Géométrie du Parallélisme

Notes de Cours du DEA "Programmation"

Eric Goubault[1]

12 janvier 2001

[1]DEIN/SLA, CEA/Saclay, 91191 Gif-sur-Yvette, FRANCE, email:Eric.Goubault@cea.fr

Chapitre 1

Géométrie et Parallélisme

1.1 Mes premiers pas

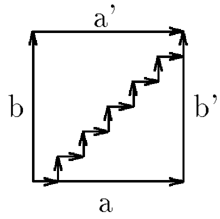


FIG. 1.1 – Entrelacements possibles.

L'article de Vaughan Pratt [Pra91], qui m'a beaucoup inspiré au début de ma thèse, était essentiellement motivé par un défaut dans la dualité entre structures d'événements et automates[1].

Les modèles introduits pour tenter de combler ce défaut, attribuable au fait qu'une des sémantiques est du "vrai parallélisme", l'autre étant une simulation par entrelacements, étaient basés sur une forme ou une autre de CW-complexes (ou de n-catégories, on y reviendra). De tels objets sont des collages de formes élémentaires le long de leurs bords. L'explication qui suit est inspirée de ma thèse.

Considérons d'abord les systèmes de transition. Ils permettent de modéliser le parallélisme avec une sémantique par entrelacement. Ce sont déjà des objets géométriques, mais peu reconnus en tant que tels. Leur représentation sous forme de graphe[2], permet d'y reconnaître les branchements et les confluences, les cycles, les états initiaux et finaux, ainsi que les états inatteignables. Toutes ces notions géométriques sont d'importance; en particulier les branchements pour les équivalences sémantiques ("branching-time"), et les points morts et états inatteignables pour l'analyse statique (par exemple le model-checking).

La modélisation de systèmes parallèles par entrelacement construit naturellement des formes cubiques; par exemple des carrés comme $a \mid b$:

$$\begin{tikzcd} \cdot \arrow[r, "a"] \arrow[d, "b"'] \arrow[dr, phantom, "A"] & \cdot \arrow[d, "b'"] \\ \cdot \arrow[r, "a'"'] & \cdot \end{tikzcd}$$

qui représente l'exécution asynchrone des actions a et b (a' et b' sont des transitions qui ont pour "étiquette" respectivement a et b).

L'idée naturelle est de se dire que cet entrelacement est un codage un peu coûteux du fait que a et b sont indépendants, à comparer avec les systèmes d'événements par exemple. Ce que l'on souhaite réellement dire, c'est que tous les mélanges possibles de sous-actions de a et de b sont des chemins d'exécution, comme on le montre dans la figure 1.1.

Il est donc naturel de considérer également toutes les subdivisions possibles du carré, c'est-à-dire l'intérieur A du carré en plus de son bord. Le fait de considérer également l'intérieur du carré, puis aussi, l'intérieur du cube (quand on a trois processus) etc. en plus du graphe de transition amène à la notion d'*ensemble semi-cubique*.

La façon habituelle de définir un ensemble semi-cubique est de définir des fonctions bords; par exemple pour le carré, on a quatre opérateurs bord, correspondant respectivement à a, b, a' et b'. Ce n'est pas tout car on veut en plus

[1] Qui plus tard, motiveront également l'introduction des espaces de Chu [Pra94].

[2] Donc d'espaces simpliciaux ou d'espaces cubiques particuliers!

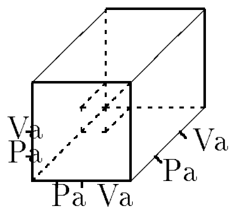


FIG. 1.2 – Un ensemble semi-cubique représentant trois accès concurrents à un sémaphore initialisé à deux.

coder la "direction du temps". En dimension un on utilise en effet un graphe dirigé de transitions pour cela; on veut faire de même pour les ensembles semi-cubiques. Le choix que j'ai fait est de diviser en deux la famille d'opérateurs bords. Dans le cas du carré, on a une famille de deux opérateurs bord début d_0^0 et d_1^0, avec $d_0^0(A) = a$ et $d_1^0(A) = b$, et une famille de deux opérateurs bord fin d_0^1 et d_1^1, avec $d_0^1(A) = a'$ et $d_1^1(A) = b'$. Cela étend simplement la notion de début et de fin dans un graphe dirigé.

Si une 2-transition (carré) n'est qu'une relation d'indépendance entre deux 1-transitions, une 3-transition, ou cube, n'est pas simplement un raccourci pour 3 relations d'indépendance. Le fait que l'action a soit indépendante de l'action b, que b soit indépendante de c et que c soit indépendante de a n'implique pas que a, b et c puissent s'exécuter toutes trois ensemble de façon asynchrone. C'est le cas par exemple d'une abstraction d'un spooler d'imprimante avec deux imprimantes, ou de deux unités flottantes[3], ou encore d'un tampon (de communication) à 2 cellules, c'est-à-dire d'un sémaphore s initialisé à 2[4]. En utilisant les notations de E. W. Dijkstra [Dij68], il suffit de considérer les trois actions $a = b = c = Ps$; s peut être partagé par deux mais pas par les trois processus en même temps (voir figure 1.2). Cette sorte d'objet qui synchronise faiblement est d'une grande importance pour les protocoles de systèmes distribués tolérants aux pannes[5].

Ces propriétés ne sont pas exprimables simplement dans la plupart des modèles du parallélisme existants comme les systèmes de transition asynchrones, les structures d'événements premières etc. une exception notable étant les réseaux de Pétri. Ceux-ci ont d'autres désavantages: ils sont très peu compositionels, ce qui les rend assez peu agréables pour l'analyse de programmes (mais utilisables néanmoins pour des protocoles booléens).

Bien sûr cette discussion se généralise. L'accès concurrent de $n+1$ processus à un sémaphore initialisé à n est représenté par le bord d'un hypercube de dimension $n+1$. Nous avons donc besoin de n-transitions pour tout $n > 0$.

De même que pour $n = 1$ et $n = 2$, on divise en deux les opérateurs bords décrivant la source et le but d'une n transition. Chaque n-transition a n sources de dimension $n-1$, données par des opérateurs bords sources d_i^0, $0 \le i \le n-1$, ainsi que n buts de dimension $n-1$, donnés par des opérateurs but d_j^1, $0 \le j \le n-1$. Par exemple pour $n = 3$,

$$\begin{tikzcd}
(0,0,0) \arrow[rr, "b"] \arrow[dd, "c"'] \arrow[dr, "a"] & & (1,0,0) \arrow[dd] \arrow[dr] & \\
& (0,1,0) \arrow[rr] \arrow[dd] & & (1,1,0) \arrow[dd] \\
(0,0,1) \arrow[rr] \arrow[dr] & & (1,0,1) \arrow[dr] & \\
& (0,1,1) \arrow[rr] & & (1,1,1)
\end{tikzcd}$$

L'intérieur D du cube (objet de dimension trois) a trois bords début, les trois faces contenant $(0,0,0)$, et trois bords fin, les trois faces contenant $(1,1,1)$.

Soient A, B et C les faces (respectivement)

$$((0,0,0),(1,0,0),(0,0,1),(1,0,1))$$

$$((0,0,0),(0,1,0),(0,0,1),(0,1,1))$$

$$((0,0,0),(1,0,0),(0,1,0),(1,1,0))$$

Soient A', B' et C' les faces parallèles à A, B et C respectivement.

Posons $d_0^0(D) = A$, $d_1^0(D) = B$, $d_2^0(D) = C$ et $d_0^1(D) = A'$, $d_1^1(D) = B'$, $d_2^1(D) = C'$. Alors $d_0^0(A) = b$, $d_1^0(A) = c$, $d_0^0(B) = a$, $d_1^0(A) = c$, $d_0^0(C) = a$, $d_1^0(C) = b$.

On peut prouver ce que l'on constate ici, c'est-à-dire que les opérateurs bords peuvent être définis de telle manière que l'on a la règle de

[3] Par exemple dans les microprocesseurs Pentium d'Intel.

[4] Ce que l'on appelera un n-sémaphore (ici avec $n = 2$).

[5] Une pile FIFO partagée à deux entrées permet par exemple de résoudre le problème du consensus sans-attente pour deux processus. Une bonne référence pour ces problèmes est [Lyn96].

commutation suivante (pour $i < j$ and $k,l = 0,1$):

$$d_i^k \circ d_j^l \quad = \quad d_{j-1}^l \circ d_i^k$$

Par exemple pour une 2-transition, la relation avec $k = 0$, $l = 1$ et $i = 0$, $j = 1$ veut dire que la source du but numéro un (b') est la même chose que le but de la source numéro zéro (a).

Inversement toute forme géométrique construite en collant des hypercubes de dimension quelconque *le long de leurs faces* peut être présenté comme un *ensemble semi-cubique*:

Définition 1 *Un ensemble semi-cubique est un ensemble gradué $M = (M_i)_{i \in \mathbb{N}}$ avec des familles d'opérateurs:*

$$M_n \underset{d_j^1}{\overset{d_i^0}{\rightrightarrows}} M_{n-1}$$

($i, j = 0, \ldots, n-1$) satisfaisant les relations

$$d_i^k \circ d_j^l \quad = \quad d_{j-1}^l \circ d_i^k$$

J'ai utilisé cette formalisation dès mon premier article sur le sujet [GJ92]. En fait, les ensembles cubiques et semi-cubiques ont une histoire assez ancienne. Ils ont été utilisés dans les premiers développements de la topologie algébrique par Daniel Kan et par Jean-Pierre Serre dans sa thèse [Ser51]. Aujourd'hui, la topologie algébrique combinatoire utilise plus volontiers les ensembles simpliciaux, union de simplexes de toute dimension, collées selon leurs faces. Dans la thèse de Jean-Pierre Serre, la raison pour laquelle les ensembles cubiques ont été préférés aux ensembles simpliciaux est que pour étudier des fibrations, qui sont localement des projections canoniques d'un produit cartésien de deux espaces topologiques vers le premier, il est plus simple de considérer des ensembles cubiques qui ont de bonnes propriétés vis-à-vis des projections[6].

Dans le cas de la sémantique du parallélisme, l'utilisation d'ensembles cubiques ou semi-cubiques est naturelle comme je l'ai expliqué plus haut (en partant de la sémantique par entrelacement).

