRuban(PN5)
  1-extrusionAreteFace(PN5)
  2-extrusionAreteFace(PN15)
  3-extrusionAreteFace(PN23)
3-colorationFace(PN9, Orange)
```

Listing 5.4 – Spécification paramétrique avec séquence

En conséquence, nous commençons par ajouter un bloc `Sequence` pour mettre en évidence une séquence d'applications issues d'un script. Ensuite, nous amorçons une nouvelle séquence

de numérotation dans chaque script. Ainsi, les trois applications du script `S_extrusionRuban` sont respectivement numérotées `1`, `2` et `3`. Enfin, puisque les paramètres topologiques sont accessibles, il est nécessaire d'étendre le schéma de nomination. Ainsi, l'historique d'un brin dont l'évolution est impactée par une application de script sera représentée selon le schéma suivant :

$$[S\{R\ noeud\}]$$

où S est le numéro d'application d'un script et R le numéro d'application d'une règle dans un script. Notons que le motif $(R\ noeud)$ est répété pour chaque évolution du brin à l'intérieur du script.

Détaillons la spécification paramétrique dans le listing 5.4 de l'objet illustré figure 5.7. L'application `1-creationCarre()` crée le brin 5. L'historique de ce brin est $[1n5]$. L'application `2-S_extrusionRuban(PN5)` entraîne les applications successives de trois règles d'extrusions à partir du brin 5. Comme prévu, nous démarrons une nouvelle séquence de numérotation dans le script. L'application `1-extrusionAreteFace(PN5)` crée une face à partir d'une arête ($\langle 0 \rangle$) ainsi que les brins 9 et 15. Les historiques de ces brins sont respectivement $[1n6; 2\{1n2\}]$ et $[1n5; 2\{1n4\}]$. L'application `2-extrusionAreteFace(PN15)` crée une nouvelle face ainsi que son brin 23. L'historique de ce brin est $[1n5; 2\{1n4; 2n4\}]$. La dernière application du script, `3-extrusionAreteFace(PN23)` crée la quatrième et dernière face. Enfin, dans le bloc principal, la dernière application `3-colorationFace` colorie la face identifiée par le brin 15.

L'ouverture des scripts permet ainsi de garder les opérations internes accessibles, ce qui facilite la nomination persistante des brins, de manière très similaire à ce qui était réalisé avant l'utilisation des scripts. Comme nous allons le voir, cela permet également de suivre les orbites à travers les DAG d'évaluation.

3.2 DAG

Avec le mécanisme de nommage persistant des brins et l'enregistrement des paramètres topologiques à l'intérieur des scripts, nous avons la possibilité de reconstituer leurs évolutions. Par extension, ce suivi nous permet, comme pour les règles, de proposer leurs appariements (s'ils existent) lors de la réévaluation.

Pour mettre en œuvre ce support, nous ajoutons deux propriétés aux DAG. Premièrement, les applications internes à un script sont regroupées dans le bloc de ce script avec leur propre numérotation. Ensuite, la réévaluation d'un script est conditionnée par le suivi de son paramètre topologique dans une nouvelle branche. Pour cela, une nouvelle relation est définie entre un nœud d'orbite et un bloc de script. Ainsi, en fonction de la stratégie mise en œuvre, la réévaluation pourra tenir compte de l'évolution du paramètre topologique, et par exemple, ne plus réévaluer le script si l'orbite correspondant à ce paramètre est supprimée.

Prenons, par exemple, le DAG d'évaluation du PN_9 illustré dans la figure 5.8. Ce DAG représente l'évolution de l'orbite face qui est coloriée en **jaune** par l'application `3-colorationFace`. Nous pouvons observer les propriétés que nous venons de présenter. Tout d'abord, une boîte de script englobe les applications qui appartient à l'application `2-S_extrusionRuban`. Ensuite, le suivi du paramètre topologique du script est effectué dans une nouvelle branche. L'orbite représentée par le nœud $\boxed{\langle 0, 1, 3 \rangle}$ est liée au bloc de script par une flèche **verte**. Cette branche identifie une face qui est créée par la première application.

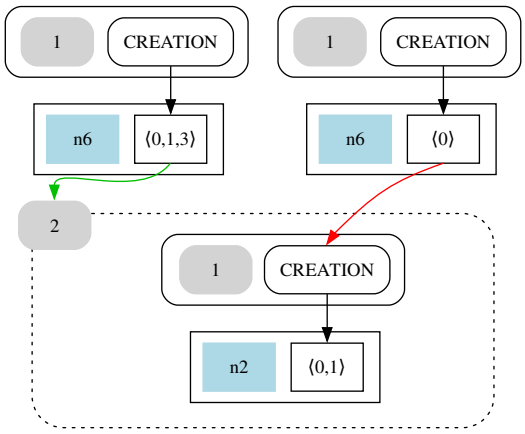


Figure 5.8 – DAG d'évaluation du PN_9

```
DELETE 1-creationCarre()
2-extrusionRuban(PN5)
  1-extrusionAreteFace(PN5)
  2-extrusionAreteFace(PN15)
  3-extrusionAreteFace(PN23)
3-colorationFace(PN9)
```

Listing 5.5 – Spécification paramétrique avec séquence

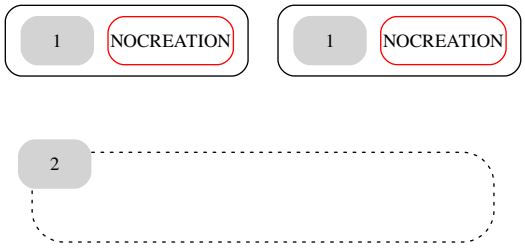


Figure 5.9 – DAG de réévaluation du PN_9

Étudions maintenant le DAG de réévaluation du PN_9 (figure 5.9) étant donné l'édition de la spécification présentée dans le listing 5.5. Dans cette édition, la première application est supprimée. Celle-ci annule alors la création des deux orbites suivies, c'est-à-dire les paramètres topologiques du script et de sa première application. Avec une telle édition, le script ne peut être réévalué ce qui annule également la réévaluation de son contenu.

4 Structure d'itération

La deuxième structure contenue dans un script, que nous étudions, est la boucle de type `foreach`.

```
for(JerboaDart brin_sous-orbite : <type_orbite>_<type_sous-orbite>(
    brin_orbite))
{
    @rule<ruleA>(brin_sous-orbite);
}
```

Le langage de script Jerboa met à disposition une instruction de la forme `<orbite>_<sous-orbite>`. Celle-ci permet de lister les instances d'un type $\langle\omega'\rangle$ de sous-orbite dans une $\langle\omega\rangle$-orbite. Utilisée comme paramètre d'itération d'une structure `foreach`, elle permet de répéter l'application d'une règle sur chacune des sous-orbites $\langle\omega'\rangle$ d'une orbite $\langle\omega\rangle$. Dans le cadre de la réévaluation, cela nous permet d'enregistrer chacune des applications dans une boucle et de proposer une solution pour tracer l'évolution de ces sous-orbites. Ainsi, le support des boucles `foreach` nécessite de les intégrer dans les spécifications paramétriques ainsi que dans les DAG, notamment avec les suivis de l'orbite à itérer et de ses sous-orbites.

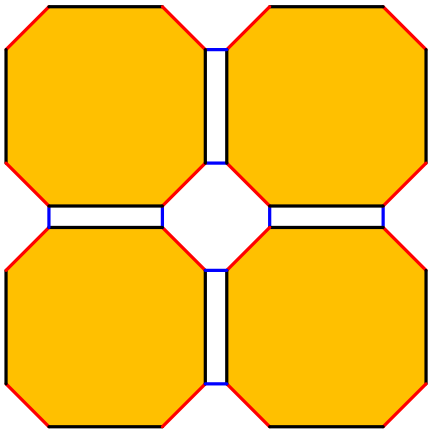


FIGURE 5.10 – Coloration d'une composante connexe

```
1 for(JerboaDart face : <0,1,2,3>_<0,1,3>(composante#0)){
2     @rule<colorationFace>(face)
3 }
```

Listing 5.6 – Script `S_colorationComposante`

Prenons le script listing 5.6 qui colorie toutes les faces ($\langle 0,1,3\rangle$) d'une composante connexe ($\langle 0,1,2,3\rangle$) illustrée dans la figure 5.10. Ce script prend une composante connexe et répète l'application `colorationFace` sur chacune de ses faces. À chaque itération, l'application prend en paramètre une instance de face, c'est-à-dire un brin, obtenue dans la boucle.

4.1 Extension de la spécification paramétrique

De la même manière que nous avons introduit les séquences dans la spécification paramétrique, nous ajoutons un bloc `For` lorsqu'un script possède une boucle de type foreach. Cependant, un bloc `For` nécessite l'enregistrement de ses paramètres d'itération. L'un représente l'orbite principale sur laquelle est effectuée la boucle et l'autre représente l'itérateur, c'est-à-dire la sous-orbite. Lors d'une réévaluation, l'ordre de parcours des sous-orbites peut être amené à changer en fonction des modifications apportées à la topologie. Cet ordre est donc considéré comme une information instable et ne peut être utilisé pour identifier une sous-orbite. C'est donc un nom persistant qui doit discriminer cette sous-orbite parmi les autres itérées.

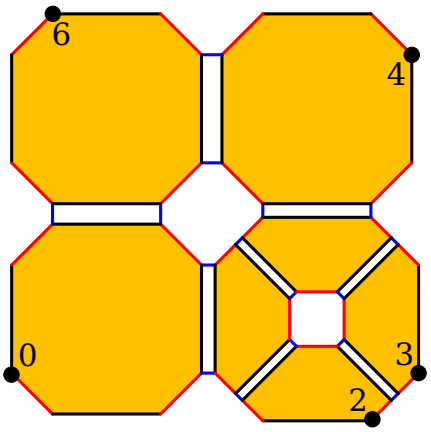


Figure 5.11 – Coloration d'une composante connexe avec triangulation

```
1-creationCarre()
2-subdivisionQuad(PN0=[1n0])
3-S_colorationFaceComposante(PN0=[1n0;2n0])
    For ⟨0,1,3⟩ in ⟨0,1,2,3⟩(PN0=[1n0;2n0])
        1-colorationFace(PN0=[1n0;2n0])
        2-colorationFace(PN2=[1n2;2n0])
        3-colorationFace(PN4=[1n4;2n0])
        4-colorationFace(PN6=[1n6;2n0])
4-triangulationFace(PN3=[1n3;2n0;3{2n0}])
```

Listing 5.7 – Spécification paramétrique avec boucle

Détaillons l'exemple de la spécification paramétrique dans le listing 5.7 de l'objet illustré figure 5.11 . L'application `1-creationCarre` crée une face carrée. Elle crée aussi les brins 0, 2, 3, 4 et 6 dont les historiques sont respectivement $[1n0]$, $[1n2]$, $[1n3]$, $[1n4]$ et $[1n6]$. La face, désignée par PN_0=[1n0], est ensuite subdivisée en plusieurs carrés par l'application `2-subdivisionQuad`. L'application `3-S_colorationFaceComposante` sur la composante connexe désignée par le PN_0=[1n0], déclenche l'enregistrement d'un bloc `For`. Celui-ci contient quatre applications de l'opération `colorationFace`, et plus particulièrement, une pour chaque face. Celles-ci sont désignées par les PN_0, PN_2, PN_4, PN_6. Les noms persistants des brins 0, 2, 3, 4 et 6 deviennent $[1n0; 2n0; 3\{1n0\}]$, $[1n2; 2n0; 3\{2n0\}]$, $[1n3; 2n0; 3\{2n0\}]$, $[1n4; 2n0; 3\{3n0\}]$ et $[1n6; 2n0; 3\{4n0\}]$. Enfin, à nouveau dans le bloc principal de la spécification paramétrique, l'opération `triangulationFace` est appliquée sur la face désignée par le PN_3=[1n3 ;2n0 ;3{ 2n0}].

4.2 DAG

Comme précédemment, nous reportons les nouvelles informations identifiées dans la section précédente dans le DAG d'évaluation. Le bloc de la boucle `For` est conditionné par ses paramètres d'itérations. Cela signifie que deux branches doivent être ajoutées dans l'historique de l'orbite suivie afin de tracer les historiques de ces paramètres : l'une pour l'orbite et l'autre pour la sous-orbite d'itération. Cette branche s'ajoute à celles correspondant respectivement aux paramètres topologiques du bloc de script et de l'application appelée à l'intérieur de la boucle.