Tout ceci a un rapport direct avec les graphes d'avancement ou "progress graphs", qui sont une analogie géométrique introduite il y a plus de trente ans. Je n'ai réalisé cela qu'assez récemment, et n'ai retrouvé cette trace assez ancienne qu'après l'article de J. Gunawardena [Gun94]. La section suivante est tirée de [Gou00].

1.2 Les origines

Le premier modèle géométrique (au sens de la topologie algébrique) du parallélisme que je connais est le modèle des "progress graphs" (que j'appelerai par la suite graphe d'avancement) et est apparu en théorie des systèmes d'exploitation, pour décrire en particulier le problème d'"étreinte fatale"[7] dans les systèmes multiprogrammés. Les graphes d'avancement ont été introduits dans [CES71] qui les attribue à E. W. Dijkstra. En fait, ils sont apparus un peu plus tôt, pour des raisons éditoriales semble-t-il, dans [SC70].

L'idée de base est de donner une description des traces d'exécution d'un système comprenant plusieurs processus modifiant des ressources partagées. Considérons une ressource partagée a que l'on identifie à un sémaphore quelconque, c'est-à-dire à un objet qui peut être partagé par n mais pas $n+1$ processus ($n \geq 0$) - dans ce cas on l'appelera n-sémaphore. Par exemple, a peut-être une simple variable partagée, et alors $n = 1$, ce qui permet d'assurer l'exclusion mutuelle d'accès à celle-ci. Alors, étant donné n processus séquentiels déterministes $Q_1, \ldots, Q_n$ que l'on abstrait par une suite de verrouillage et de déverrouillages d'objets partagés,

$$Q_i = R^1 a_i^1 . R^2 a_i^2 \cdots R^{n_i} a_i^{n_i}$$

(R^k étant soit P - verrouillage - soit V - déverrouillage - en utilisant la notation de E. W. Dijkstra [Dij68]), il y a un moyen très naturel de représenter les comportements possibles de leur exécution en parallèle. Pour ce faire, on associe à chaque processus une coordonnée dans $\mathbb{R}^n$. L'état du système parallèle correspond à un point de $\mathbb{R}^n$, dont la $i$ème coordonnée décrit l'état (ou temps local) sur le $i$ème processus.

Supposons que chaque processus démarre au temps local 0 et finit au temps local 1; à chaque action P et V peut être associée une suite de nombres réels entre 0 et 1, qui reflète leur ordre d'exécution. L'état initial du système est alors $(0, \ldots, 0)$ et l'état final $(1, \ldots, 1)$. Considérons

[6] Même si plus tard, une construction simpliciale a été publiée, voir [MC85] et la "construction de Dress".

[7] "Deadly embrace" en anglais, comme E. W. Dijkstra l'a nommé à l'origine; on dit plus couramment maintenant "point mort".

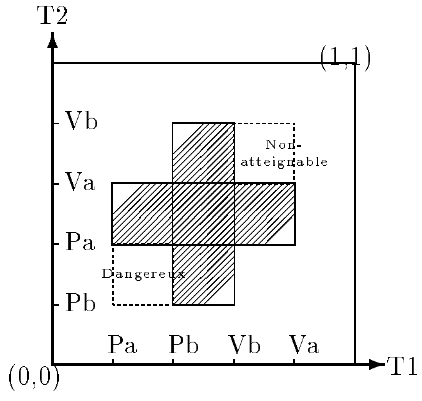


FIG. 1.3 – Exemple de graphe d'avancement

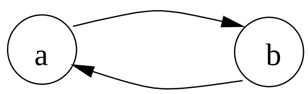


FIG. 1.4 – Le graphe de requêtes correspondant

par exemple le programme composé des deux processus parallèles $T_1 = Pa.Pb.Vb.Va$ et $T_2 = Pb.Pa.Va.Vb$, utilisant deux ressources partagées a et b, dont l'ensemble des états est représenté à la figure 1.3.

La partie grisée représente des états qui ne sont permis dans aucune exécution: en effet un point intérieur à cette partie est un point ne respectant pas la propriété d'exclusion mutuelle. On appelle cette partie, la partie interdite. Un chemin d'exécution est un chemin de l'état initial $(0, \ldots, 0)$ à l'état final $(1, \ldots, 1)$ n'entrant pas dans la région interdite et qui de plus a la propriété d'être croissant en chaque coordonnée; on ne peut pas inverser le sens du temps. On appelle ces chemins, chemins dirigés ou dichemins.

Cette contrainte de croissance en chaque coordonnée implique que les dichemins atteignant la région hachurée sous la région interdite, appelée "région dangereuse", sont obligés d'arriver à un point mort, et donc ne peuvent atteindre l'état terminal. On voit également sur la figure 1.3 que tous les dichemins atteignant l'état terminal et passant au dessus de la région interdite sont équivalents en un certain sens: ils sont tous caractérisés par le fait que T_2 acquiert a et b avant T_1. Nous appelons cette propriété, une propriété d'ordonnancement (sous-entendu, des accès des processus aux ressources partagées). De même, tous les chemins en dessous de la région interdite sont caractérisés par le fait que T_1 acquiert a et b avant T_2.

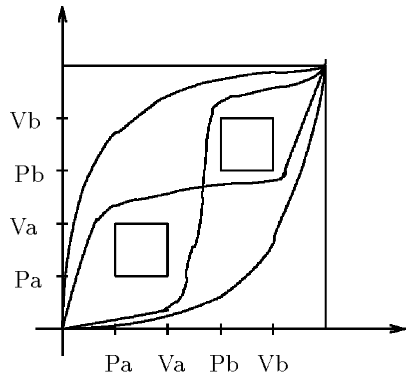


FIG. 1.5 – Le graphe d'avancement correspondant à $Pa.Va.Pb.Vb \mid Pa.Va.Pb.Vb$

On peut déjà reconnaître dans cette présentation tous les ingrédients qui sont au coeur des principaux problèmes de la topologie algébrique, c'est-à-dire la classification des formes modulo déformation "élastique" (on peut déformer sans jamais déchirer ou coller). D'ailleurs, les coordonnées choisies pour représenter les différentes actions P et V ne sont pas importantes, leur ordre seul compte. On peut donc les déformer de toutes manières, en conservant leur ordre. Donc les propriétés d'ordonnancement, de point mort etc. sont invariantes par déformation, ou homotopie. Il s'agit néanmoins d'une homotopie un peu particulière, et c'est ce qui fait que le reste de la théorie est un peu compliquée. Cette homotopie, qui ne doit pas inverser le sens du temps, est appelée homotopie dirigée, ou dihomotopie, on la définira plus formellement dans la section 1.3.

Un point important est que si on arrive à caractériser de façon pratique ces classes de dihomotopie, on pourra trouver des méthodes efficaces de réduction de l'espace d'états pour faire de l'analyse statique de programmes parallèles. On y reviendra dans le chapitre 2.

Pour se convaincre de la différence avec l'homotopie traditionnelle, considérons deux formes homotopiquement équivalentes au sens usuel du terme. Ces deux formes, figure 1.5 et figure 1.6, sont bidimensionnelles et comportent deux trous. Mais elles ont respectivement quatre dichemins (partant de l'état initial et allant à l'état final) et trois dichemins ou ordonnancements modulo dihomotopie.

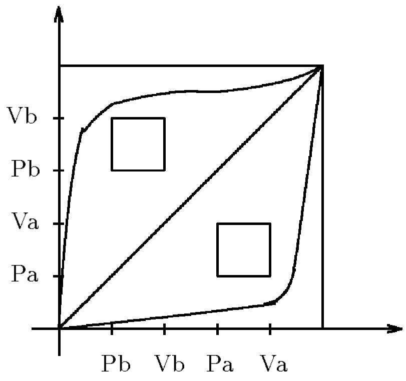


FIG. 1.6 – Le graphe d'avancement correspondant à $Pb.Vb.Pa.Va \mid Pa.Va.Pb.Vb$

Il y a évidemment une méthode simple pour déterminer les points morts pour les processus PV, qui était connue bien avant les graphes d'avancement. Il s'agit d'une méthode utilisant un "graphe de requêtes". La figure 1.4 montre le graphe correspondant dans le cas des processus de la figure 1.3. Les noeuds du graphe de requêtes sont les ressources du système parallèle, c'est-à-dire ici, les sémaphores. On trace une arête orientée d'une ressource (ou noeud) x à une ressource y s'il existe un processus qui, ayant acquis un verrou sur x, a besoin d'un verrou sur y à un instant donné. Une condition suffisante pour qu'un système parallèle soit sans point mort est que son graphe de requêtes soit acyclique - c'est évidemment une condition d'ordre géométrique.

Malheureusement, ce n'est pas une condition nécessaire en général. Par exemple, un graphe de requêtes ne peut exprimer ce qui se passe dans le cas de n-sémaphores, c'est-à-dire de ressources qui peuvent être partagées par n mais pas par $n+1$ processus. Cela nécessite en fait une version multidimensionnelle de graphes.

En utilisant les graphes d'avancement, on peut trouver par exemple dans [CES71] un algorithme pour déterminer l'absence de point mort en $O(n^2)$ où n est le nombre de processus du système parallèle. La notion de région dangereuse y avait été introduite également, avec l'espoir de pouvoir déterminer automatiquement quels seraient les ordonnanceurs qui empêcheraient d' arriver à un point mort, et qui seraient équivalents au système de départ. Ce travail était limité aux sémaphores binaires[8]. Un algorithme complet de détection de points morts sur des graphes d'avancement est décrit dans [CRJ87], qui ne souffre pas de cette restriction. Ce n'est cependant pas un algorithme optimal.

L'article [LP81] décrit un algorithme permettant de prouver l'absence de point mort pour deux transactions avec des sémaphores binaires, en complexité en temps $O(n \log n \log \log n)$ (où n est le nombre de rectangles interdits).

L'utilisation principale de ces considérations géométriques a été de permettre de prouver que des protocoles d'accès à des bases de données distribuées, contenant des objets partagés, protégés par des sémaphores, ne créent pas de point mort, et également, qu'ils assurent des propriétés de cohérence sans trop limiter la performance du système.

Un exemple simple de propriété permettant d'assurer la cohérence d'une base de données distribuées est la *séquentialisation*. Par exemple, la figure 1.6 représente le graphe d'avancement d'une base de données avec deux transactions

$$T_1 = Pb.Vb.Pa.Va$$

$$T_2 = Pa.Va.Pb.Vb$$

qui essaient de modifier deux items a et b. Tous les chemins d'exécution au dessus du trou de gauche sont équivalents à une exécution séquentielle de la transaction T_1 puis de la transaction T_2; ils sont tous dihomotopiquement équivalents. Tous les chemins d'exécution en dessous du trou de droite sont équivalents à une exécution séquentielle de la transaction T_2 puis de la transaction T_1. Le troisième type de dichemin, entre les deux trous, décrit la situation suivante: T_1 acquiert b, T_2 acquiert a puis T_1 acquiert a et T_2 acquiert b. Pour se convaincre que ce type de chemin peut ne pas être très souhaitable pour la cohérence de la base de données distribuées considérée, prenons une image simple: supposons que les items représentent des billets d'avions. Le billet aller est représenté par a, le billet retour par b. Les transactions T_1 et T_2 sont le fait de deux clients, chacun à deux guichets très éloignés qui n'ont aucun moyen de se synchroniser directement. Ce troisième dichemin correspond donc au fait que T_1 a réservé uniquement son billet aller, alors que T_2 a uniquement réservé son billet retour! Ce n'est certainement pas un comportement équivalent à

[8] Il y a un moyen de traduire les sémaphores généraux en sémaphores binaires, voir [Dij68], mais cela utilise un codage avec des entiers, qui ne sont pas représentables dans les graphes d'avancement.

une exécution séquentielle des deux transactions T_1 et T_2.

La propriété de séquentialisation: "chaque exécution parallèle des transactions doit être équivalente à une exécution séquentielle des transactions" peut être prise comme définition de correction de protocoles d'accès à des bases de données distribuées[9].

Malheureusement, être séquentialisable est une propriété NP-complète (voir [Pap79]) même quand le modèle est restreint aux sémaphores binaires. Malgré tout, on peut arriver à certaines caractérisations agréables avec les graphes d'avancement.

L'étude de bases de données distribuées avec des graphes d'avancement a débuté avec l'article [YPK79] puis a été poursuivie dans [LP81] et [Pap83]. Dans [LP81], on peut trouver une description d'un algorithme de preuve de séquentialisation dérivé d'un modèle de graphe d'avancement, mais pour deux transactions seulement. Le principe de l'algorithme est de déterminer la connexité de la clôture de la région interdite[10]. Bien entendu, le problème avec l'exemple de la figure 1.6 est que la région interdite n'est pas connexe, permettant aux dichemins d'entrelacer certaines requêtes de façon inextricable.

Une méthode décrite dans [YPK79] permet de généraliser la vérification des conditions de séquentialisabilité à plus de transactions. On peut prouver (encore dans [LP81]) que la complexité en temps pour la vérification de la propriété de séquentialisation pour d transactions manipulant des sémaphores binaires est de $O(nd\, 2^d + d^2 \log n \log\log n)$.

Beaucoup de travail de nature algorithmique a été effectué pour améliorer ces complexités. L'algorithme optimal, décrit dans [SSW85], a une complexité de $O(n \log n)$ en temps et de $O(n)$ en espace pour deux transactions. Pour plus de deux transactions, il suffit d'utiliser le procédé de M. Yannakakis permettant de réduire le problème de la séquentialisation en dimension supérieure à deux en plusieurs problèmes en dimension deux.

De nouveaux algorithmes ont été développés depuis, voir par exemple [Rau00], qui reposent sur des notions nouvelles, introduites par L. Fajstrup, M. Raussen et moi-même dans [FGR99].

Je vais maintenant passer en revue les diverses formalisations géométriques du parallélisme sur lesquelles j'ai travaillé, sans suivre l'ordre chronologique. La formalisation "topologique" qui fait l'objet de la section suivante est la plus récente, et date essentiellement de [FGR98a] et de [FGR99][11]. Les calculs d'invariants homologiques sont eux beaucoup plus anciens, et ont constitué le fil rouge de ma thèse [Gou95a], à partir de [GJ92], [Gou93], [Gou95b]. On peut trouver un embryon de formalisation à partir de ω-catégories dans [Pra91], mais il a fallu beaucoup plus de temps pour faire le lien entre cette formalisation et l'approche homologique. J'y ai réfléchi de manière régulière durant les cinq dernières années, et ai beaucoup interagi avec P. Gaucher, qui est celui qui non seulement a dégagé les bonnes constructions qui permettent d'améliorer les théories homologiques introduites dans ma thèse [Gou95a] mais qui en plus a mis au point des concepts nouveaux permettant de produire quantité d'invariants algébriques associés aux HDA [Gau00c]. Nous collaborons actuellement en vue d'un article "unificateur" des deux approches, qui permettra de disposer du même arsenal d'invariants algébriques dans l'approche topologique. J'en dis quelques mots aux sections 1.6 et 1.7.

1.3 Point de vue topologique

La plupart des modèles géométriques utilisés depuis l'article [Pra91] sont basés sur une forme ou une autre d'ensembles cubiques. Dans des travaux récents, j'ai introduit avec Martin Raussen et Lisbeth Fajstrup, dans [FGR98a] en particulier, des modèles topologiques. Ce sont les espaces localement partiellement orientés, qui généralisent les modèles précédents. L'idée était de pouvoir vraiment raisonner géométriquement, à homéomorphisme près, et non plus algébriquement ou combinatoirement. Bien sûr, comme dans le cas des espaces simpliciaux et de la topologie algébrique classique, il y a, comme on le montre dans [FGR99], des relations naturelles entre les représentations combinatoires et topologiques, grâce à une paire de foncteurs adjoints: la réalisation géométrique et le foncteur cube

[9] C'est néanmoins une notion un peu contraignante, et il existe d'autres versions, plus faibles.

[10] Ces algorithmes sont également décrits dans le chapitre "The Geometry of Rectangles" du livre [PS93].

[11] Même si [Gou96c] contenait déjà des idées topologiques, pour une application aux processus temps-réels, mais qui maintenant me semblent un peu trop restrictives.

singulier[12]. Je reprends dans cette section quelques éléments de l'article [FGR99].

Un graphe d'avancement est un espace topologique - un sous-espace de $\mathbb{R}^n$ en fait. En plus de la topologie qui nous permet de définir la notion de chemin continu, on a besoin d'un ordre partiel nous permettant de caractériser l'avancement du temps. Les deux doivent être compatibles un minimum, ce qui amène à la définition suivante:

Définition 2 *Un po-espace (ou espace partiellement ordonné) est une paire $(X, \leq)$ formée d'un espace topologique X et d'un ordre partiel $\leq$ tel que $\leq$ soit fermée (c'est-à-dire que $\leq$ soit un fermé de $X \times X$ avec la topologie produit).*

Cela implique deux choses qui semblent naturelles: les ensembles $\uparrow x = \{y \mid x \leq y\}$ et $\downarrow x = \{y \mid y \leq x\}$ sont des fermés de X.

Si on peut donner une sémantique pour les processus dont nous parlions dans la section 1.2, on est bien vite limité: par exemple il n'est pas possible de donner une sémantique aux processus récursifs ou aux boucles autrement qu'en les dépliant entièrement. Cela n'est pas bien satisfaisant et motive une définition plus locale: on peut penser imposer par exemple, un ordre partiel localement dans un espace topologique, mais pas globalement.

Définition 3 *Soit X un espace topologique. Une collection $\mathcal{U}(X)$ de paires $(U, \leq_U)$ d'ouverts de X, recouvrant X, partiellement ordonnés par $\leq_U$ est appelé un ordre partiel local sur X si pour tout $x \in X$ il existe un voisinage ouvert non vide $W(x) \subset X$ tel que les restrictions de $\leq_U$ à $W(x)$ coincident pour tout $U \in \mathcal{U}(X)$, c'est-à-dire,*

pour tout $U_1, U_2 \in \mathcal{U}(X)$ tels que $x \in U_i$ et pour tout $y, z \in W(x) \cap U_1 \cap U_2$:

$$y \leq_{U_1} z \Leftrightarrow y \leq_{U_2} z.$$

C'est une définition[13] très similaire à celle des variétés différentielles, qui "ressemblent" localement à $\mathbb{R}^n$. On appelera la collection des ouverts U un atlas pour X.

Comme pour les variétés différentielles, on peut avoir plusieurs atlas équivalents:

Définition 4
- *Deux ordres partiels locaux sur X sont équivalents si leur union est un ordre partiel local sur X.*
- *Un espace localement partiellement ordonné est la donnée d'un espace topologique X et d'une classe d'équivalence d'ordre partiels locaux sur X. Si de plus il existe un recouvrement $\mathcal{U}$ dans cette classe d'équivalence tel que tous les $(U, \leq_U) \in \mathcal{U}$ sont des po-espaces, alors on dit que X est un po-espace local.*

On gagne ici la possibilité de considérer des boucles, c'est-à-dire des points (ou états) par lesquels on peut passer dans un chemin d'exécution un nombre quelconque de fois.

Donnons un exemple simple. Une boucle "dirigée" $S^1 = \{e^{i\theta} \in \mathbb{C}\}$ est un po-espace local: il suffit de prendre $U_1 = \{e^{i\theta} \in S^1 | 0 < \theta < 2\pi\}$ avec l'ordre induit par l'ordre naturel sur les θ et $U_2 = \{e^{i\theta} \in S^1 | \pi < \theta < 3\pi\}$ encore une fois avec l'ordre naturel sur les θ.

Nous avons maintenant besoin de définir les morphismes entre espaces localement partiellement ordonnés, qui vont nous donner à leur tour la notion de dichemin.

Définition 5 *Soient $(X, \mathcal{U})$ et $(Y, \mathcal{V})$ deux espaces localement partiellement ordonnés.*

Une fonction continue $f : X \rightarrow Y$ est appelée difonction *si pour tout $x \in X$ il existe un sous-ensemble $V(f(x)) \subset Y$ sur lequel $\leq_Y$ est bien défini et $U(x) \subset f^{-1}(V(f(x)))$ sur lequel $\leq_X$ est bien défini, tels que, pour tout $y, z \in U(x)$: $\quad y \leq_X z \Rightarrow f(y) \leq_Y f(z)$.*

Un dichemin sur X est alors une difonction $f : \vec{I} \rightarrow X$ où $\vec{I}$ est l'espace topologique $I = [0, 1] \subseteq \mathbb{R}$ avec l'ordre partiel (global) hérité de celui de $\mathbb{R}$. On note $P_1(X)$ l'ensemble des dichemins de X, et $P_1^{\alpha,\beta}(X)$ l'ensemble des dichemins de X de α à β.