Prenons, par exemple, le DAG du PN_3 illustré dans la figure 5.12. Ce DAG, représente l'évo-

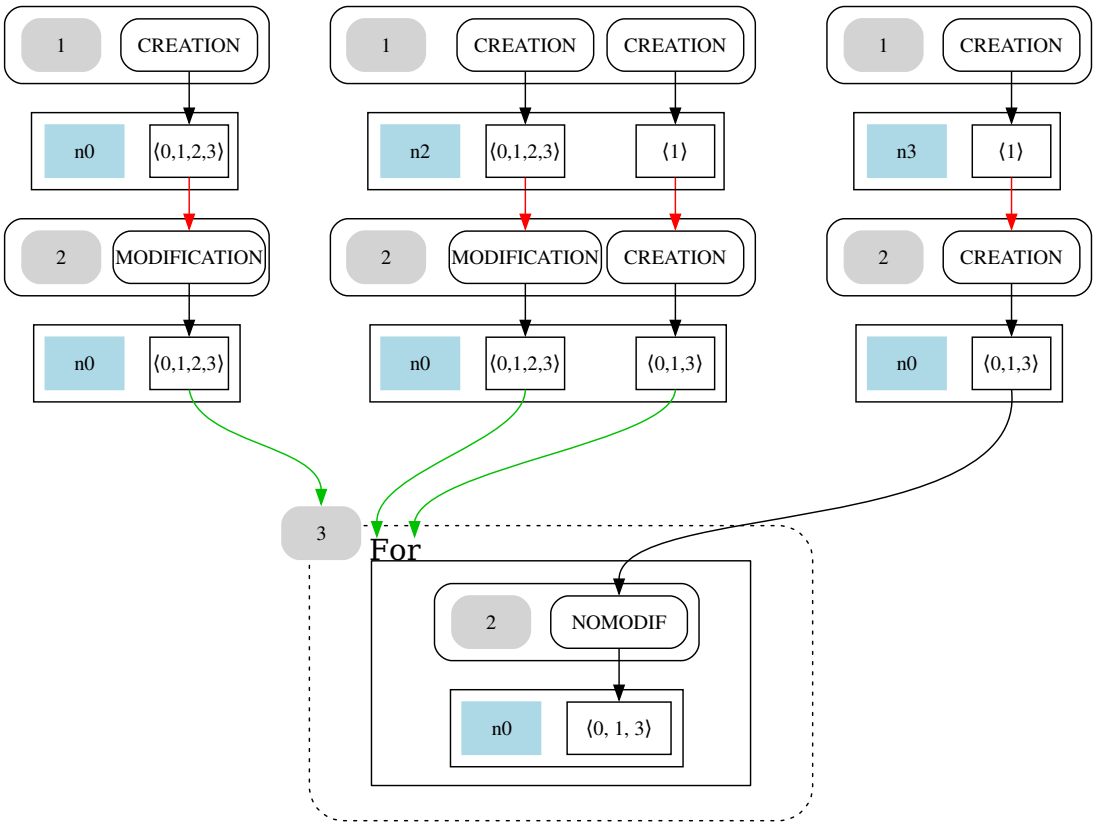


FIGURE 5.12 – DAG d'évaluation du PN_3

lution de l'orbite face ($\langle 0,1,3 \rangle$) qui doit être scindée par l'application `3-triangulationFace`. Nous pouvons observer l'ajout d'un bloc `For` contenant les applications qui ont eu un impact dans l'historique de la face à trianguler. Ce bloc est paramétré par deux orbites dont le suivi est effectué dans deux branches (au milieu). L'une de ces orbites est la composante connexe ($\langle 0,1,2,3 \rangle$) parcourue, l'autre est une instance parmi les sous-orbites d'itérations, c'est-à-dire une face ($\langle 0,1,3 \rangle$). Tout comme pour le bloc de script, les deux nœuds d'orbites de cette nouvelle branche conditionnent la réévaluation de la boucle et plus précisément d'une de ses itérations. Ainsi les nœuds $\boxed{\langle 0,1,2,3 \rangle}$ et $\boxed{\langle 0,1,3 \rangle}$ sont liés par des flèches **vertes** au même bloc `For`.

```
1-creationCarre()
ADD 5-suppressionSommet(3)
2-subdivisionQuad(PN0=[1n0])
3-S_colorationFaceComposante(PN0=[1n0;2n0])
  For ⟨0,1,3⟩ in ⟨0,1,2,3⟩(PN0=[1n0;2n0])
    1-colorationFace(PN0=[1n0;2n0])
    2-colorationFace(PN2=[1n2;2n0])
    3-colorationFace(PN4=[1n4;2n0])
    4-colorationFace(PN6=[1n6;2n0])
4-triangulationFace(PN3=[1n3;2n0;3{2n0}])
```

Listing 5.8 – Spécification paramétrique avec boucle

Étudions maintenant le DAG de réévaluation du PN_3 (figure 5.13) étant donné l'édition de la spécification paramétrique présentée dans le listing 5.8. Dans cette édition, nous ajoutons la règle `suppressionSommet`. Du point de vue du paramètre topologique du script, la composante connexe ($\langle 0,1,2,3 \rangle$) est simplement modifiée. Du point de vue des paramètres de la boucle, c'est

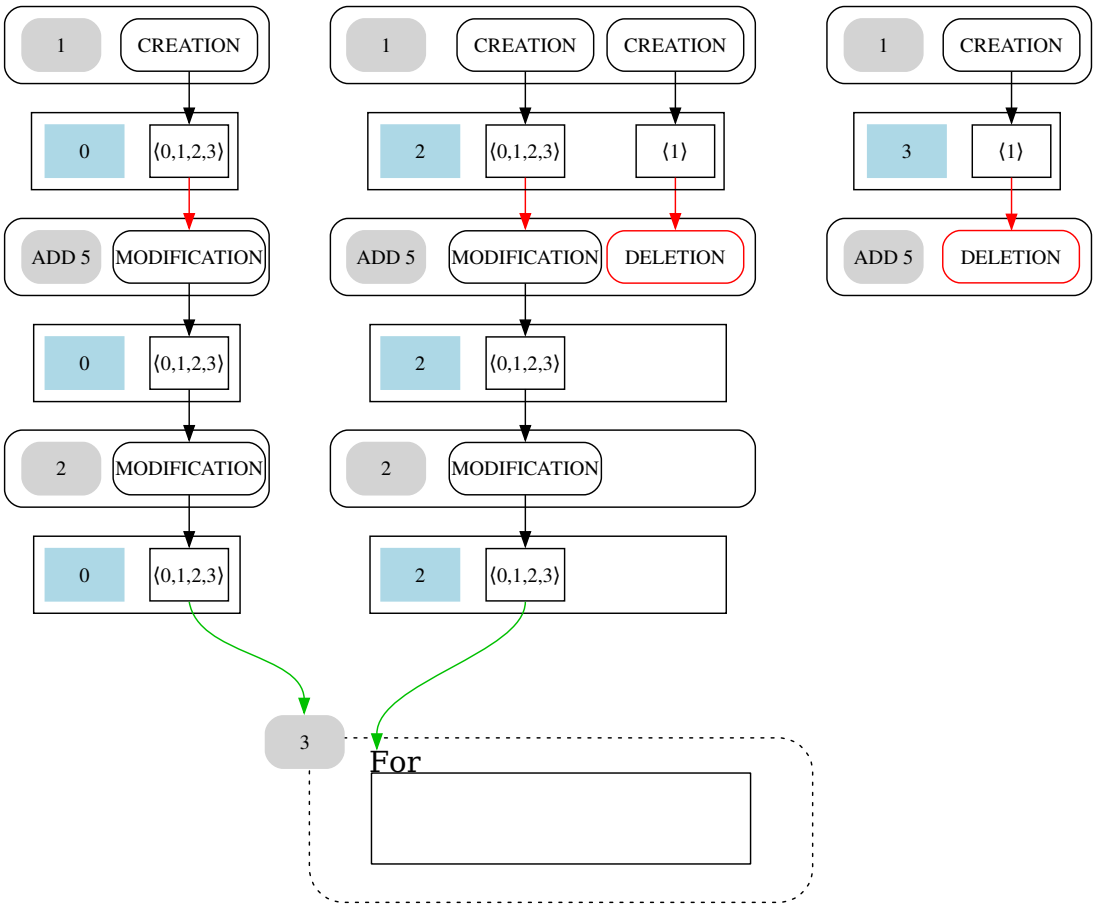


Figure 5.13 – DAG de réévaluation du PN_3

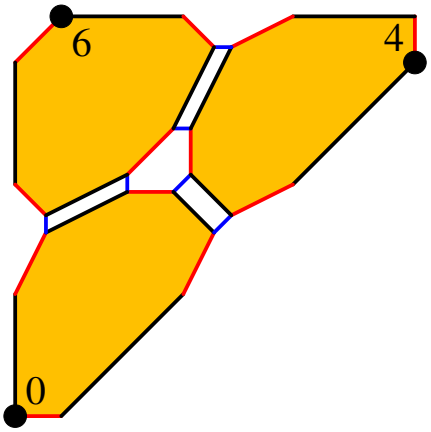


Figure 5.14 – DAG de réévaluation du PN_3

également le cas, à l'exception de la sous-orbite d'itération, pour laquelle nous enregistrons la suppression totale de l'orbite sommet. Étant donné que ce sommet est supprimé, cette itération ne peut être réévaluée. En effet, la suppression du sommet empêche la création d'une face lors de l'application de `subdivisionQuad`. Ainsi, cette face ne peut être ni colorée ni triangulée, et à l'issue de la réévaluation, nous obtenons l'objet illustré dans la figure 5.14.

Dans cet exemple nous avons montré le cas ou une sous-orbite d'itération est supprimée. Évidemment, si l'orbite principale n'est pas recréée, alors la boucle n'est pas réévaluée, ce qui

permet *in fine* d'éviter un mauvais appariement d'orbite. Si c'est une sous-orbite qui est supprimée, comme dans notre exemple, l'application qui la prend en paramètre n'est pas réévaluée. Dans le cas où l'orbite est scindée par une opération ajoutée, alors chacune des instances d'orbites issues de la scission est réévaluée avec cette boucle. Si c'est une sous-orbite qui est scindée, la boucle est naturellement prolongée sur les instances de sous-orbites issues de la scission. Dans le cas où deux orbites sont fusionnées, nous appliquons l'itération sur l'ensemble de la nouvelle orbite. Enfin, si des sous-orbites sont fusionnées, la boucle est réduite aux sous-orbites restantes.

Il ne s'agit là, toutefois, que de stratégies par défaut. Dans le cas d'une scission l'utilisateur pourrait souhaiter ne pas vouloir itérer sur les nouvelles orbites ou sous-orbites. De même, dans une fusion, puisque deux historiques fusionnent, l'utilisateur pourrait souhaiter préserver uniquement l'un des deux historiques et ainsi changer le comportement de la boucle.

5 Structure d'alternatives

La troisième et dernière structure que nous étudions est l'alternative.

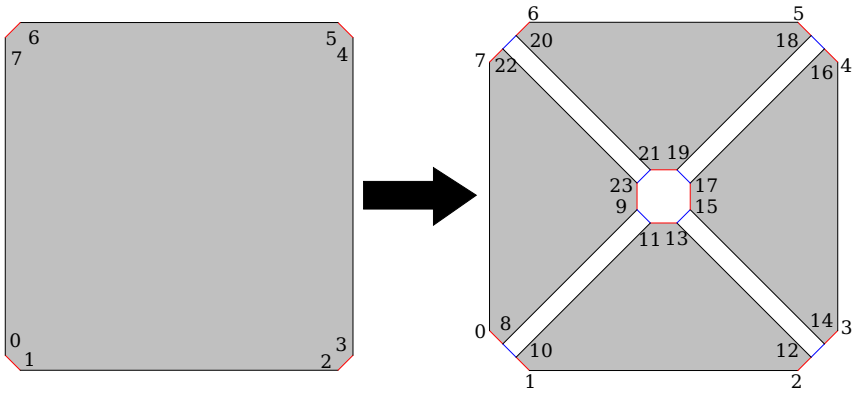
```
if(CONDITION)
{
    @rule<ruleA>(brin);
} else {
    @rule<ruleB>(brin);
}
```

Une structure d'alternative `if-then-else` permet d'appliquer, alternativement, une séquence d'opérations ou une autre en fonction de l'évaluation d'une condition donnée. L'intégration de cette structure pour la réévaluation nécessite, tout comme pour la séquence et la boucle, un enregistrement des opérations évaluées dans la spécification paramétrique ainsi que dans les DAG. Cependant, lors de l'évaluation initiale, une orbite est représentée dans un contexte correspondant uniquement à l'une des deux branches, `Then` ou `Else`. En effet, les règles appelées dans une alternative peuvent être très différentes et, sans vérification supplémentaire, l'intention de conception peut être perdue. Il est alors nécessaire, lorsque cela se justifie, d'effectuer un appariement entre deux règles alternatives.