Maintenant, on peut définir précisément ce que l'on entend par déformer un dichemin, c'est la notion d'homotopie dirigée, ou dihomotopie. Il est important ici de faire attention aux extrémités des chemins. L'idée est que contrairement au cas classique où il suffit de choisir un point "base" et ensuite de considérer les boucles autour de ce point base modulo homotopie, pour caractériser une forme, il nous faut considérer deux points base. En effet, il est peu probable qu'il y ait en général beaucoup de boucles orientées (par exemple il n'y a que les boucles triviales constantes dans un graphe d'avancement), donc il faut plutôt choisir un point base de départ et un d'arrivée et ensuite étudier tous les dichemins entre ces deux points, modulo dihomotopie.

[12] Mais cela doit être adapté à la nouvelle contrainte de non-réversibilité du temps!

[13] Légèrement transformée par rapport à [FGR99].

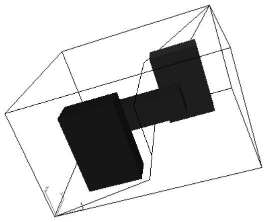


FIG. 1.7 – “Une chambre à trois trous” et deux dichemins non-dihomotopes.

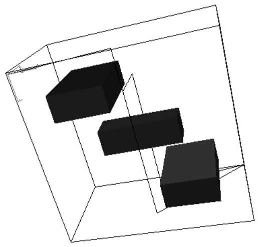


FIG. 1.8 – “Une chambre à trois trous” (autre vue).

Définition 6 *Soient f et g deux dichemins de X entre un point initial α et un point final β. Une dihomotopie entre f et g est une difonction de $\vec{I} \times I$ dans X telle que pour tout $x \in \vec{I}$, $t \in I$, $H(x,0) = f(x)$, $H(x,1) = g(x)$ et $H(0,t) = \alpha$, $H(1,t) = \beta$. On note alors $f \sim g$.*

Si l'on veut être plus général, il faut considérer des dichemins maximaux et non des dichemins entre un point initial et un point final. Ceci est partiellement développé dans [FGR99] mais pose encore un certain nombre de problèmes, en particulier pour les dichemins infinis (sur des po-espaces locaux qui ne sont pas compacts).

On a déjà vu que les classes d'équivalence de dihomotopie de dichemins sont moins nombreuses en général que les classes d'équivalence d'homotopie de chemins. Depuis l'article de V. Pratt [Pra91], y compris dans ma thèse [Gou95a], j'avais l'intuition qu'étudier les classes de *dihomotopie* de dichemins avec extrémités imposées, était équivalent à étudier les classes d'*homotopie* de dichemins avec extrémités imposées. En fait, cela n'est pas vrai. Il suffit pour cela de considérer l'exemple des figures 1.7 et 1.8 qui viennent

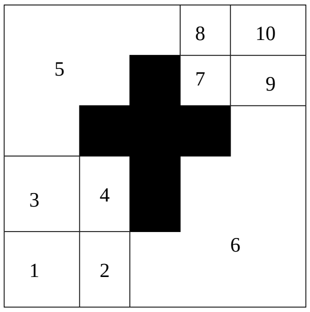


FIG. 1.9 – Le “drapeau Suisse”.

des termes[14],

```
#sem c 2
A=Pa.Pc.Va.Pb.Vc.Vb
B=Pa.Va.Pc.Vc.Pb.Vb
C=Pc.Vc
PROG=A|B|C
```

Les deux dichemins représentés sur ces figures sont homotopes mais pas dihomotopes. Il nous faut donc utiliser de nouveaux outils.

On a introduit dans [FGR99] la notion de “composantes diconnexes” pour étudier les classes de dihomotopie des dichemins. Cela est censé être le pendant des composantes connexes en topologie, mais comme la relation xRy s'il existe un dichemin de x à y n'est certainement pas une relation d'équivalence (et on ne veut pas la rendre symétrique, ce qui reviendrait à étudier des composantes connexes par arcs) cela est un peu plus compliqué.

Définition 7
1. *L'*historique homotopique *d'un dichemin maximal $\alpha : I \rightarrow X$ est*
$$h\alpha := \{ \quad y \in X | \exists \text{ un dichemin } \beta \text{ passant par } y \text{ et } \alpha \sim \beta \}$$
2. *Deux points sont* équivalents modulo leur historique homotopique *(“homotopy history equivalent”) si*
$$x \in h\alpha \Leftrightarrow y \in h\alpha \text{ pour tout } \alpha \in P_1(X).$$
3. *Les* composantes diconnexes *de X sont les composantes connexes (au sens classique du terme) des classes d'équivalence modulo l'historique homotopique de X.*

14 `#sem 2` veut dire que `c` est un 2-sémaphore. J'ai utilisé dans toute la suite la syntaxe reconnue par mon analyseur.

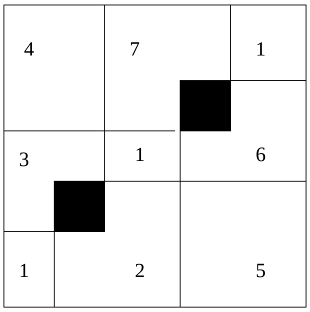


FIG. 1.10 – “Deux trous ordonnés”.

Par exemple, le complément du “drapeau Suisse” dans I^2 (voir figure 1.9) a 10 composantes diconnexes. Cet exemple est la sémantique du programme ayant pour processus parallèles $T_1 = Pa.Pb.Vb.Va$ et $T_2 = Pb.Pa.Va.Vb$ (où a et b sont des 1-sémaphores). Dans la région 1, on a tous les futurs possibles. Dans la région 2, on ne peut aller dans le futur que dans les régions 4 et 6, c'est-à-dire que le programme va arriver à un point mort ou que T_2 va accéder à a et b avant T_1. Dans la région 6, on ne peut venir que de 2 et aller à 9: T_2 accède à a et b avant T_1. Dans la région 9, on peut “venir” de la région non-atteignable 7 ou de 6. Dans la région 10, on peut être venu de n'importe quelle autre région (sauf la 4).

Le complément des “deux trous ordonnés” dans I^2 (voir figure 1.10, qui donne la sémantique de $Pa.Va.Pb.Vb \mid Pa.Va.Pb.Vb$) a 7 classes modulo l'historique homotopique. Une d'entre elles contient à la fois le point initial **0**, le point final **1**, et une région dans le centre de la figure. Cette classe se décompose donc en 3 composantes diconnexes.

L'exemple de la “chambre à 3 trous” dans I^3 (voir Ex. 1.7) a quant à lui 8 classes modulo l'historique homotopique. Une des classes (au centre) se décompose en deux composantes diconnexes.

Ce point de vue a en particulier permis de prouver que le protocole “2-phase locking” d'accès à des champs d'une base de données distribuées est correct (c'est-à-dire ici, séquentialisable). On pourra trouver cette preuve, faite par M. Raussen, basée sur les idées de [Gun94], dans notre article commun [FGR99]. Il est à noter que S. Sokolowski a défini dans [Sok99] un point de vue assez similaire à ces composantes diconnexes, mais en ne discriminant que le futur des dichemins. Cela peut s'avérer intéressant dans certaines situations (dont l'équivalence par bisimulation me semble-t-il). On pourra se reporter à ses autres travaux, très similaires à ce que l'on expose dans ce chapitre, [Sok98b], [Sok98c] et [Sok98a].

Le problème maintenant est de savoir comment calculer (éventuellement un sur- ou sous-ensemble) de ces régions dangereuses etc. On verra en détail des algorithmes pour déterminer les régions 4 (dangereuses) et 7 (inatteignables) de la figure 1.9 au chapitre 2. Je citerai également plus tard quelques travaux donnant des réponses partielles pour le reste des régions. Maintenant je discute brièvement de quelques idées d'approximation de ces régions que j'ai pu tester ces dernières années.

1.4 Invariants

1.4.1 Homologie

En topologie algébrique, il est bien connu que l'homotopie est une notion délicate. Il est dur en général de prouver que deux espaces topologiques sont homotopiquement équivalents, c'est-à-dire que l'un est une déformation “élastique” de l'autre. Les groupes d'homotopie, dont l'isomorphie est nécessaire et parfois suffisante pour décider de l'équivalence d'homotopie sont eux-mêmes durs à calculer. Les topologues algébristes ne savent par exemple toujours pas calculer les groupes d'homotopie de toutes les sphères de dimension quelconque (alors que ce sont des objets géométriques fondamentaux). De toutes façons, même pour des ensembles simpliciaux finis, la connexité est une propriété NP-complète et la connexité simple, c'est-à-dire le fait que le premier groupe d'homotopie soit trivial est non décidable; cela est dû au fait que le problème du mot dans les groupes quelconques est indécidable.

Il existe néanmoins des “invariants homotopiques” calculables. Un invariant homotopique est un foncteur qui à tout espace topologique X associe un objet mathématique $S(X)$ tel que si X et Y sont homotopiquement équivalents, $S(X)$ et $S(Y)$ sont isomorphes. En quelque sorte, ce sont des approximations sûres de l'équivalence d'homotopie, tout à fait dans le même sens qu'en interprétation abstraite.

Ma première idée, exprimée dans [GJ92], était qu'il valait mieux considérer des invariants homotopiques tels que l'homologie pour calculer les propriétés importantes des systèmes parallèles et distribués.

Pour commencer, on peut faire une première remarque, qui peut paraître un peu sybilline: au lieu de partir d'une sémantique de programmes parallèles sous forme d'ensembles semi-cubiques $M = (M_i)_{i \in \mathbb{N}}$, on peut utiliser des ensembles semi-cubiques "bi-gradués", c'est-à-dire des ensembles

$$N = (N_{p,q})_{p,q \in \mathbb{N}}$$

Les opérateurs bords début d_i^0 vont maintenant de $N_{p,q}$ vers $N_{p-1,q}$ et les opérateurs bords fin d_i^1 vont de $N_{p,q}$ vers $N_{p,q-1}$. Les ensembles $N_{p,q}$ ne sont disjoints que s'il n'y a pas de boucle dirigée dans l'automate de dimension supérieure.

L'observation cruciale est la suivante:

Lemme 1 *Considérons le diagramme suivant de R-modules (R étant un anneau intègre, par exemple $\mathbb{Z}$ ou $\mathbb{Z}/2\mathbb{Z}$ comme dans [GJ92]):*

$$\mathcal{A}(N) = \begin{tikzcd} \mathcal{A}(N_{p,q}) \arrow[r, "\partial_0"] \arrow[d, "\partial_1"'] & \mathcal{A}(N_{p-1,q}) \ldots \\ \mathcal{A}(N_{p,q-1}) \ldots & \vdots \end{tikzcd}$$

où $\mathcal{A}(N_{p,q})$ est le R-module libre engendré par $N_{p,q}$ et[15]

$$\partial_0 = \sum_{i=0}^{i=p+q-1} (-1)^i d_i^0$$

$$\partial_1 = \sum_{i=0}^{i=p+q-1} (-1)^i d_i^1$$

C'est un bicomplexe (faible) de module, c'est-à-dire qu'il vérifie les égalités $\partial_0 \circ \partial_0 = 0$, $\partial_1 \circ \partial_1 = 0$, et $\partial_0 \circ \partial_1 + \partial_1 \circ \partial_0 = 0$.

Par exemple, l'automate,

$$\begin{tikzcd} & \beta \arrow[dr, "a'"] & \\ 1 \arrow[ur, "b"] \arrow[dr, "a"'] & & \gamma \\ & \alpha \arrow[ur, "b'"'] & \end{tikzcd}$$

est représenté par le bicomplexe de $\mathbb{Z}$-modules,

$$\begin{tikzcd} & (a) \oplus (b) \arrow[r, "\partial_0"] \arrow[d, "\partial_1"'] & (1) \arrow[d, "\partial_1"] \\ (a') \oplus (b') \arrow[r, "\partial_0"] \arrow[d, "\partial_1"'] & (\alpha) \oplus (\beta) \arrow[r, "\partial_0"] \arrow[d, "\partial_1"'] & 0 \arrow[d, "\partial_1"] \\ (\gamma) \arrow[r, "\partial_0"] & 0 \arrow[r, "\partial_0"] & 0 \end{tikzcd}$$

avec $\partial_0(a) = \partial_0(b) = 1$, $\partial_1(a) = \partial_0(b') = \alpha$, $\partial_1(b) = \partial_0(a') = \beta$ et $\partial_1(a') = \partial_1(b') = \gamma$.

Les bicomplexes (ou complexes doubles de modules) sont des objets très importants en homologie, ils ont en effet de nombreuses propriétés.

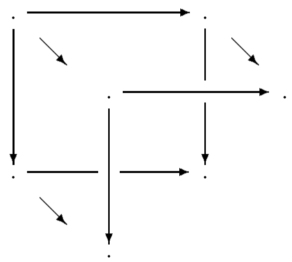
Posons,

- $H_i(N, \partial_0) = \frac{Ker\ \partial_0^i}{Im\ \partial_0^{i+1}}$
- $H_i(N, \partial_1) = \frac{Ker\ \partial_1^i}{Im\ \partial_1^{i+1}}$

où $Ker\ f$ (respectivement $Im\ f$) est le noyau (respectivement l'image) de l'application linéaire f. Ce sont les groupes d'homologie horizontale (respectivement verticale), qui permettent de déterminer les branchements (respectivement les confluences) des HDA.

Dans le cas de notre exemple, on trouve, $H_0(M, \partial_0) = (\alpha)$, $H_0(M, \partial_1) = (1)$, $H_1(M, \partial_0) = (b - a)$, $H_1(M, \partial_1) = (b' - a')$, et les autres groupes d'homologie sont nuls. Le générateur $(b - a)$ du groupe d'homologie horizontale de dimension un exprime le fait qu'il y a un choix non-déterministe entre les actions a et b. Le générateur du premier groupe d'homologie verticale $(b' - a')$ montre qu'il y a une confluence entre les actions a' et b'.

Un branchement typique en dimension deux est par exemple:



où les trois faces sont remplies.

Les foncteurs homologies ont de belles propriétés calculatoires (colimites, produit tensoriel, Mayer-Vietoris), ce qui permet de les calculer inductivement comme dans [Gou95a] où l'on déduit les groupes en question pour l'algèbre de processus CCS.

En particulier le foncteur homologie totale défini à partir de l'opérateur bord total $\partial_0 - \partial_1$ permet aussi (voir [Gou95a] et [Gou95b]) de

[15] Pour se souvenir de la dimension des objets sur lesquels agissent ∂_0 et ∂_1, on notera parfois ∂_0^{p+q} et ∂_1^{p+q}.

FIG. 1.11 – Cycles pour l'homologie des branchements.

définir une théorie homologique des dichemins. Cette théorie homologique permet de définir des invariants calculables de la dihomotopie, mais a les défauts suivants:

- Ce sont des invariants trop faibles de la dihomotopie. Par exemple ce foncteur n'arrive pas à séparer les deux dichemins de l'exemple de la figure 1.7.
- Les foncteurs branchement et confluence sont sensibles aux subdivisions: $a-b$ est un générateur de ce groupe d'homologie pour les deux exemples de la figure 1.11, et $a+c-b-d$ en est aussi un pour celui de droite, ce qui est en quelque sorte redondant.

J'ai pensé pendant longtemps que pour corriger ces défauts, il fallait définir une meilleure catégorie d'ensembles cubiques, en rajoutant des dégénérescences. Je connaissais les travaux de M. Evrard [Evr], mais ils ne me donnaient pas la solution. Je n'ai découvert que plus tard (grâce à P. Gaucher) les travaux de R. Brown et P. Higgins [BH81b] et [BH81a]. C'est finalement P. Gaucher qui a compris le rôle que devaient jouer, non seulement certaines dégénérescences supplémentaires mais aussi les compositions de cubes et qui a construit des invariants homologiques plus fins insensibles aux subdivisions, et dont je parle rapidement à la section 1.6.

La théorie que je proposais dans [Gou95b] définit bien sûr des groupes d'homologie de dimension supérieure à un. Le but était de pouvoir distinguer la forme de la figure 1.2 représentant un 2-sémaphore que cherchent à accéder trois processus, avec un 3-sémaphore dans la même situation. La différence entre un 1-sémaphore et un 2-sémaphore que cherchent à accéder 2 processus se remarque en examinant les classes de dihomotopie des dichemins: dans le premier cas, il y a nécessairement une exclusion mutuelle qui sérialise l'accès à la ressource partagée. Dans le second cas, il n'y a aucune sérialisation. Quand on prend des n-sémaphores avec $n > 1$, on ne peut plus distinguer leurs comportements par la différence de comportement entre deux actions à la suite, mais entre $n+1$ actions (accès) à la suite en général.

Pour les distinguer géométriquement, il faut examiner des hypersurfaces de dimension n modulo dihomotopie au lieu de dichemins modulo dihomotopie. Là encore, il faut faire attention à fixer les extrémités des hypersurfaces.

L'idée de [Gou95b] était simple et on peut la retrouver sous différentes formes dans les travaux plus récents de S. Sokolowski [Sok99] et de P. Gaucher [Gau00c]. Comme on peut le voir à la figure 1.12 (à gauche en dimension deux, à droite en dimension trois), en fixant deux chemins dihomotopes ayant les mêmes sources et buts, on peut considérer les surfaces qui sont le lieu d'une dihomotopie entre ces chemins. On dit alors que deux telles surfaces sont dihomotopes s'il existe une dihomotopie entre chacune des dihomotopies qui définissent ces surfaces. Dans la figure 1.12 les deux surfaces (l'une au dessus du trou, l'autre en dessous) ne sont pas déformables continûment l'une en l'autre, alors qu'elles le seraient si le cube était entièrement plein. Le biais pris dans [Gou95b] était homologique donc je considérais des surfaces avec bords fixés modulo l'homologie totale. Tout ceci peut se construire par récurrence en toute dimension en considérant certaines hypersurfaces de dimension n correspondants à des dihomotopies entre des hypersurfaces de dimension $n-1$, modulo dihomotopie. Si je reprends l'idée avec le formalisme plus moderne des po-espaces locaux, on sait déjà définir la dihomotopie entre objets de dimension 1 que sont les dichemins. Supposons maintenant que nous ayons défini les dihomotopies d'ordre au plus n ($n \geq 1$) entre les dihomotopies d'ordre $n-1$ avec extrémités fixées, qui forment alors l'ensemble $P_n(X)$[16]:

Définition 8 *Une dihomotopie d'ordre $n+1$ ($n \geq 1$) entre des dihomotopies H, G d'ordre n avec les mêmes extrémités est une difonction $A : \vec{I} \times I^{n+1} \to X$ telle que pour tout $x \in I^n \times \vec{I}$, $A(x,0) = H(x)$ et $A(x,,1) = G(x)$. Le début de A est $s_n(A) = H$ et son but est $t_n(A) = G$. On note $P_n(X)$ l'ensemble de telles dihomotopies.*

On peut définir également des compositions sur l'ensemble $P_n(X)$ ($n \geq 1$) comme suit: $*_{n-1} : P_n(X) \times P_n(X) \to P_n(X)$ est défini pour (f,g) tels que $t_n(f) = s_n(g)$:

$$(f *_{n-1} g)(x_0, \cdots, x_n) =$$

$$\begin{cases} 0 \leq x_n \leq \frac{1}{2} & f(x_0, \cdots, x_{n-1}, 2x_n) \\ \frac{1}{2} \leq x_n \leq 1 & g(x_0, \cdots, x_{n-1}, 2x_n - 1) \end{cases}$$

[16] Ceci n'étant pas publié encore à ce jour.

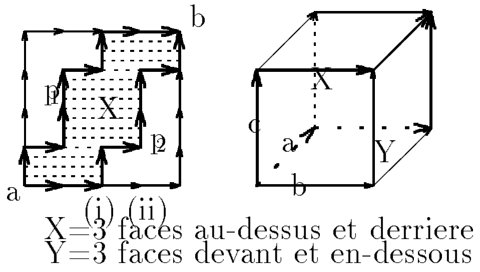


FIG. 1.12 – Deux surfaces ayant les mêmes bords non-dihomotopes.