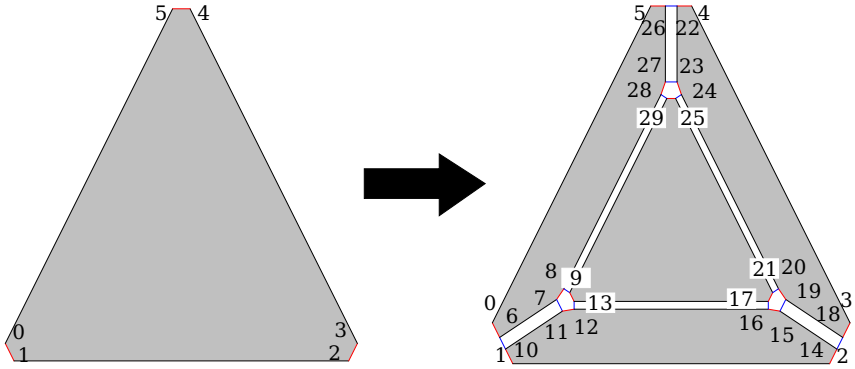
```
int nbSommets = <0,1,3>_<1,3>(face#0).size();
if(nbSommets == 4)
{
    @rule<triangulationFace>(face);
}
else
{
    @rule<insertionFace>(face);
}
```

Listing 5.9 – Script de triangulation ou d'insertion de face dépendante du nombre de sommets

Considérons le script présenté au listing 5.9, qui applique une triangulation sur une face si celle-ci est carrée, ou bien insère une autre face, isomorphe, dans le cas contraire (voir figures 5.15a et 5.15b). Ce script prend en entrée une face ($\langle 0, 1, 3 \rangle$) dont nous comptons le nombre de sommets. Si `face` possède quatre sommets, alors le script appelle la règle `triangulationFace`, sinon elle appelle `insertionFace`.



(a) Triangulation d'un carré



(b) Insertion de face dans un triangle

Figure 5.15 – Alternative sur un nombre de sommets

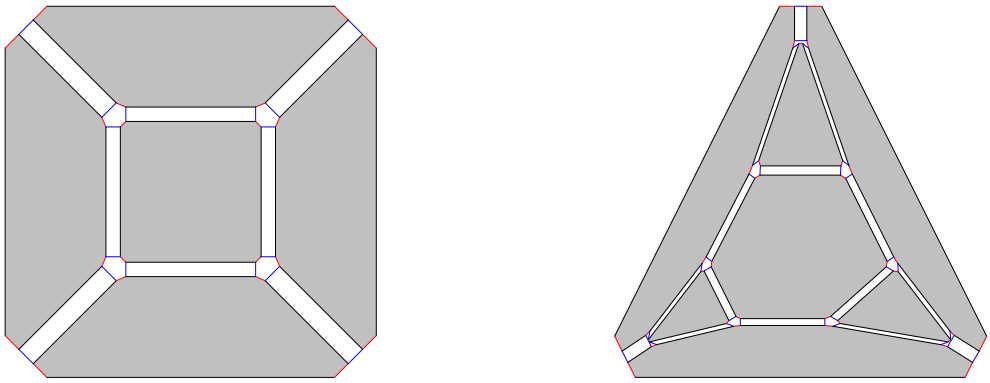
Comme pour les autres structures, l'alternative implique un certain nombre de changements pour les spécifications paramétriques et les DAG. Cependant, cette approche nécessite également la mise en place d'une procédure supplémentaire. En effet, dans une structure alternative, les deux règles appliquées peuvent engendrer des événements très différents sur une même orbite. Par conséquent, il est crucial de mettre en œuvre un mécanisme d'appariement qui respecte l'intention initiale de conception.

5.1 Extension de la spécification paramétrique

Pour supporter la réévaluation des structures d'alternatives, il est nécessaire, une fois encore, d'adapter les spécifications paramétriques. Toutefois, ces modifications sont plus simples que pour les deux autres structures.

```
1 1-creationCarre()
2 2-S_insertionCarreOuTriangulationFace(PN0=[1n0])
3   Alternative - Then:
4     1-triangulationFace(PN0=[1n0])
5 3-chanfreinSommet(PN13=[1n2;2{1n2}])
```

Listing 5.10 – Spécification paramétrique avec alternative



(a) `Then` (b) `Else`

FIGURE 5.16 – Évaluations `Then` et `Else` dans un script

Prenons la spécification initiale donnée dans le listing 5.10. Celle-ci produit l'objet illustré dans la figure 5.16a puisque la face créée par `1-creationCarre` est carrée (l'objet qui aurait été produit avec une face initiale triangulaire est illustré sur la figure 5.16b).

Comme pour les autres structures, une fois dans le script, nous enregistrons un bloc `Alternative` en notant la branche évaluée, c'est-à-dire soit la branche `Then`, soit la branche `Else`. Contrairement aux deux autres structures, nous n'enregistrons pas de paramètre topologique pour le bloc `Alternative`. En effet, la branche évaluée dépend strictement du résultat d'une évaluation booléenne. Si cette évaluation peut porter sur une entité topologique, ce n'est en aucun cas une obligation. Ainsi, seuls les paramètres topologiques du script et de l'opération à l'intérieur de l'alternative sont ajoutés à la spécification paramétrique (dans cet exemple, il s'agit deux fois du même).

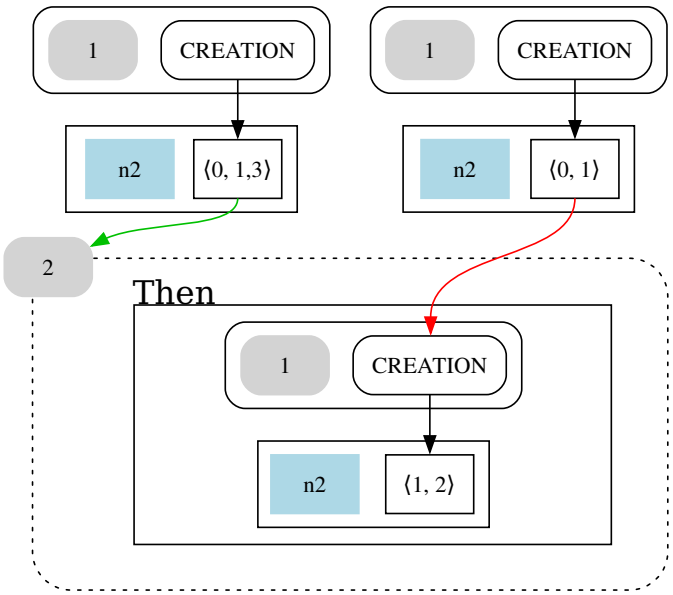


FIGURE 5.17 – DAG d'évaluation du PN_{13}

5.2 DAG

Le DAG d'évaluation présenté figure 5.17 correspond au PN_{13}, le sommet utilisé comme paramètre de l'opération `3-chanfreinSommet`. La branche évaluée étant la branche `Then`, nous ajoutons donc, tout simplement, un bloc appelé `Then` sans paramètre topologique pour conditionner son évaluation. Notons que seule la branche évaluée est tracée, car, logiquement, son alternative n'est pas appliquée.

```
1-creationCarre()
ADD 4-suppressionSommet(5)
2-S_insertionCarreOuTriangulationFace(PN0=[1n0])
  Alternative - Then:
    1-triangulationFace(PN0=[1n0])
3-chanfreinSommet(PN13=[1n2;2{1n2}])
```

Listing 5.11 – Spécification paramétrique avec alternative éditée

Étudions maintenant le DAG de réévaluation du PN_{13} étant donné l'édition de la spécification paramétrique présentée dans le listing 5.11. Dans cette édition, nous ajoutons l'opération `4-suppressionSommet` qui modifie la face carrée en triangle. Avec cet ajout et la modification de topologie provoquée sur la face, le système de réévaluation évalue désormais la branche `Else` au lieu de la branche `Then`. Le DAG de réévaluation du PN_{13} est présenté figure 5.18. Cependant, comme évoqué précédemment, les règles appelées dans ces deux branches pouvant être différentes, il est nécessaire d'effectuer un travail spécifique d'appariement entre ces deux règles afin de pouvoir construire ce DAG. C'est ce travail d'appariement que nous présentons dans la section suivante.

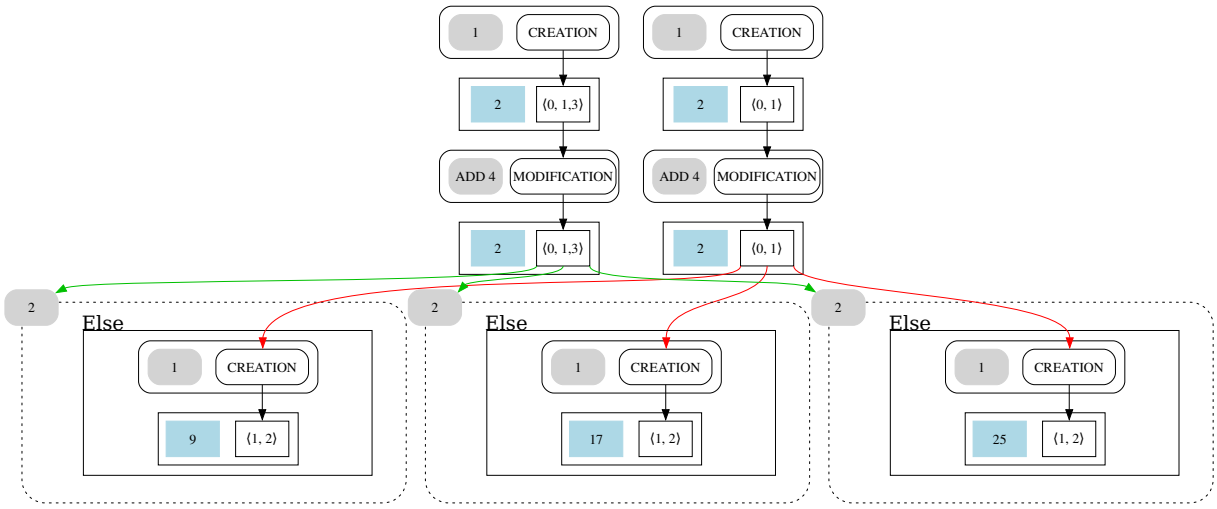
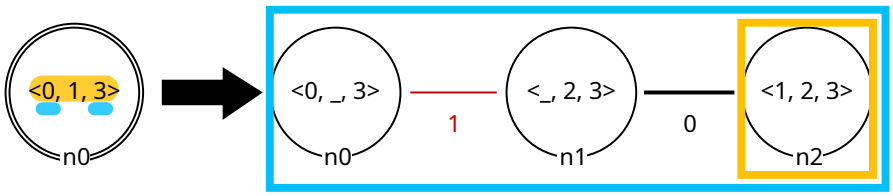


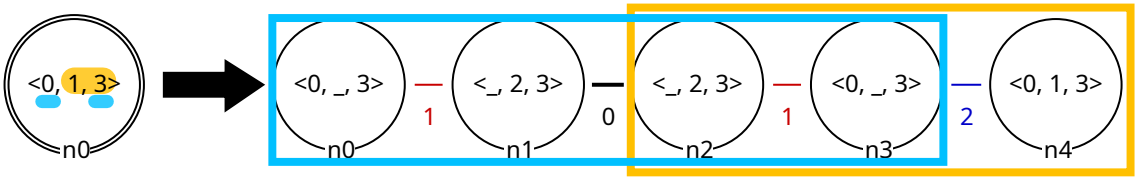
Figure 5.18 – DAG de réévaluation du PN_{13}

5.3 Appariement entre deux règles

L'appariement d'orbites entre deux règles est une procédure particulière puisqu'elle propose d'apparier des orbites qui peuvent évoluer de manières très différentes. À ce titre, nous proposons de définir des catégories d'évènements.



(a) Règle de triangulation



(b) Règle d'insertion de face

FIGURE 5.19 – Règles alternatives du script (listing 5.9)

`GENERATION :` qui comprend l'évènement `CREATION` ;

`DESTRUCTION :` qui comprend l'évènement `DELETION` ;

`OTHER :` qui comprend les évènements de modification de la topologie au sens large, à savoir `SPLIT`, `MERGE`, `NOMODIF`, `MODIFICATION` et `NOEFFECT`.