Ce qui n'est pas clair maintenant, ce sont les points suivants, qui a ma connaissance ne sont pas résolus, aussi bien dans mon ancienne approche que dans celles de P. Gaucher et S. Sokolowski:

- Quelle est la structure algébrique de ces "ensembles dihomotopiques de dimension n", que l'on voudrait être un pendant des groupes d'homotopie de dimension n dans le cas classique?
- A-t'on la même information[17] sur le HDA étudié si on construit seulement les hypersurfaces particulières en dimension n supérieure ou égale à 2 que sont les "globes" c'est-à-dire les hypersurfaces entre une seule hypersurface de dimension $n-1$ et elle-même?

Pour le premier point il est cependant clair dans les travaux de P. Gaucher que la structure doit être celle d'une ω-catégorie, dont on introduit le concept dans la section 1.6, qui très probablement est un ω-groupoïde sauf en dimension 1. Donc pour le deuxième point, on obtiendrait naturellement des groupes d'homotopie en dimension supérieure ou égale à deux. A-t'on encore plus de structure en dimension supérieure ou égale à 3, comme par exemple la commutativité de l'opération de groupe[18]? Ceci reste pour le moment un problème ouvert.

1.4.2 Coupes intemporelles

Dans la section précédente, on a tenté de ramener le problème de la classification des dichemins modulo dihomotopie à un problème plus simple de classification modulo homologie.

[17] On vise évidemment ici une notion d'équivalence de dihomotopie qu'il reste à définir à ce jour.

[18] Comme c'est le cas quand on passe de la structure du groupe fondamental à la structure des groupes d'homotopie de dimension supérieure

Une autre idée naturelle s'impose [FGR99], celle de prendre des "clichés" instantanés et d'observer l'évolution dans le temps. En fait, j'y reviens à la section 2.4 car c'est la base de la méthode de M. Herlihy et de ses collaborateurs pour étudier la calculabilité et la complexité des protocoles distribués tolérants aux pannes.

Définition 9 *Soit U un ensemble muni d'un ordre partiel $\leq$. Un sous-ensemble $V \subset U$ est appelé* achronal *si pour tout $x, y \in V : x \leq y \Rightarrow x = y$ (similaire à la notion de [Pen72]).*

Définition 10 *Soit $(X, \leq)$ un po-espace.*

1. *On appelle $(X, \leq)$ po-espace* paramétré *s'il existe une difonction $F : X \rightarrow \mathbb{R}$ telle que $X_t := F^{-1}(t)$ soit achronal pour tout $t \in \mathbb{R}$.*
2. *On dit que $\leq$ est* Euclidien*, s'il existe un nombre fini de difonctions $f_i : X \rightarrow \mathbb{R}$ telles que*
$$\forall x, y \in X : \quad x < y \Leftrightarrow \forall i : \quad f_i(x) \leq f_i(y);$$
$$\exists i : \quad f_i(x) < f_i(y).$$
3. *Un ordre partiel local sur un espace topologique X est dit* paramétré*, respectivement* Euclidien *s'il (ou un de ses raffinements) est un po-espace paramétré, respectivement Euclidien.*

Un ordre partiel Euclidien est en fait une transcription de l'ordre naturel, composante par composante, sur un espace Euclidien de dimension finie $\mathbb{R}^n$: étant donné deux points $\mathbf{x} = [x_1, \ldots, x_n], \mathbf{y} = [y_1, \ldots, y_n] \in \mathbb{R}^n$,
$$\mathbf{x} \leq \mathbf{y} \Leftrightarrow \forall i : \quad x_i \leq y_i.$$

Dans un po-espace paramétré, on peut toujours reparamétrer les dichemins et les dihomotopies, de telle sorte que, pour un dichemin p, et $t \in \vec{I}$, $p(t) \in F^{-1}(t)$.

Soit $H : J \times I \rightarrow X$ une dihomotopie bien paramétrée entre deux dichemins bien paramétrés $\alpha, \alpha' : \vec{I} \rightarrow X$. Alors pour tout $t \in \vec{I}$, $\alpha(t)$ et $\alpha'(t)$ sont contenus dans la *même* composante connexe de X_t (qui est la "coupe à l'instant t de X").

Cela nous donne donc un moyen, par l'étude (topologie classique) des coupes, de déterminer des obstructions à la dihomotopie, donc de trouver un sous-ensemble des ordonnancements possibles. Malheureusement, cela ne nous donne qu'une approximation en général; soit X le sous-ensemble $[0,3] \times [0,3] \times [0,3] \backslash [1,2] \times [1,2] \times [0,3]$

de $\mathbb{R}^3$ avec l'ordre partiel naturel. Il y a deux classes de dihomotopie de $(0,0,0)$ à $(3,3,3)$, mais les coupes induites par $F(x,y,z) = x+y+z$ sont toutes connexes. Ainsi, pour obtenir toute l'information sur les classes de dihomotopies, il ne suffit pas de considérer une seule famille de coupes. En fait, en un certain sens (voir la section 1.6), il semble qu'il nous faille toutes les familles de coupes, dans le cas d'un ensemble semi-cubique. D'un point de vue informatique, cela revient à dire que certains systèmes asynchrones n'ont pas d'horloge globale.

1.5 Ensembles cubiques

Il existe un certain nombre de notions d'espaces semi-cubiques. On peut tout d'abord introduire les dégénérescences, utiles au moins dans les applications informatiques pour pouvoir considérer n'importe quelle n-transition comme une m-transition avec $m \geq n$ (où $m-n$ processeurs ne font rien), donc comme une sorte de transition paresseuse ("idle transition") comme dans les systèmes de transitions de [WN94].

Définition 11 *Un ensemble cubique K est un ensemble semi-cubique (K, ∂_i^α) qui a en plus des fonctions dégénérescences $\epsilon_i : K_{n-1} \rightarrow K_n$ $(0 \leq i \leq n-1)$ vérifiant les relations:*

$$\begin{array}{lcll} \epsilon_i\epsilon_j & = & \epsilon_{j+1}\epsilon_i & (i \leq j) \\ \partial_i^\alpha \epsilon_j & = & \left\{ \begin{array}{ll} \epsilon_{j-1}\partial_i^\alpha & (i<j) \\ \epsilon_j \partial_{i-1}^\alpha & (i>j) \\ Id & (i=j) \end{array} \right. & \end{array}$$

R. Brown et P. Higgins ont rajouté plus tard dans [BH81b] des dégénérescences spéciales appelées connections:

Définition 12 *K est un ensemble cubique avec connections s'il a en plus des fonctions appelées* connections $\Gamma^\alpha : K_{n-1} \rightarrow K_n$ *$(0 \leq i \leq n-2$, $\alpha = +,-)$ vérifiant les relations:*

$$\begin{array}{lcl} \Gamma_i^\alpha \Gamma_j^\beta & = & \Gamma_{j+1}^\beta \Gamma_i^\alpha \quad (i \leq j) \\ \Gamma_i^\alpha \epsilon_j & = & \left\{ \begin{array}{ll} \epsilon_{j-1}\Gamma_i^\alpha & (i<j) \\ \epsilon_j \Gamma_{i-1}^\alpha & (i>j) \\ \epsilon_j^2 = \epsilon_{j+1}\epsilon_j & (i=j) \end{array} \right. \\ \partial_i^\alpha \Gamma_j^\beta & = & \left\{ \begin{array}{ll} \Gamma_{j-1}^\beta \partial_i^\alpha & (i<j) \\ \Gamma_j^\beta \partial_{i-1}^\alpha & (i>j+1) \end{array} \right. \\ \partial_j^\alpha \Gamma_j^\alpha & = & \partial_{j+1}^\alpha \Gamma_j^\alpha = Id \\ \partial_j^\alpha \Gamma_j^{-\alpha} & = & \partial_{j+1}^\alpha \Gamma_j^{-\alpha} = \epsilon_j \partial_j^\alpha \end{array}$$

Maintenant, on peut définir des collages de n-cubes ou compositions, qui sont également nécessaires dans la modélisation informatique si on veut pouvoir considérer algébriquement des chemins (qui sont évidemment des collages de n-transitions ou n-cubes):

Définition 13 *K est un ensemble cubique avec connections et compositions si c'est un ensemble cubique avec connections et si en plus on a n opérations de composition en chaque dimension n, $+_j$ $(0 \leq j \leq n-1)$, telles que,*

Si $a, b \in K_n$ alors $a +_j b$ est défini ssi $\partial_j^0 b = \partial_j^1 a$. Quand les termes des égalités suivantes sont définis, on a:

$$\begin{array}{lcl} \partial_j^0(a +_j b) & = & \partial_j^0 a \\ \partial_j^1(a +_j b) & = & \partial_j^1 b \\ \partial_i^\alpha(a +_j b) & = & \left\{ \begin{array}{ll} \partial_i^\alpha a +_{j-1} \partial_i^\alpha b & (i<j) \\ \partial_i^\alpha a +_j \partial_i^\alpha b & (i>j) \end{array} \right. \\ (a +_i b) +_j & & \\ \ (c +_i d) & = & (a +_j c) +_i (b +_j d) \\ \epsilon_i(a +_j b) & = & \left\{ \begin{array}{ll} \epsilon_i a +_{j+1} \epsilon_i b & (i \leq j) \\ \epsilon_i a +_j \epsilon_i b & (i>j) \end{array} \right. \\ \Gamma_i^\alpha(a +_j b) & = & \left\{ \begin{array}{ll} \Gamma_i^\alpha a +_{j+1} \Gamma_i^\alpha b & (i<j) \\ \Gamma_i^\alpha a +_j \Gamma_i^\alpha b & (i>j) \end{array} \right. \\ \Gamma_j^+(a +_j b) & = & (\Gamma_j^+ a +_j \epsilon_j a) \\ & & +_{j+1}(\epsilon_{j+1} a +_j \Gamma_j^+ b) \\ \Gamma_j^-(a +_j b) & = & (\Gamma_j^- a +_j \epsilon_{j+1} b) \\ & & +_{j+1}(\epsilon_j b +_j \Gamma_j^- b) \end{array}$$

Une catégorie ω-cubique est un ensemble cubique avec connections et compositions tel que chaque $+_j$ donne une structure de catégorie à K_n, avec pour identités $\epsilon_j y$ $(y \in G_{n-1})$ et avec les conditions supplémentaires:

$$\Gamma_i^+ +_i \Gamma_i^- x = \epsilon_{i+1} x$$

$$\Gamma_i^+ +_{i+1} \Gamma_i^- x = \epsilon_i x$$

Je n'irai pas plus loin dans ce sens, qui nécessiterait néanmoins un travail approfondi. Il se trouve qu'une notion équivalente à celle de catégorie ω-cubique (voir [FBS00]) est celle de ω-catégorie, dont l'utilisation pour la modélisation informatique a été mise au point par P. Gaucher.