Étant donné les différences qui peuvent exister entre deux règles, cette classification nous permet d'être moins restrictif dans les appariements potentiels pouvant être considérés.

Les opérations de notre script (listing 5.9), sont illustrées dans les figures 5.19a et 5.19b. Lors de l'évaluation initiale la face passée en entrée du script était un carré, la règle appliquée par le script était la triangulation et le sommet central créé par ce script était utilisé comme paramètre de l'opération de chanfreinage. Dans ce contexte, commençons par étudier l'appariement du sommet créé. Nous présenterons dans un second temps, en guise d'exemple et sur les mêmes règles, l'appariement d'une face scindée.

5.3.1 Appariement d'un sommet créé

L'appariement entre les orbites de deux règles distinctes nécessite, dans un premier temps, de définir le type de suivi et d'origine ainsi que la catégorie d'évènement sur la règle utilisée à l'évaluation. Puis, dans un second temps, d'identifier les orbites éligibles à l'appariement sur la règle utilisée à la réévaluation.

Types de suivi et d'origine et catégorie d'évènement

La première étape pour effectuer l'appariement consiste donc à récupérer l'évènement et l'origine de l'orbite que l'on souhaite apparier. Dans notre exemple, il s'agit du sommet $\langle 1, 2, 3\rangle$($n2$) de la règle de triangulation. L'analyse de la règle (figure 5.19a) montre que ce sommet (encadré en **jaune**) est créé. Son évènement appartient donc à la catégorie d'évènement `GENERATION`. L'analyse montre aussi que son origine est de type face $\langle 0, 1, 3\rangle$ (surligné en **jaune**).

Appariement des orbites

La deuxième étape est l'identification des orbites éligibles pour l'appariement. Pour cela, nous cherchons dans le membre droit d'`insertionFace` les orbites de type sommet ($\langle 1,2,3\rangle$) qui partagent la même catégorie d'évènement (c'est-à-dire, un sommet créé). Dans notre exemple, nous ne trouvons qu'une seule orbite sommet créée dans la règle `insertionFace`, c'est l'orbite $R\langle 1,2,3\rangle(n2)$ (encadrée en **jaune** dans la figure 5.19b).

Cependant, l'origine de cette orbite n'est pas identique à celle du sommet créé par la règle `triangulationFace`. En effet, l'orbite $R\langle 1,2,3\rangle(n2)$ de `insertionFace` a pour origine une orbite de type $\langle 1,3\rangle$, c'est-à-dire une sous-orbite de $\langle 0,1,3\rangle$. Nous proposons alors deux stratégies d'appariement basées sur les types d'origines. La première consiste à rechercher un appariement exact et, dans ce cas, les deux types d'origines doivent être exactement identiques. La seconde consiste en un appariement moins strict autorisant des sur ou sous-orbites pour origine.

Pour cette deuxième stratégie, il faut considérer que l'appariement entre une orbite et une sous-orbite n'est pas forcément `1:1`, cette dernière pouvant correspondre à plusieurs instances dans l'objet. Dans le cas d'un appariement avec une sur-orbite, l'orbite initiale, ainsi que plusieurs autres, peuvent être comprises dans la sur-orbite trouvée.

Dans notre exemple, il n'existe pas d'appariement exact. Nous poursuivons donc avec un appariement moins strict, c'est-à-dire l'appariement d'un sommet d'origine de type $\langle 1,2,3\rangle$ avec les sommets d'origine de type $\langle 1,3\rangle$. Nous obtenons ainsi les trois sommets créés ($\langle 1,2,3\rangle(9)$, $\langle 1,2,3\rangle(17)$, $\langle 1,2,3\rangle(25)$), apparaissant dans le DAG réévalué (figure 5.18) et instanciés par le noeud $n2$ de la règle d'insertion de face (figure 5.19b).

5.3.2 Appariement d'une face scindée

Nous présentons maintenant, en guise d'exemple, l'appariement d'une face scindée en nous basant sur les mêmes règles et les mêmes objets.

Types de suivi et d'origine et catégorie d'évènement

Pour la première étape, nous analysons à nouveau la règle `triangulationFace`. Nous trouvons que l'orbite $L\langle 0,1,3\rangle(n0)$ (encadrée en **bleu** dans la figure 5.19a) est scindée et est donc associée à l'évènement `SPLIT` qui appartient à la catégorie d'évènements `OTHER`. Elle a pour origine une $\langle 0,3\rangle$-orbite (soulignée en **bleu**).

Appariement des orbites

L'analyse de la règle `insertionFace` montre ici aussi qu'il n'y a qu'une seule orbite éligible pour l'appariement. Il s'agit de l'orbite $R\langle 0,1,3\rangle(n0)$ (encadrée en **bleu** dans la figure 5.19b). Elle est aussi scindée. Cette fois, elle a aussi pour origine une $\langle 0,3\rangle$-orbite. Il est donc possible de procéder à un appariement exact entre les deux règles. Prenons l'exemple de l'opération `3-chanfreinSommet` des spécifications initiale et réévaluée. Nous la remplaçons par l'opération `3-colorationFace` que nous appliquons sur la face $\langle 0,1,3\rangle(0)$ (figure 5.15b) instanciée par le

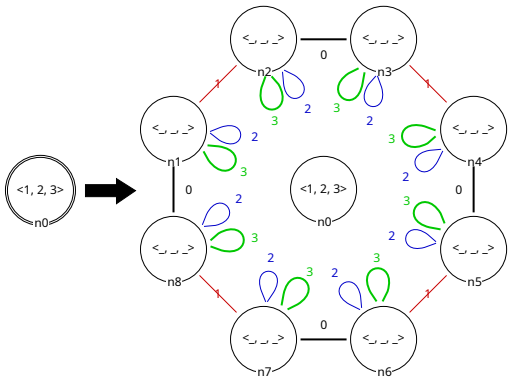


FIGURE 5.20 – Règle de création d'une face carrée duale d'un sommet

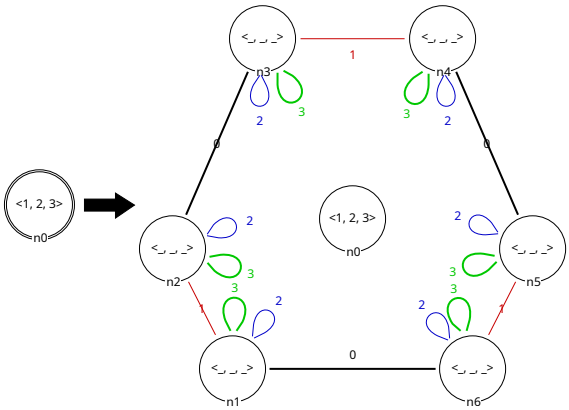


FIGURE 5.21 – Règle de création d'une face triangulaire duale d'un sommet

nœud $n0$ de la règle de triangulation. Alors, au rejeu, cette face est finalement appariée avec la face $\langle 0, 1, 3 \rangle(0)$ suite à l'insertion de face (figure 5.15b).

6 Exemple d'imbrication de structures de contrôle

Comme dans tout langage de programmation, les structures de contrôles que nous venons d'étudier doivent pouvoir être imbriquées. Les extensions de DAG et de spécification paramétrique proposées permettent cette imbrication. Dans cette section, nous présentons un exemple de structure `if-then-else` imbriquée dans un `foreach`.

Pour cet exemple, nous présentons l'extrusion d'une séquence d'arêtes à partir d'un brin permettant de construire un squelette. Pour chaque sommet de ce squelette, nous souhaitons créer une face dont la topologie correspond à l'arité d'un sommet. Pour garder cet exemple simple, nous posons une restriction sur le voisinage d'un sommet : ceux-ci ne peuvent avoir plus de trois voisins. Ainsi, les faces générées sont uniquement des carrés et des triangles. Enfin, nous ne traitons pas non plus leurs coutures.

Étudions la spécification paramétrique, détaillée dans le listing 5.12, correspondant à la construction de l'objet illustré dans la figure 5.22. La première opération enregistrée est la création d'un brin. Ce brin est ensuite désigné comme paramètre topologique pour l'opération suivante. La deuxième opération est un script, contenant une séquence, similaire à celui étudié dans

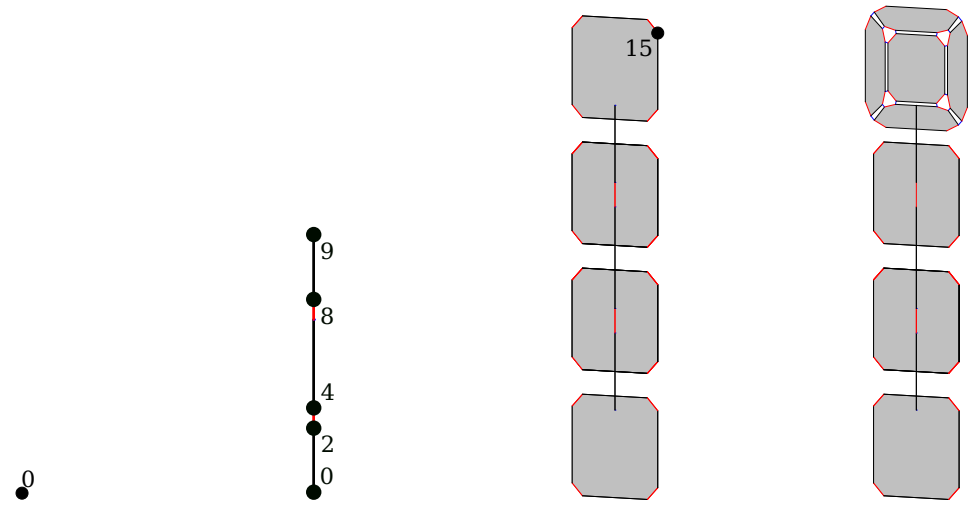


Figure 5.22 – Évaluation initiale de la création du dual d'un squelette 1D

la section 3. Ce script, au lieu de créer un ruban à partir de l'extrusion d'une face, crée un squelette à partir de l'extrusion d'un sommet. La troisième opération est un script qui contient une boucle avec une alternative imbriquée. Cette opération crée une face pour chacun des sommets du squelette. Cette face est triangulaire si le sommet a trois voisins et carrée sinon. Enfin, la quatrième opération insère une face dans la face désignée par le brin 15.

```
1-creationSommet()
2-S_extrusionSquelette(PN0=[1n0])
  1-extrusionSommet(PN0=[1n0])
  2-extrusionSommetArete(PN2=[1n0;2{1n1}])
  3-extrusionSommetArete(PN5=[1n0;2{1n1;2n2}])
3-S_creationFacesDualesStruct(PN9=[1n0;2{1n1;2n2;3n2}])
  For ⟨1,2,3⟩ in ⟨0,1,2,3⟩ :
    Alternative - Then :
      1-creationFaceDualeCarree(PN9=[1n0;2{1n1;2n2;3n2}])
    Alternative - Then :
      2-creationFaceDualeCarree(PN8=[1n0;2{1n1;2n2;3n1}])
    Alternative - Then :
      3-creationFaceDualeCarree(PN4=[1n0;2{1n1;2n1}])
    Alternative - Then :
      4-creationFaceDualeCarree(PN0=[1n0;2{1n1;2n1}])
4-insertionFace(PN15=[1n0;2{1n1;2n2;3n2};3{1n4}])
```

Listing 5.12 – Spécification paramétrique initiale de la création du dual d'un chemin 1D

Cette spécification paramétrique contient deux scripts qui construisent le squelette et la face désignée par le brin 15 dans l'application `4-insertionFace`. Étudions le DAG d'évaluation de ce brin illustré dans la figure 5.23. Ce DAG retrace l'histoire d'une face ($\langle 0,1,3\rangle$) créée par le script `3-S_creationFacesDualesStruct`.

Cette face est créée à l'issue d'une itération de la boucle par l'opération `1-creationFaceDualeCarre`. Nous pouvons observer que le nœud d'orbite $\boxed{\langle 0,1,3\rangle}$, est intégré dans un ensemble de structures imbriquées. Comme décrit dans la spécification paramétrique, l'alternative `Then` est imbriquée dans un `For`, lui-même à l'intérieur du bloc correspondant à l'application `3-S_creationFacesDualesStruct`.

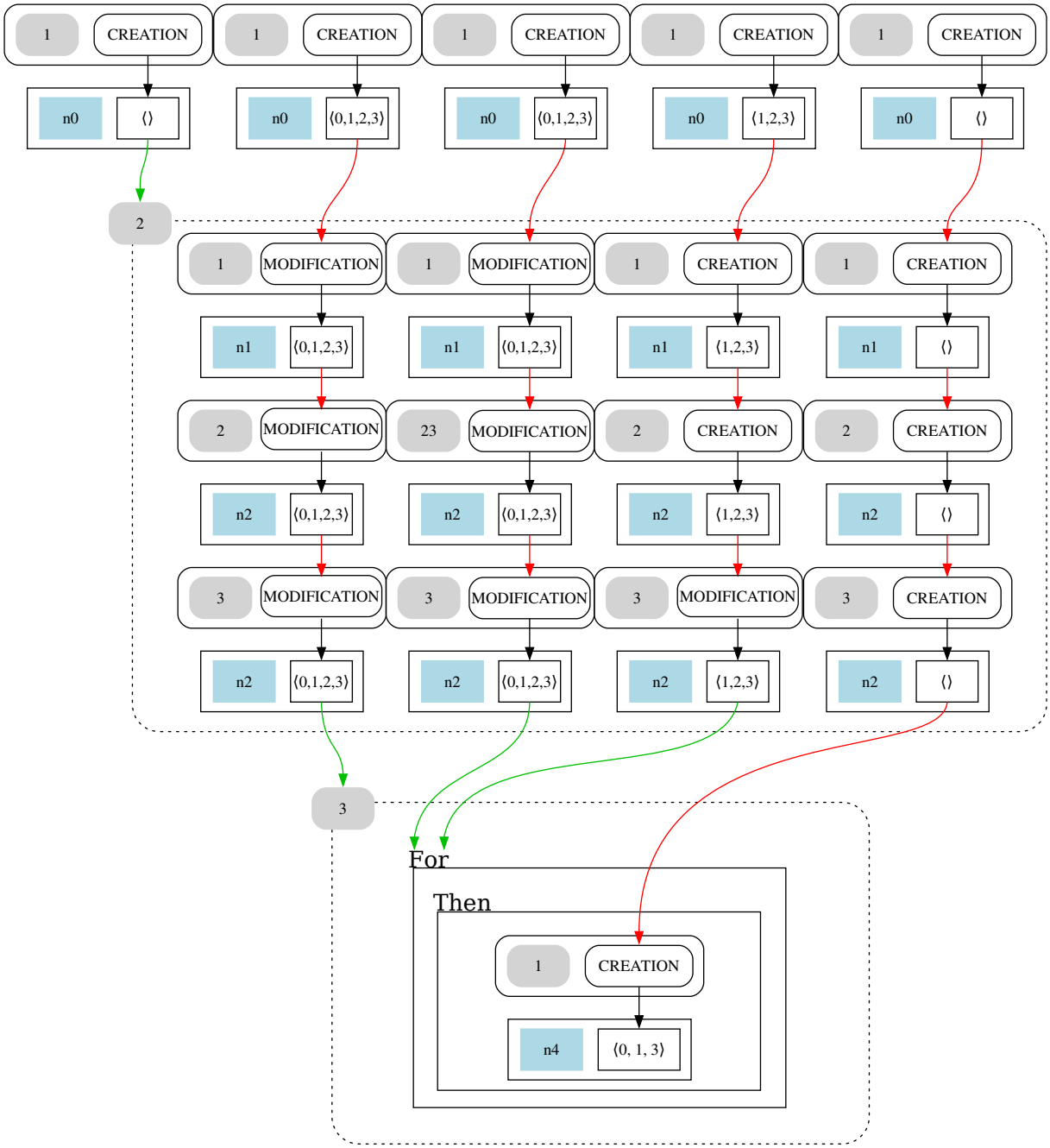


FIGURE 5.23 – DAG d'évaluation du PN_{15}

Le DAG d'évaluation du PN_{15} contient alors quatre branches permettant de tracer quatre orbites. La première correspond à l'origine de la face désignée par le brin 15 (flèche rouge). La deuxième correspond au paramètre topologique du script (la composante connexe représentant le squelette - flèche verte de gauche). La troisième et la quatrième correspondent aux deux paramètres de la boucle (la composante connexe représentant le squelette et un sommet - deux flèches vertes centrales).

Les évolutions de ces orbites sont ensuite tracées au travers du bloc correspondant au script `2-S_extrusionSquelette`. Cela induit l'ajout d'une cinquième branche dans le DAG permettant de tracer l'historique du paramètre de ce script (flèche verte). Chacune des règles appelées dans ce script modifie la composante connexe correspondant au squelette. Ces règles créent deux sommets dans l'historique du sommet désigné comme paramètre du `For` ainsi que deux brins dans l'historique du sommet désigné comme paramètre de l'application `4-insertionFace`.

Une fois l'historique de chaque orbite entièrement tracé, nous obtenons ainsi, dans l'ordre de gauche à droite, cinq branches conditionnant la réévaluation du script `2-S_extrusionSquelette`, du script `3-S_creationFacesDualesStruct`, de la structure `For` et de la règle `creationFaceDualeCarre` dans la structure `Then`.

Étudions maintenant la réévaluation de cette spécification paramétrique suivant l'édition présentée dans le listing 5.13. Cette édition propose simplement d'ajouter deux applications. D'abord `ADD 5-extrusionSommetDoubleArete`, qui ajoute deux arêtes en forme d'embranchement au squelette à partir du sommet désigné par le brin 9, et `ADD 6extrusionAreteSommet` qui extrude une arête à partir du sommet désigné par le brin 14.

```
1-creationSommet()
2-S_extrusionSquelette(PN0=[1n0])
  1-extrusionSommet(PN0=[1n0])
  2-extrusionSommetArete(PN2=[1n0;2{1n1}])
  3-extrusionSommetArete(PN5=[1n0;2{1n1;2n2}])
ADD 5-extrusionSommetDoubleArete(9)
ADD 6-extrusionSommetArete(14)
3-S_creationFacesDualesStruct(PN9=[1n0;2{1n1;2n2;3n2}])
  For ⟨1,2,3⟩ in ⟨0,1,2,3⟩ :
    Alternative - Then :
      1-creationFaceDualeCarree(PN9=[1n0;2{1n1;2n2;3n2}])
    Alternative - Then :
      2-creationFaceDualeCarree(PN8=[1n0;2{1n1;2n2;3n1}])
    Alternative - Then :
      3-creationFaceDualeCarree(PN4=[1n0;2{1n1;2n1}])
    Alternative - Then :
      4-creationFaceDualeCarree(PN0=[1n0;2{1n1;2n1}])
4-insertionFace(PN15=[1n0;2{1n1;2n2;3n2};3{1n4}])
```

Listing 5.13 – Spécification paramétrique initiale de la création du dual d'un squelette 1D

Dans le DAG de réévaluation du PN_{15} (figure 5.24), nous pouvons voir, d'une part, que les applications ajoutées sont bien enregistrées dans le DAG et, d'autre part, que la branche `Else` a été évaluée. Cette différence est due à l'application `ADD 5-extrusionSommetDoubleArete` qui modifie l'arité du sommet 9 et induit un changement dans l'évaluation du bloc d'alternative dans le script `3-S_creationFacesDualesStruct`. En effet, le sommet ayant maintenant trois voisins, la règle `creationFaceDualeTriangle` est dorénavant appelée dans le script. Ainsi, la face créée à partir du sommet $\langle 1, 2, 3\rangle(9)$ est un triangle et l'appariement entre les deux règles

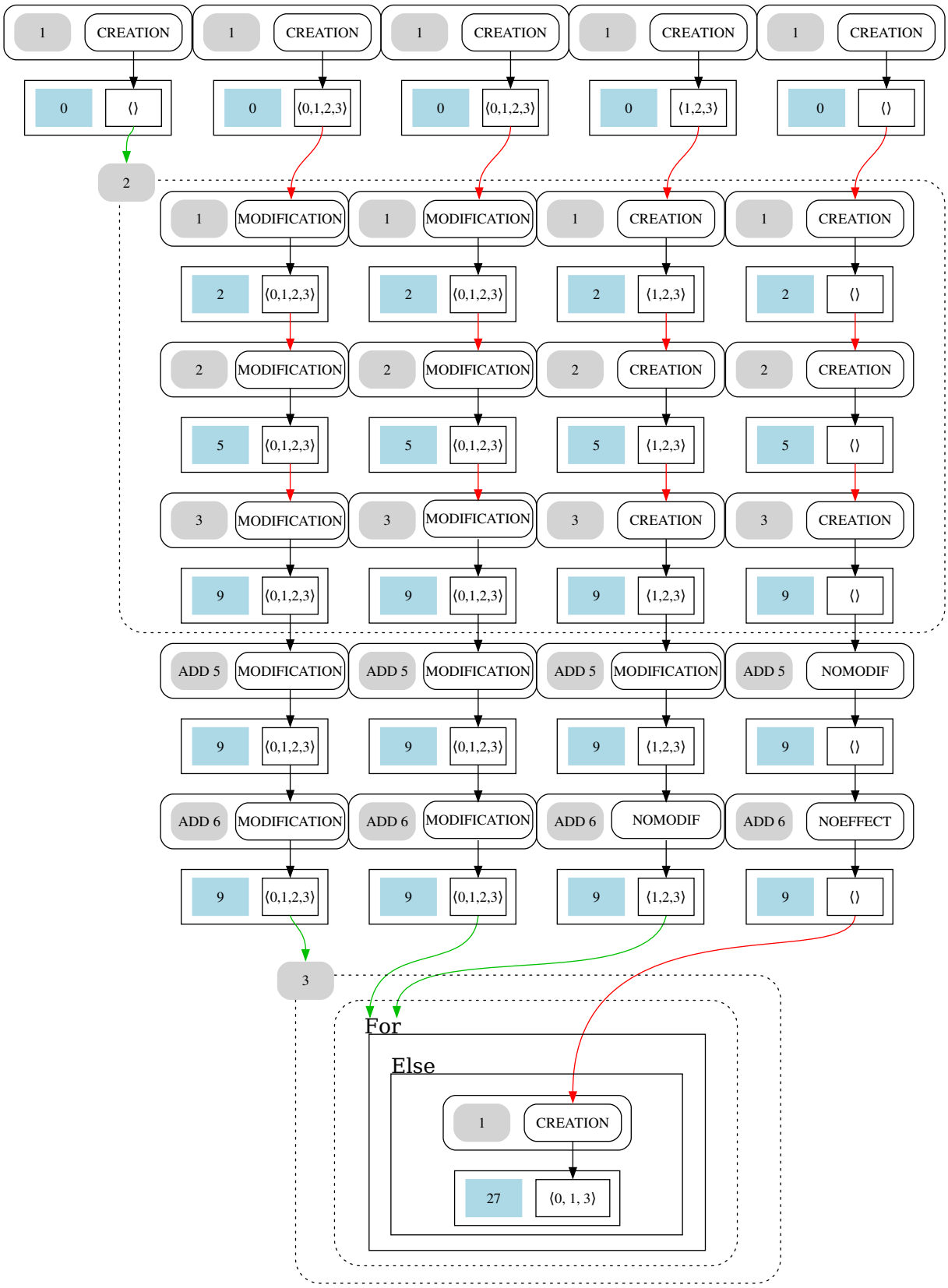


FIGURE 5.24 – DAG de réévaluation du PN_{15}

(figure 5.20 et figure 5.21) du brin créé par le nœud $n4$ nous donne le brin 27.