1.6 Point de vue ω-catégorique

L'idée remonte à l'article [Pra91], mais pendant longtemps j'ai trouvé cela trop compliqué pour m'y attaquer seul. Depuis, j'ai rencontré Philippe Gaucher (chercheur au CNRS en mathématiques à l'IRMA, ancien camarade de lycée!), puis Ronnie Brown, Tim Porter (Université de Bangor) et Richard Steiner (Université de Glasgow), experts en complexes cubiques, topologie algébrique.

Une ω-catégorie est une catégorie, avec des morphismes en toutes dimensions et des compositions correspondantes, cohérentes entre elles. L'idée pour la modélisation du parallélisme est que les objets, ou 0-morphismes représentent les états d'un automate de dimension supérieure, les 1-morphismes représentent tous les chemins possibles d'exécution[19] et les morphismes de dimension supérieure représentent les dihomotopies entre les morphismes de dimension inférieure[20]. En particulier les 2-morphismes représentent les dihomotopies entre les chemins d'exécution. Les lois de composition entre les morphismes de dimension supérieure formalisent les compositions des dihomotopies d'ordre supérieur. Comme V. Pratt l'avait déjà remarqué dans [Pra91], les axiomes des ω-catégories codent les propriétés des compositions des chemins et des dihomotopies dans un HDA. On trouve l'exploitation des ses idées dans [Gau00b], [Gau00a] et [Gau00d].

On va donner la définition formelle d'une ω-catégorie en trois étapes (voir [BH81c], [Str87] et [Ste91] pour plus de détails):

Une 1-catégorie est une paire $(A,(*,s,t))$ satisfaisant les propriétés suivantes:

1. A est un ensemble,
2. s et t sont des fonctions de A dans A appelées respectivement source et but,
3. pour tout $x,y \in A$, $x*y$ est defini dès que $tx = sy$,
4. $x*(y*z) = (x*y)*z$ dès lors que les deux membres de l'égalité sont bien définis,
5. $sx * x = x * tx = x$, $s(x*y) = sx$ et $t(x*y) = ty$ (donc $ssx = sx$ et $ttx = tx$).

Une 2-catégorie est un triplet

$$(A,(*_0,s_0,t_0),(*_1,s_1,t_1))$$

tel que,

1. les deux paires $(A,(*_0,s_0,t_0))$ et $(A,(*_1,s_1,t_1))$ sont des 1-catégories,
2. $s_0s_1 = s_0t_1 = s_0$, $t_0s_1 = t_0t_1 = t_0$, et pour $i \geq j$, $s_is_j = t_is_j = s_j$ et $s_it_j = t_it_j = t_j$ (Axiomes globulaires)
3. $(x *_0 y) *_1 (z *_0 t) = (x *_1 z) *_0 (y *_1 t)$ (Axiome de Godement)
4. si $i \neq j$, alors $s_i(x *_j y) = s_ix *_j s_iy$ et $t_i(x *_j y) = t_ix *_j t_iy$.

[19] Il ne faut pas oublier que cela doit rester un ensemble stable par composition, qui est alors la concaténation des chemins.

[20] Précisément celles introduites à la section 1.4.1.

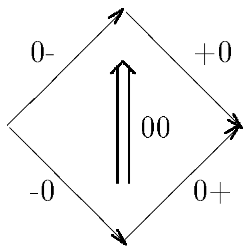


FIG. 1.13 – La ω-catégorie représentant une 2-transition.

Une ω-catégorie globulaire $\mathbb{C}$ est composée d'un ensemble A et d'une famille $(*_n,s_n,t_n)_{n\geq 0}$ telle que

1. pour tout $n \geq 0$, $(A,(*_n,s_n,t_n))$ est une 1-catégorie
2. pour tout $m,n \geq 0$ avec $m < n$, $(A,(*_m,s_m,t_m),(*_n,s_n,t_n))$ est une 2-catégorie
3. pour tout $x \in A$, il existe $n \geq 0$ tel que $s_nx = t_nx = x$ (le plus petit de ces n est appelé la dimension de x).

Un élément n-dimensionnel de $\mathbb{C}$ est appelé n-morphisme. Un 0-morphisme est aussi appelé un état de $\mathbb{C}$, et un 1-morphisme, une flèche. Si x est un morphisme d'une ω-catégorie $\mathbb{C}$, on appelle $s_n(x)$ la n-source de x et $t_n(x)$ le n-but de x. La catégorie de toutes les ω-catégories est notée ωCat. Les morphismes correspondants sont appelés des ω-foncteurs.

Maintenant, on peut donner, comme dans [Gau00c] quelques exemples de ω-catégories venant d'ensembles cubiques, ou de HDA. Par exemple, l'automate engendré par une unique 2-transition est représenté comme une ω-catégorie à la figure 1.13[21].

P. Gaucher dans [Gau00d] et [Gau00c] utilise la ω-catégorie engendrée par un ensemble cubique, pour construire trois théories homologiques correspondant respectivement aux branchements confluences et globes (ou exclusions mutuelles). Il les construit comme des foncteurs nerfs, respectivement de branchement, confluence et globulaire. C'est possible car les ensembles simpliciaux peuvent se représenter par des ω-catégories, grâce au résultat de [Str87]. En quel-

[21] La flèche double est un 2-morphisme dans la ω-catégorie correspondante. Cela est du en fait au résultat de [Ait86].

que sorte, les nerfs branchement et confluence décrivent simplicialement toutes les coupes intemporelles des automates, comme indiqué à la section 1.4.2.

Ces constructions ont de multiples avantages sur celles de ma thèse:

- Ils sont bien plus précis (l'exemple de la chambre à trois trous devrait pouvoir être complètement décrit par ces homologies),
- En ce qui concerne les branchements et les confluences, ils n'ont pas le problème identifié dans la section 1.4.1, c'est-à-dire qu'ils ne sont pas sensibles à la subdivision des actions.

Deux autres points importants doivent être mentionnés.

A partir de ces constructions, les ω-catégories semblent pouvoir donner la bonne structure aux "ensembles de dihomotopies". L'équivalence de catégorie entre la catégorie des catégories cubiques avec connections et compositions et la catégorie des ω-catégories récemment démontrée dans [FBS00] me conforte dans cette idée. De plus, les travaux de R. Brown et de P. Higgins (voir [BH81b] et [BH81a]) me portent à espérer avoir également un théorème de van Kampen, très certainement affaibli. (j'en dis un mot à la section 1.8).

1.7 Rapports entre les modèles introduits

Dans [Gou95a], j'ai décrit un certain nombre de relations formelles entre les automates de dimension supérieure et d'autres modèles du parallélisme. Je n'y reviens pas.

J'explicite ici les rapports entre présentations topologique, catégorique et combinatoire, extraits de [FGR99].

Soit $\square_n$ le cube "standard" dans $\mathbb{R}^n$ $(n \geq 0)$,

$$\square_n = \{(t_1, \ldots, t_n) | \forall i, 0 \leq t_i \leq 1\}$$

$$\square_0 = \{0\}$$

et soient $\delta_i^k : \square_{n-1} \rightarrow \square_n$, $1 \leq i \leq n$, $k = 1, 2$, les fonctions continues $(n \geq 1)$,

$$\begin{tikzcd} \square_n & \square_{n-1} \arrow[l, "\delta_i^0"'] \\ \square_{n-1} \arrow[u, "\delta_i^1"] & \end{tikzcd}$$

définies par,

$$\delta_i^k(t_1, \ldots, t_{n-1}) = (t_1, \ldots, t_{i-1}, k, t_i, \ldots, t_{n-1})$$

Considérons maintenant, étant donné un ensemble semi-cubique M, l'ensemble $\mathbf{R}(M) = \coprod_n M_n \times \square_n$. Les ensembles M_n peuvent être considéré comme des espaces topologiques avec la topologie discrète et $\square_n$ hérite de la topologie de $\mathbb{R}^n$. Ainsi $\mathbf{R}(M)$ est un espace topologique avec la topologie de l'union disjointe. Soit maintenant $\equiv$ la relation d'équivalence induite par les identités:

$$\forall k, i, n, \forall x \in M_{n+1}, \forall t \in \square_n, n \geq 0,$$

$$(\partial_i^k(x), t) \equiv (x, \delta_i^k(t))$$

Soit $| M | = \mathbf{R}(M)/\equiv$ avec la topologie quotient. L'espace topologique $| M |$ est appelé la *réalisation géometrique* de M.

On a besoin maintenant de trouver comment interpréter l'orientation du complexe semi-cubique en termes d'ordres partiels locaux. Pour ce faire, on doit[22] d'abord se restreindre à des cas pas trop pathologiques:

Définition 14 *Soit M un ensemble semi-cubique. On dit que M n'est pas* auto-lié *("non self-linked") si pour tous ses n-cubes x, $\partial_l^k(x) = \partial_{l'}^{k'}(x)$ implique $k = k'$ et $l = l'$.*

On pose alors

$$(x, t) \leq_{U^x} (y, u) \text{ si } \delta_{l_i}^{k_i} \ldots \delta_{l_1}^{k_1}(t) \leq u \text{ dans } \square_{n+i}$$

$$(y, u) \leq_{U^x} (x, t) \text{ si } \delta_{l_i}^{k_i} \ldots \delta_{l_1}^{k_1}(t) \geq u \text{ dans } \square_{n+i}$$

Appelons étoile d'un élément x de M

$$St(x, M) = \{y \mid d_{l_1}^{k_1} \ldots d_{l_u}^{k_u} y = x\}$$

(voir par exemple [Spa66]). Soient x une arête de M et (z, v) un point de U^x dont le support est z. On pose $(z, v) \leq_x (y, u)$ si il existe b dans l'étoile de x et t tels que $(z, v) \leq_{U^b} (b, t) \leq_{U^b} (y, u)$.