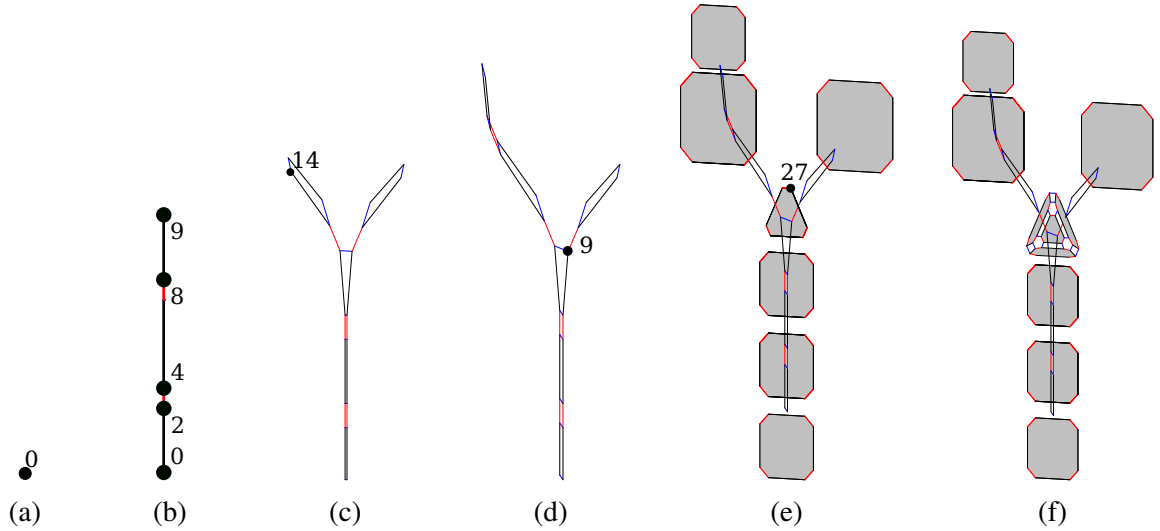


Figure 5.25 – Réévaluation de la création du dual d'un chemin 1D

À l'issue de la réévaluation, nous obtenons l'objet illustré dans la figure 5.25f. Les deux premières étapes sont bien reproduites à l'identique. La première application ajoutée crée un embranchement à partir du sommet $\langle 1, 2, 3\rangle(9)$. La seconde, `ADD 6-extrusionSommetArete`, crée une nouvelle arête à partir du sommet $\langle 1, 2, 3\rangle(14)$. Étant donné ce nouveau contexte, l'évaluation conditionnelle contenue dans le script `3-S_creationFacesDualesStruct` fait appel à une règle différente de celle utilisée lors de l'évaluation initiale. C'est donc une face triangulaire, et non carrée, qui est créée à partir du sommet incident au brin 9. L'appariement obtenu à partir du DAG de PN_{15} permet alors d'insérer une face isomorphe à la face triangulaire en rejouant correctement la dernière opération `4-insertionFace` et d'obtenir ainsi l'objet final.

7 Conclusion

Dans ce chapitre, nous avons proposé d'étendre notre mécanisme de réévaluation afin de supporter les opérations construites à l'aide du langage de scripts de Jerboa. Nous avons étudié plus particulièrement trois structures de contrôle : les séquences, les boucles `foreach` et les alternatives `if-then-else`. Une première problématique que nous avons abordée concerne le mode de représentation des scripts pour déterminer comment traiter leurs contenus. La première option concernait la représentation boîte fermée. Dans ce cas, un script ne nécessite pas de traitement particulier et, en contrepartie, les développeurs de scripts doivent construire une sortie à la manière des règles Jerboa. La seconde option, que nous avons choisie, est la boîte ouverte. Dans ce mode de représentation, nous avons connaissance du contenu des scripts, à savoir les règles appelées et leurs paramètres ainsi que les structures de contrôle utilisées.

Dans cette démarche, nous avons proposé d'étendre nos structures de spécification paramétrique et de DAG. L'extension des spécifications paramétriques consiste à représenter un script comme une spécification imbriquée et à lister son contenu, c'est-à-dire les opérations appliquées et les structures de contrôles utilisées. Pour les DAG, nous considérons qu'il est nécessaire d'ajouter une branche pour chacun des paramètres qui conditionnent la réévaluation d'une orbite.

Ainsi, lorsque l'évolution d'une orbite est impactée par un script, celle-ci est aussi conditionnée par l'existence du paramètre topologique de ce script. De même, nous considérons qu'une boucle `foreach` dispose de deux paramètres topologiques, à savoir l'orbite à itérer et l'instance de sous-orbite d'itération que nous souhaitons apparier.

Enfin, nous avons proposé une dernière contribution relative à la structure d'alternative `if-then-else`. Cette structure permet, étant donné l'évaluation d'une condition, d'appliquer une séquence d'instruction ou une autre. Dans le cas où une autre branche que celle évaluée initialement serait appliquée, il est nécessaire d'identifier précisément les orbites sur lesquelles les utilisateurs souhaitent travailler. En effet, deux orbites appelées de la sorte peuvent être très différentes. Pour respecter les intentions de modélisation, nous mettons en œuvre une procédure basée sur l'analyse des règles alternatives. Cette procédure effectue l'appariement des orbites en fonction des transformations similaires qu'elles subissent. Enfin, puisque ces appariements ne sont pas explicitement exprimés par les utilisateurs, nous proposons la sélection de stratégies pour configurer la qualité d'appariement recherchée et permettre de les adapter à différents contextes d'application.

Les implantations des contributions présentées dans ce chapitre sont des travaux en cours. Au moment de la rédaction de ce manuscrit, seules la procédure d'appariement des orbites et les stratégies de degré d'exactitude détaillées dans la section 5.3.1 ont été implantées. Les extensions de spécification paramétrique et de DAG relatives à chacune des structures de contrôles ne sont, quant à elles, pas encore implantées dans notre modeleur.

Conclusion

Contributions

La réévaluation de modèles paramétriques est une fonctionnalité incontournable des modeleurs interactifs. Ceux-ci permettent en effet d'éviter de refaire le travail fastidieux de conception d'un modèle pour en générer une simple variation. Cela s'avère particulièrement avantageux lorsque le processus de conception ne change pas de manière substantielle ; par exemple, avec la modification des paramètres géométriques des opérations qui n'implique aucun ajout, suppression ou déplacement d'opération.

Afin d'être considéré fonctionnel, un mécanisme de réévaluation doit respecter un certain nombre de contraintes. Le mécanisme de nomination doit permettre l'identification de manière unique et non ambiguë des entités topologiques. L'objet issu du processus de réévaluation doit respecter les intentions de modélisation de l'utilisateur, c'est-à-dire que le programme ne doit pas prendre d'initiative à la place de l'utilisateur. L'objet doit être topologiquement et géométriquement cohérent.

Dans les solutions existantes, nous avons identifié plusieurs faiblesses liées à ces différentes contraintes. Premièrement, si les systèmes de nomination permettent déjà d'identifier des entités topologiques avec une certaine fidélité, ceux-ci ne sont pas génériques. Leurs schémas de nomination sont adaptés à une dimension précise de modélisation. Les étendre à d'autres dimensions nécessite alors des efforts importants ainsi que la définition de nouvelles procédures d'appariement. Deuxièmement, le recours à la géométrie de certaines méthodes introduit des ambiguïtés lors de l'appariement. Ces ambiguïtés entraînent elles-mêmes des erreurs d'appariement et, par conséquent, le non-respect des intentions de modélisation exprimées par un utilisateur. Enfin, ces mêmes erreurs d'appariement peuvent produire des objets qui ne sont pas cohérents du point de vue topologique. Il est donc nécessaire de contrôler et, le cas échéant, de réparer l'objet produit.

Détection automatique des évènements

Notre première contribution concerne la détection des évènements (création, suppression, scission, fusion, non-changement et modification). Nous avons formalisé un ensemble de critères permettant de caractériser chacun des évènements aussi bien dans les objets que dans les règles de transformation de graphe.

Nous avons ensuite étendu ce formalisme au langage de règle Jerboa qui permet de représenter des opérations de modélisation géométrique. De plus, l'analyse étant effectuée uniquement à partir des règles Jerboa, les évènements sont déterminés indépendamment des objets sur lesquels les règles sont appliquées. Nous décrivons aussi les limites des détections liées à ce formalisme.

Dans de tels cas, une simple vérification locale, dans l'objet, permet de confirmer ou non la détection d'un évènement.

Avec ce formalisme, nous proposons une nouvelle méthode permettant de suivre l'évolution des entités topologiques dans un modèle, exclusivement grâce à une analyse syntaxique des règles Jerboa. Celle-ci peut être opérée statiquement en pré-calcul au premier lancement du modeleur, ce qui s'avère particulièrement efficace en temps de calcul. Ainsi, cette méthode s'affranchit de la nécessité de parcourir en parallèle deux objets pour les comparer. De même, il devient inutile de spécifier explicitement quels évènements sont attendus suite à l'application d'une opération.

Enfin, cette méthode de détection est générique. Celle-ci étant basée sur l'analyse des règles, elle n'est pas conçue pour une dimension spécifique. Elle permet alors la détection des évènements quelle que soit la dimension de modélisation du modeleur ou des orbites manipulées par l'opération.

Réévaluation dans un système de modélisation à base de règles

À l'aide des formalismes des cartes généralisée et des règles Jerboa enrichis de notre formalisation des évènements, nous avons implanté un mécanisme de nomination persistante. Nous décrivons ce mécanisme en deux couches.

La première couche consiste en la construction puis la maintenance d'un historique d'opérations et de nœuds de règles pour chacun des brins d'une carte généralisée. Grâce à cela, tout brin peut être identifié de manière unique et non ambiguë. Ainsi, à chaque fois qu'une règle est appliquée, l'historique du brin sélectionné à l'instance courante de modélisation est enregistré dans une spécification paramétrique. Cet historique ainsi enregistré est immuable et devient alors un nom persistant de brin.

Dans la seconde couche, nous avons implanté une procédure qui, pour chaque nom persistant enregistré au cours de la modélisation, permet la reconstitution de l'historique des évolutions d'une orbite incidente au brin désigné. Un tel historique est représenté par un DAG dit d'évaluation nous permettant de traiter l'information qu'il contient. Dans cette procédure, nous utilisons la formalisation des évènements pour détecter et enregistrer dans le DAG chacun des évènements impactant une orbite au cours de la modélisation. De plus, cet historique est enrichi de la notion d'origine qui nous permet alors d'identifier sans ambiguïté une orbite lors de la réévaluation sur la base de son historique de construction. Ces origines sont elles-mêmes tracées jusqu'à représenter leurs ancêtres les plus anciens.

Pour la réévaluation à proprement parler, nous avons implanté une procédure d'appariement produisant des DAG de réévaluation. Ceux-ci sont créés à partir des DAG d'évaluation, puis mis à jour en tenant compte de l'édition de la spécification paramétrique. De cette manière, nous pouvons conserver à la fois les DAG d'évaluation et de réévaluation. Les DAG de réévaluation permettent la mise en œuvre de procédures d'appariement au fur et à mesure de leur construction. Ainsi, une fois le DAG complété, celui-ci désigne les brins à utiliser comme paramètres topologiques pour l'application suivante.

Enfin, nous proposons de prendre en compte l'ajout, la suppression et le déplacement d'opérations. Ce type de modification au niveau de l'historique de construction a un impact plus important sur la réévaluation et les mécanismes de nomination persistante que la simple édition des

paramètres géométriques, qui s'avère restrictive. En complément, nous proposons des stratégies paramétrables pour adapter la réévaluation. Par exemple, lorsqu'une orbite est scindée par une application ajoutée dans la spécification réévaluée, il est possible de choisir les différents paramètres topologiques sur lesquels poursuivre la réévaluation. Au contraire, lorsque deux orbites sont fusionnées par la suppression d'une sous-orbite, nous proposons de choisir de préserver l'une ou l'autre ou les deux historiques de ces orbites dans celle fusionnée. Actuellement, seuls l'ajout et la suppression d'opération sont supportés par le mécanisme de réévaluation.

Extension aux scripts

Les règles Jerboa ne permettent pas de représenter des opérations complexes, c'est-à-dire faisant appels à plusieurs orbites non isomorphes. Ces opérations complexes sont réalisées par les scripts Jerboa qui composent plusieurs règles. Nous proposons une extension de notre mécanisme de réévaluation afin de supporter ces scripts et donc des opérations complexes. Nous définissons les extensions nécessaires à nos structures de données pour trois des structures de contrôles usuelles : la séquence, l'itération de type `foreach` et l'alternative.

Les scripts représentent, d'une certaine façon, des spécifications paramétriques à part entière. Dans notre démarche, nous avons fait le choix de la représentation en « boîte ouverte ». En effet, contrairement à la « boîte fermée » qui permet seulement de gérer un script comme une règle, la boîte ouverte nous permet d'accéder au contenu des scripts, à savoir les opérations, les paramètres topologiques et géométriques ainsi que les structures de contrôle utilisées.

Nous proposons ainsi, sur la base de la représentation boîte ouverte, des extensions de structures de spécification paramétrique et de DAG d'évaluation et de réévaluation. Dans les spécifications paramétriques, l'extension doit nous permettre d'enregistrer des paramètres topologiques à la fois pour les scripts et pour les règles qu'ils appellent. Il est également nécessaire d'enregistrer les structures de contrôles employées afin de représenter fidèlement le processus constructif d'un objet. Dans les DAG, nous réutilisons les informations de scripts ajoutées dans la spécification paramétriques. Ainsi, les opérations appartenant à un script sont englobées dans celui-ci. De plus, nous proposons de conditionner la réévaluation d'une orbite en fonction des paramètres topologiques des scripts qui impactent son évolution ainsi qu'à leurs structures de contrôles. De cette manière, une structure itérative est conditionnée par l'existence de l'orbite itérée ainsi que celle de ses instances de sous-orbites d'itération.

Enfin, nous avons implanté une nouvelle procédure d'appariement d'orbite dans le cadre de l'alternative. Étant donné que deux règles peuvent être très différentes, nous proposons une procédure capable d'apparier des orbites initialement évaluées à celles réévaluées sur la base de leurs origines et de leurs types d'évolutions respectives. Cette procédure d'appariement est, elle aussi, configurable par le moyen d'une stratégie de qualité d'appariement. Nous offrons ainsi la possibilité de rechercher un appariement exact ou moins strict dans la règle alternative.

Perspectives

Certains des travaux présentés dans ce manuscrit étant en cours d'implantation, nous disposons déjà d'un certain nombre de perspectives à court terme.

Tout d'abord, nous prévoyons de terminer l'implantation des déplacements d'opérations dans une spécification paramétrique et ainsi compléter la liste des éditions que nous proposons pour les spécifications paramétriques. Nous compléterons, dans le même temps, les stratégies de réévaluation lors de l'ajout d'une suppression d'orbite, de sorte à gérer la fusion de deux orbites.

Ensuite, nous poursuivrons l'extension de notre mécanisme de réévaluation pour permettre la réévaluation de spécifications paramétriques comportant des scripts Jerboa. Nous implanterons les extensions identifiées dans le dernier chapitre pour les spécifications paramétriques ainsi que les DAG. Nous mettrons en œuvre des tests afin de contrôler le bon fonctionnement des procédures de réévaluation. Puis, nous étendrons encore nos scripts avec des fonctionnalités telles que des boucles itérant sur autre chose que des sous-orbites.

À moyen terme, nous prévoyons d'étendre l'interface graphique de notre modeleur pour y intégrer des stratégies de réévaluation. L'objectif est de permettre aux utilisateurs de configurer leurs modeleurs avec différentes stratégies, d'en choisir une par défaut ou d'interagir dynamiquement avec le modeleur au cours de la réévaluation. Cette extension concerne les stratégies présentées dans ce manuscrit.

Pendant la réalisation de ces travaux, nous avons identifié plusieurs perspectives à plus long terme. Celles-ci nous permettent d'ouvrir nos travaux à d'autres champs d'études.

La première piste concerne le domaine de l'archéologie. L'objectif est de développer des opérations complexes à l'aide de scripts pour la modélisation en archéologie. En effet, dans le cadre de leur recherche, les archéologues peuvent être amenés à créer des restitutions virtuelles d'artefacts archéologiques ou des bâtiments historiques, issus de fouilles et dans des états souvent dégradés. Afin de mener à bien ces travaux, ils émettent des hypothèses sur l'état initial de leur sujet d'étude et ont alors besoin de confronter les différentes hypothèses de restitution. Cette démarche implique ainsi la création d'une variation pour chaque hypothèse. Il s'agit, autrement dit, d'un problème qui peut être pris en compte par la réévaluation. Ainsi, nous prévoyons d'étudier l'applicabilité de notre modeleur et de son mécanisme de réévaluation pour l'archéologie.

Une seconde piste que nous avons identifiée, dans la continuité de la première, est l'annotation temporelle des opérations. L'objectif, dans un premier temps, est d'ajouter une annotation temporelle aux opérations. Puis, dans un second temps, de permettre la réévaluation du modèle suivant l'ordre chronologique, c'est-à-dire réappliquer les opérations de la spécification en fonction des dates qui leur ont été attribuées. Pour cela, nous pouvons utiliser les DAG conçus au cours de nos travaux. Puisque les DAG représentent des évolutions d'orbites, il est donc possible de les utiliser pour déterminer l'instant de création d'une orbite et à quel moment celle-ci est supprimée. Ainsi, il est possible de garantir la cohérence des annotations temporelles vis-à-vis des paramètres topologiques d'une application. À terme, une telle extension permettrait de réévaluer des bâtiments historiques, par exemple, suivant un ordre chronologique supposé. Les résultats de ces réévaluations permettant, *in fine*, de confronter des hypothèses sur la construction et l'évolution de certains biens patrimoniaux au cours du temps.

Bibliographie

[Agb+03] Dago Agbodan, David Marcheix, Guy Pierra et Christophe Thabaud. « A topological entity matching technique for geometric parametric models ». In : *2003 Shape Modeling International.* IEEE. 2003, p. 235-244 (cf. p. 33).

[ALS15] Ken Arroyo Ohori, Hugo Ledoux et Jantien Stoter. « A dimension-independent extrusion algorithm using generalised maps ». In : *International Journal of Geographical Information Science* 29.7 (2015), p. 1166-1186 (cf. p. 1).

[AMP00] Dago Agbodan, David Marcheix et Guy Pierra. « Persistent Naming For Parametric Models ». In : *Proc. 8-th International Conference in Central Europe on Computer Graphics and Visualization and Interactive Digital Media (ECGVIDM 2000)*. 2000, p. 418-425 (cf. p. 2, 35, 37).

[AMP99] Dago Agbodan, David Marcheix et Guy Pierra. « A data model architecture for parametrics ». In : *Journal for Geometry and Graphics* 3.1 (1999), p. 17-38 (cf. p. 32).

[Arn+22] Agnès Arnould, Hakim Belhaouari, Thomas Bellet, Pascale Le Gall et Romain Pascual. « Preserving consistency in geometric modeling with graph transformations ». In : *Mathematical Structures in Computer Science* 32.3 (2022), p. 300-347 (cf. p. 16).

[Auta] Autodesk. *Autodesk 3ds Max product page*. `https://www.autodesk.com/products/3ds-max/overview`. Accessed 2024-07-05 (cf. p. 1, 8).

[Autb] Autodesk. *Autodesk AutoCAD product page*. `https://www.autodesk.com/eu/products/autocad/overview`. Accessed 2024-07-05 (cf. p. 1, 28).

[Autc] Autodesk. *Autodesk Maya product page*. `https://www.autodesk.com/eu/products/maya/overview`. Accessed 2024-07-05 (cf. p. 1, 8).

[Autd] Autodesk. *Autodesk Revit product page*. `https://www.autodesk.com/eu/products/revit/overview`. Accessed 2024-07-05 (cf. p. 1, 3, 28).

[Bab+07] Mehdi Baba-Ali, David Marcheix, Xavier Skapin et Yves Bertrand. « Generic Computation of bulletin boards into Geometric Kernels ». In : *Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*. 2007, p. 85-93 (cf. p. 35).

[Bab10] Mehdi Baba-Ali. « Systeme de nomination hierarchique pour les systemes parametriques ». Thèse de doct. 2010. url : `http://theses.univ-poitiers.fr/notice/view/5362` (cf. p. 2, 37).

[BAL11] Thomas Bellet, Agnès Arnould et Pascale Le Gall. « Rule-based transformations for geometric modeling ». In : *6th International Workshop on Computing with Terms and Graphs (TERMGRAPH 2011), Part of ETAPS 2011*. Saarbrücken, Germany, avr. 2011, 20p (cf. p. 16).

[BB10] Jane Burry et Mark Burry. *The new mathematics of architecture*. Thames & Hudson London, 2010 (cf. p. 2).

[Bel+14] Hakim Belhaouari, Agnès Arnould, Pascale Le Gall et Thomas Bellet. « Jerboa : A graph transformation library for topology-based geometric modeling ». In : *International Conference on Graph Transformation*. Springer. 2014, p. 269-284 (cf. p. 2, 8, 16, 18, 47, 123).

[Bel+17] Thomas Bellet, Agnès Arnould, Hakim Belhaouari et Pascale Le Gall. « Geometric Modeling : Consistency Preservation Using Two-Layered Variable Substitutions ». In : *Graph Transformation : 10th International Conference, ICGT 2017, Held as Part of STAF 2017, Marburg, Germany, July 18-19, 2017, Proceedings*. LNCS 10373. Springer International Publishing, 2017, p. 36-53 (cf. p. 2).

[Ben+17] Fatma Ben Salah, Hakim Belhaouari, Agnès Arnould et Philippe Meseure. « A Modular Approach Based on Graph Transformation to Simulate Tearing and Fractures on Various Mechanical Models ». In : *Journal of WSCG* 25.1 (juin 2017), p. 39-48 (cf. p. 2).

[Ble] Blender. *Blender*. https ://www.blender.org/. Accessed 2024-10-20 (cf. p. 1, 8).

[BMS09] Mehdi Baba-Ali, David Marcheix et Xavier Skapin. « A method to improve matching process by shape characteristics in parametric systems ». In : *Computer-Aided Design and Applications* 6.3 (2009), p. 341-350 (cf. p. 38).

[BNB05] Rafael Bidarra, Paulos J Nyirenda et Willem F Bronsvoort. « A feature-based solution to the persistent naming problem ». In : *Computer-Aided Design and Applications* 2.1-4 (2005), p. 517-526 (cf. p. 2, 37, 38, 44, 53).

[BTG15] Evans Bohl, Olivier Terraz et Djamchid Ghazanfarpour. « Modeling fruits and their internal structure using parametric 3Gmap L-systems ». In : *The Visual Computer* 31 (2015), p. 819-829 (cf. p. 1, 2, 8).

[Car+19] Anaïs Cardot, David Marcheix, Xavier Skapin, Agnès Arnould et Hakim Belhaouari. « Persistent naming based on graph transformation rules to reevaluate parametric specification ». In : *Computer-Aided Design and Applications* 16.5 (2019), p. 985-1002 (cf. p. 35).

[Car19] Anaïs Cardot. « Rejeu basé sur des règles de transformation de graphes ». Thèse de doct. 2019. url : `http://www.theses.fr/2019POIT2254/document` (cf. p. 2, 3, 47, 51, 92, 96).

[CBS23] Dan Cascaval, Rastislav Bodik et Adriana Schulz. « A Lineage-Based Referencing DSL for Computer-Aided Design ». In : *Proceedings of the ACM on Programming Languages* 7.PLDI (2023), p. 76-99 (cf. p. 51).

[CCH96] Vasilis Capoyleas, Xiangping Chen et Christoph M. Hoffmann. « Generic naming in generative, constraint-based design ». In : *Computer-Aided Design* 28.1 (1996), p. 17-26 (cf. p. 2, 35, 37, 40).

[Che95] Xiangping Chen. « Representation, evaluation and editing of feature-based and constraint-based design ». Thèse de doct. Purdue University, 1995 (cf. p. 2, 33, 37, 40).

[Dam24a] Guillaume Damiand. « Combinatorial Maps ». In : *CGAL User and Reference Manual*. 6.0.1. CGAL Editorial Board, 2024. url : https://doc.cgal.org/6.0.1/Manual/packages.html#PkgCombinatorialMaps (cf. p. 1).

[Dam24b] Guillaume Damiand. « Generalized Maps ». In : *CGAL User and Reference Manual*. 6.0.1. CGAL Editorial Board, 2024. url : https://doc.cgal.org/6.0.1/Manual/packages.html#PkgGeneralizedMaps (cf. p. 1).

[Dam24c] Guillaume Damiand. « Linear Cell Complex ». In : *CGAL User and Reference Manual*. 6.0.1. CGAL Editorial Board, 2024. url : https://doc.cgal.org/6.0.1/Manual/packages.html#PkgLinearCellComplex (cf. p. 1).

[Dasa] Dassault Systèmes. *3DS CATIA product page*. https://www.3ds.com/products/catia. Accessed 2024-07-05 (cf. p. 1, 8, 28, 58).

[Dasb] Dassault Systèmes. *3DS Solidworks 3D CAD product page*. https://www.solidworks.com/product/solidworks-3d-cad. Accessed 2024-07-05 (cf. p. 1, 3, 8, 28).

[DL14] Guillaume Damiand et Pascal Lienhardt. *Combinatorial Maps : Efficient Data Structures for Computer Graphics and Image Processing*. A K Peters/CRC Press, sept. 2014 (cf. p. 1, 8, 14, 47).

[Ehr+06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange et Gabriele Taentzer. *Fundamentals of algebraic graph transformation*. Springer, 2006 (cf. p. 12, 14).

[Epi] EpicGames. *Procedural Content Generation Overview*. https://dev.epicgames.com/documentation/en-us/unreal-engine/procedural-content-generation-overview. Accessed 2024-10-20 (cf. p. 1, 8).

[ESR] ESRI. *ArcGIS CityEngine product page*. https://www.esri.com/en-us/arcgis/products/arcgis-cityengine/overview. Accessed 2023-05-09 (cf. p. 1, 8).

[FHM16] Shahjadi Hisan Farjana, Soonhung Han et Duhwan Mun. « Implementation of persistent identification of topological entities based on macro-parametrics approach ». In : *Journal of Computational Design and Engineering* 3.2 (2016), p. 161-177 (cf. p. 44).

[FMH15] Shahjadi Hisan Farjana, Du Hwan Mun et Soon Hung Han. « A method for persistent identification of topological entities in a parametric cad model ». In : *ICMDT (International Conference on Manufacturing, Machine Design and Tribology) 2015*. 2015 (cf. p. 44).

[Fre] FREECAD. *FreeCAD*. https://www.freecad.org/. Accessed 2024-20-24 (cf. p. 1).

[Gau+16] Valentin GAUTHIER, Agnès ARNOULD, Hakim BELHAOUARI, Sébastien HORNA, Michel PERRIN, Mathieu POUDRET et Jean-François RAINAUD. « A topological approach for automated unstructured meshing of complex reservoir ». In : *ECMOR XV-15th European Conference on the Mathematics of Oil Recovery*. EAGE Publications BV. 2016, cp-494 (cf. p. 2).

[Gau19] Valentin GAUTHIER. « Développement d'un langage de programmation dédié à la modélisation géométrique à base topologique, application à la reconstruction de modèles géologiques 3D ». Thèse de doct. 2019. URL : http://www.theses.fr/2019POIT2252/document (cf. p. 2, 123).

[God] GODOT. *Procedural geometry generation*. https://docs.godotengine.org/en/3.1/tutorials/content/procedural_geometry.html. Accessed 2024-10-20 (cf. p. 1, 8).

[Gon+17] Carmen GONZÁLEZ-LLUCH, Pedro COMPANY, Manuel CONTERO, Jorge D. CAMBA et Raquel PLUMED. « A survey on 3D CAD model quality assurance and testing tools ». In : *Computer-Aided Design* 83 (2017), p. 64-79 (cf. p. 3).

[HJ92] Christoph M HOFFMANN et Robert JUAN. « Erep : An editable, high-level representation for geometric design and analysis ». In : *Selected and Expanded Papers from the IFIP TC5/WG5. 2 Working Conference on Geometric Modeling for Product Realization*. 1992, p. 129-164 (cf. p. 40).

[HMV09] Simon HAEGLER, Pascal MÜLLER et Luc VAN GOOL. « Procedural modeling for digital cultural heritage ». In : *EURASIP Journal on Image and Video Processing* (2009) (cf. p. 1).

[Hor+09] Sébastien HORNA, Daniel MENEVEAUX, Guillaume DAMIAND et Yves BERTRAND. « Consistency constraints and 3D building reconstruction ». In : *Computer-Aided Design* 41.1 (2009), p. 13-27 (cf. p. 3).

[Kra+14] Pierre KRAEMER, Lionel UNTEREINER, Thomas JUND, Sylvain THERY et David CAZIER. « CGoGN : N-dimensional meshes with combinatorial maps ». In : *Proceedings of the 22nd International Meshing Roundtable*. Springer. 2014, p. 485-503 (cf. p. 1).

[Kri95] Jiri KRIPAC. « A mechanism for persistently naming topological entities in history-based parametric solid models ». In : *Proceedings of the third ACM symposium on Solid modeling and applications*. 1995, p. 21-30 (cf. p. 2, 35, 37, 38).

[Lie91] Pascal LIENHARDT. « Topological models for boundary representation : a comparison with n-dimensional generalized maps ». In : *Computer-aided design* 23.1 (1991), p. 59-82 (cf. p. 1, 14).

[Lie94] Pascal LIENHARDT. « N-dimensional generalized combinatorial maps and cellular quasi-manifolds ». In : *International Journal of Computational Geometry & Applications* 4.03 (1994), p. 275-324 (cf. p. 8, 47).

[Lin74] Aristid Lindenmayer. « Adding continuous components to L-systems ». In : *L systems* (1974), p. 53-68 (cf. p. 1, 8).

[Mül+06] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer et Luc Van Gool. « Procedural modeling of buildings ». In : *ACM SIGGRAPH Papers*. 2006, p. 614-623 (cf. p. 1).

[Ope] Open CASCADE Technology. *OpenCascade Application Framework*. https://dev.opencascade.org/about/application_framework. Accessed 2024-09-03 (cf. p. 8, 58).

[Pas+22] Romain Pascual, Pascale Le Gall, Agnès Arnould et Hakim Belhaouari. « Topological consistency preservation with graph transformation schemes ». In : *Science of Computer Programming* 214 (2022) (cf. p. 16, 17).

[Pie+96] Guy Pierra, Yamine Ait-Ameur, Frédéric Besnard, Patrick Girard et Jean-Claude Potier. « A general framework for parametric product model within STEP and Parts Library ». In : *European Conference Product Data Technology*. 1996, p. 18-19 (cf. p. 28, 29).

[PL90] Przemyslaw Prusinkiewicz et Aristid Lindenmayer. « The Algorithmic Beauty of Plants ». In : *The Virtual Laboratory* (1990) (cf. p. 1).

[Pou+08] Mathieu Poudret, Agnès Arnould, Jean-Paul Comet et Pascale Le Gall. « Graph transformation for topology modelling ». In : *Graph Transformations : 4th International Conference, ICGT 2008, Leicester, United Kingdom, September 7-13, 2008. Proceedings 4*. Springer. 2008, p. 147-161 (cf. p. 2).

[Pou09] Mathieu Poudret. « Transformations de graphes pour les opérations topologiques en modélisation géométrique : application à l'étude de la dynamique de l'appareil de Golgi ». Thèse de doct. 2009. url : http://www.theses.fr/2009EVRY0039/document (cf. p. 16).

[QB15] Ramona Quattrini et Eleonora Baleani. « Theoretical background and historical analysis for 3D reconstruction model. Villa Thiene at Cicogna ». In : *Journal of Cultural Heritage* 16.1 (jan. 2015), p. 119-125 (cf. p. 1).

[Sie] Siemens. *Siemens NX overview page*. https://plm.sw.siemens.com/en-US/nx/. Accessed 2024-07-05 (cf. p. 1, 8, 28).

[Ter+09] Olivier Terraz, Guillaume Guimberteau, Stéphane Mérillou, Dimitri Plemenos et Djamchid Ghazanfarpour. « 3Gmap L-systems : an application to the modelling of wood ». In : *The Visual Computer* 25 (2009), p. 165-180 (cf. p. 1, 8).

[Uni] Unity. *Unity Engine*. https://unity.com/products/unity-engine. Accessed 2024-10-20 (cf. p. 1, 8).

[Vin83] Andrew Vince. « Combinatorial maps ». In : *Journal of Combinatorial Theory, Series B* 34.1 (1983), p. 1-21 (cf. p. 1).

[WN05] Yan Wang et Bartholomew O. Nnaji. « Geometry-based semantic ID for persistent and interoperable reference in feature-based parametric modeling ». In : *Computer-Aided Design* 37.10 (2005), p. 1081-1093 (cf. p. 2, 37, 38).

[WS05] Peter WORTH et Susan STEPNEY. « Growing music : Musical interpretations of L-systems ». In : *APPLICATIONS OF EVOLUTIONARY COMPUTING, PROCEEDINGS*. Sous la dir. de F ROTHLAUF, J BRANKE, S CAGNONI, DW CORNE, R DRECHSLER, Y JIN, P MACHADO, E MARCHIORI, J ROMERO, GD SMITH et G SQUILLERO. Springer, 2005, p. 545-550 (cf. p. 1).

[Wu+01] Junjun WU, Tianbing ZHANG, Xinfang ZHANG et Ji ZHOU. « A face based mechanism for naming, recording and retrieving topological entities ». In : *Computer-Aided Design* 33.10 (2001), p. 687-698 (cf. p. 2, 33-38, 42).

Table des figures

Table des listings

Liste des tableaux

Modélisation et rejeu basés sur des règles

La construction d'un objet complexe est un processus fastidieux, impliquant des cycles successifs d'essais et de corrections. Pour alléger la difficulté de la conception, les systèmes de modélisation paramétrique actuels proposent une fonctionnalité de rejeu permettant de regénérer un objet grâce à l'édition de ses paramètres. Cependant, ce processus nécessite soit une détection des changements topologiques basée sur une analyse complète de l'objet modifié, ce qui est coûteux ; soit ces changements sont codés explicitement dans les opérations, ce qui peut être source d'erreurs. Pour répondre à ces limites, nos travaux se concentrent sur le développement d'un système de modélisation basé sur des règles, permettant de rejouer un processus constructif, notamment avec l'ajout, la suppression et le déplacement d'opérations. Nous considérons des opérations formalisées par des règles de transformation de graphe Jerboa. Notre première contribution est une analyse syntaxique des opérations permettant de détecter les changements topologiques (création, scission, fusion, etc.). Cette analyse des règles est statique et indépendante des objets auxquels celles-ci sont appliquées. Ainsi, les changements topologiques peuvent être détectés et suivis automatiquement. Dans certains cas définis, une analyse localisée au sein de l'objet permet de confirmer ou non les changements détectés. La seconde contribution s'appuie sur notre approche de détection des changements topologiques pour proposer un mécanisme de rejeu. Un processus constructif est une suite d'opérations appliquées sur des entités topologiques spécifiques. Rejouer un processus constructif nécessite de résoudre le problème classique de la nomination persistante des entités topologiques dans un modèle géométrique. Pour répondre à ce problème, nous proposons une reconstitution de l'historique des entités topologiques référencées dans le processus constructif. Chaque entité possède une histoire unique qui l'identifie. Grâce à ce mécanisme, nous proposons un référencement robuste des entités topologiques lors du rejeu. Notre troisième contribution vise à étendre notre mécanisme de rejeu aux scripts de règles qui permettent de construire des opérations complexes à l'aide de structures de contrôles. Nous proposons pour cela d'utiliser les scripts Jerboa avec des structures de contrôles classiques telles que les itérations et les alternatives. Nous proposons une extension de notre mécanisme de rejeu pour inclure ces structures de contrôle, offrant ainsi une plus grande flexibilité dans la modélisation et la gestion des opérations complexes.

Mots clés : Modélisation 3D / Graphes topologiques, Théorie des / Topologie Combinatoire / Réécriture, Systèmes de (informatique) / Modélisation à base topologique / Règles de transformation de graphe / Nomination persistante / Réévaluation / Détection de changements topologiques / Cartes généralisées

Rule-based modelling and reevaluation

Designing a complex object is a tedious process involving repeated cycles of trial and error. In order to alleviate such a difficulty in the designing process, current parametric modelling systems offer some reevaluation mechanisms allowing a user to rebuild an object based on the editing of its parameters. However, such a process requires either processing an entire mesh in order to detect topological changes, which is computationally expensive, or hard-coding the changes in the modelling operations, which is computationally efficient but increases the risks to introduce detection errors. To address these limitations, our works focus on the development of a rule-based modelling system dedicated to the reevaluation of modelling processes. In particular, this system allows the addition, deletion and reordering of the operations defining those processes.We consider operations formalised with Jerboa's graph transformation rules. Our first contribution is the syntactic analysis of the operations allowing for the detection of topological changes (creation, split, merge, and so on). These analyses are statically performed on rules, independently of the object onto which they are being applied. Thus, topological changes can be automatically detected and tracked. In some defined cases, a localised analysis performed within the object can assert whether the event has occurred or not. Our second contribution makes use of our topological changes detection approach in order to offer a reevaluation mechanism. Considering that a modelling process is a record of operations sequentially applied on specific topological entities, reevaluating a modelling process first requires solving the long-standing problem of persistently naming topological entities in a geometric model. To achieve this goal, we offer to reconstitute the histories of topological entities referenced within a modelling process. Each entity can be identified through its history, which is unique. With this mechanism, we reference topological entities in a robust way throughout reevaluation. Our third contribution aims to extend our reevaluation mechanism to include scripts of rules which can be used to create more complex operations. The Jerboa script language makes it possible to create such scripts with usual control structures such as alternatives and iterations. We extend our reevaluation mechanism by including these control structures, hence, enabling the user with greater versatility in modelling and managing complex operations.

Keywords: Three-dimensional modeling / Topological Graph Theory / Combinatorial Topology / Graph rewriting systems (Computer science) / Topology-based modeling / Graph transformation rules / Persistant naming / Reevaluation / Topological change detection / Generalized Maps