C'est bien un ordre partiel; la seule difficulté de la preuve réside dans la transitivité, voir figure 1.14. En effet, aussi bien à droite qu'à gauche de la figure, on a $z \leq_b y$ et $y \leq_{b'} a$ mais à gauche, on a $z \leq_x a$ et à droite $z \leq_c a$ où b est l'arête allant en profondeur à partir de x.

Alors, la réalisation géométrique d'un ensemble semi-cubique non auto-lié M définit un po-espace local avec l'atlas $\{\mathrm{St}(x, M)/x \in M_0\}$ et

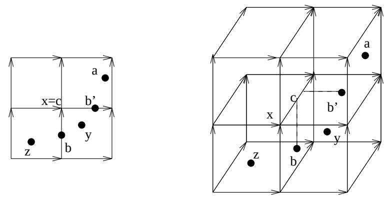


FIG. 1.14 – Illustration de la transitivité de $\leq_x$.

les ordres partiels locaux $\leq_x$ sur chaque St(x, M).

La réalisation géométrique est fonctorielle, je passe les détails ici. On a également un adjoint (à droite) de la réalisation géométrique: prenons $(M, \leq)$ un po-espace local. Soit $S(M)$ l'espace topologique gradé tel que, pour tout $n \in \mathbb{N}$, $S(M)_n$ est l'ensemble des n-dicubes singuliers de M, c'est-à-dire les difonctions du n-dicube standard vers M. On peut définir les opérateurs ∂_l^k avec $\partial_l^k(f) = f \circ \delta_l^k$. Ceux-ci donnent à $S(M)$ la structure d'un ensemble semi-cubique.

La correspondance entre les propriétés homotopiques dans les cas topologiques et combinatoires (cubiques) est heureusement agréable ce qui fait que l'on peut raisonner aussi bien géométriquement sur les po-espaces locaux (par exemple les graphes d'avancement) qu'algébriquement ou combinatoirement sur les espaces cubiques (donc également homologiquement). On a pu pour le moment au moins le prouver dans un cas simple, la dimension 2, voir [FGR99]. J'ai de bonnes pistes pour généraliser le résultat.

Soit N un ensemble semi-cubique. Un dichemin dans N est une suite $p = (p_1, \cdots, p_k)$ d'éléments de N_1 tels que pour tout i, $1 \leq i < k$, $d_1^1(p_i) = d_1^0(p_{i+1})$. $d_1^0(p_1)$ est le point initial de p. $d_1^1(p_k)$ est le point final de p.

La réalisation géométrique d'un dichemin combinatoire p de M induit un dichemin (topologique) $\mid p \mid$ dans $\mid M \mid$.

On dit que deux dichemins combinatoires sont dihomotopes si on peut passer de l'un à l'autre par un nombre fini de commutations locales de deux actions consécutives. C'est exactement la même notion que celle de "commutativité partielle" des traces de Mazurkiewicz [Maz88].

Encore une fois, toute dihomotopie combinatoire entre p et q dans M induit une dihomotopie (topologique) entre $\mid p \mid$ et $\mid q \mid$.

On a aussi l'inverse souhaité. Soient L un ensemble semi-cubique fini et h un dichemin de $\mid L \mid$ (c'est-à-dire un dichemin de $\square_1$ vers $\mid L \mid$). Alors il existe une "approximation cubique" f : $S_k \to L$ de h, où S_k est une subdivision de $\vec{I}$. De plus f definit un dichemin (combinatoire) $(f(u_1), \ldots, f(u_k))$ que l'on appelle $\tilde{f}$. Enfin $\mid f \mid$ est homotope à $\mid \tilde{f} \mid$ dans $\mid L \mid$.

Je suis en train, avec P. Gaucher, de mettre au point la relation entre sa présentation ω-catégorique et les po-espaces locaux. Nous sommes confiants dans le résultat, étant donné l'article de M. M. Kapranov et V. A. Voevodsky [KV91] qui établit l'équivalence de l'approche topologique et de l'approche ω-groupoïdale dans le cadre de la topologie algébrique classique.

1.8 Quelques autres idées

1.8.1 Compositionalité et van Kampen

L'idée que j'essaie de creuser en ce moment, est celle d'une définition d'un objet mathématique qui contienne l'essentiel de l'information des dichemins modulo dihomotopie d'un espace localement partiellement orienté, et qui soit calculable compositionnellement, c'est-à-dire à partir d'une décomposition de l'espace en l'union de deux sous-espaces. Cela rendrait l'analyse de certains systèmes plus aisée. En fait, on peut construire le pendant d'un groupoïde fondamental, qui ne sera en fait qu'une catégorie fondamentale[23]. C'est évidemment relié aux constructions des régions diconnexes, et aussi aux constructions récentes de S. Sokolowski, voir [Sok99], mais il reste encore à comparer formellement toutes ces approches.

Soit $\mathcal{X} = (X, \mathcal{U}, (\leq_U)_{U \in \mathcal{U}})$ un espace localement partiellement orienté.

On peut définir une opération de composition entre certains dichemins de X, allant de $P_1^{\alpha,\beta}(X) \times P_1^{\beta,\gamma}(X)$ vers $P_1^{\alpha,\gamma}(X)$:

Définition 15 *Soit $f \in P_1^{\alpha,\beta}(X)$, $g \in P_1^{\beta,\gamma}(X)$. On définit $h : \vec{I} \to X$ comme suit: pour $x \in \vec{I}$,*

$$h(x) = \begin{cases} f(2x) & si\ 0 \leq x \leq \frac{1}{2} \\ f(2x-1) & si\ \frac{1}{2} \leq x \leq 1 \end{cases}$$

Aors $h \in P_1^{\alpha,\gamma}(X)$.

[22] Cette restriction sera sans doute bientôt levée, semble-t-il.

[23] C'est la traduction sur les po-espaces locaux de ce que j'avais défini comme le groupe d'homotopie orienté "total" sur les bicomplexes, dans [Gou95a].

Ce n'est pas une opération commutative ni associative en général, de même qu'en topologie "non-dirigée". Par contre, encore une fois comme en topologie algébrique classique, cette opération s'étend aux classes de dihomotopie des dichemins, et devient alors associative.

On peut alors définir la catégorie suivante $\mathcal{C}(X)$:

- ses objets sont les points de X,
- ses morphismes sont les classes de dihomotopie des dichemins: un morphisme de x vers y est une classe de dihomotopie $[f]$ d'un dichemin f allant de x vers y.

La composition que l'on a définie plus haut est bien une opération associative avec identités (les classes de dihomotopie de dichemins constants), on l'utilise comme composition de morphismes. En fait, cette construction définit même un foncteur de la catégorie des espaces localement partiellement orientés vers la catégorie des catégories. En effet, soit $f : X \to Y$ une difonction d'un espace localement partiellement orientés X vers un espace localement partiellement orienté Y. On définit $\mathcal{C}(f)$ comme un morphisme de $\mathcal{C}(X)$ vers $\mathcal{C}(Y)$:

- sur les objets x de $\mathcal{C}$, $\mathcal{C}(f)(x) = f(x)$,
- sur les morphismes $[\omega]$ de $\mathcal{C}$, avec ω un dichemin, $\mathcal{C}(f)([\omega]) = [f \circ \omega]$.

Alors, j'ai pensé que l'on aurait sans doute un équivalent au théorème de van Kampen sur les groupoïdes fondamentaux:

Soit X un espace localement partiellement orienté, X_1 et X_2 deux espaces localement partiellement orientés tels que,

- $X = \overset{\circ}{X_1} \cup \overset{\circ}{X_2}$,
- tous les chemins continus sur $X_1 \cap X_2$ peuvent être décomposés en un nombre fini de dichemins.

Soit $j_1 : X_1 \cap X_2 \to X_1$ (respectivement $j_2 : X_1 \cap X_2 \to X_2$) et $i_1 : X_1 \to X$ (respectivement $i_2 : X_2 \to X$) les morphismes d'inclusion canoniques. Alors le diagramme suivant dans la catégorie des catégories,

$$\begin{tikzcd}
\mathcal{C}(X_1 \cap X_2) \arrow[r, "\mathcal{C}(j_1)"] \arrow[d, "\mathcal{C}(j_2)"'] & \mathcal{C}(X_1) \arrow[d, "\mathcal{C}(i_1)"] \\
\mathcal{C}(X_2) \arrow[r, "\mathcal{C}(i_2)"'] & \mathcal{C}(X)
\end{tikzcd}$$

est co-cartesien.

Apparemment (communication privée de M. Raussen), ce serait faux. Peut-être peut-on au moins espérer un résultat similaire mais avec des hypothèses plus fortes sur les sous-espaces X_1 et X_2?

1.8.2 Théorie des domaines

De fait, je n'ai réalisé que depuis peu qu'il y a des connections fortes entre un cas particulier du modèle topologique dont j'ai parlé dans la section 1.3, et la théorie des domaines (en particulier le chapitre VII de [Joh82]), ou d'autres considérations topologiques anciennes (le livre [Nac65]). Cela a fait l'objet en particulier d'une présentation au séminaire Dagstuhl 00231 "Topology in Computer Science", pour lequel j'espère envoyer une soumission. Je résume les considérations qui font l'objet de ce projet d'article dans cette section.

Pour des raisons qui restent encore obscures à mes yeux, L. Nachbin dans [Nac65] a étudié des espaces topologiques particuliers, les espaces ordonnés compacts Hausdorff. En fait, il ne s'agit de rien d'autre que de po-espaces compacts. C'est le cas des graphes d'avancement finis. Un des résultats particulièrement intéressant est qu'il existe une paire de foncteurs adjoints entre ces po-espaces compacts et un certain type d'espace topologique, les espaces stablement-compacts.

On notera PO la catégorie des po-espaces compacts avec pour morphismes ceux que l'on a déjà défini sur les po-espaces. Maintenant, on définit les espaces stablement-compacts:

Définition 16 *Un espace topologique stablement-compact est un ensemble X muni d'une topologie τ (c'est-à-dire un ensemble de sous-ensembles ouverts de X) tel qu'il existe une topologie τ^* sur X avec,*

- *$\tau \cup \tau^*$ est compact,*
- *pour tous $x \neq y$ dans X, il existe un élément $O \in \tau$, un élément $O^* \in \tau^*$ tels que $x \in O$, $y \in O^*$ et $O \cap O^* = \emptyset$.*

En quelque sorte, (X, τ) est compact Hausdorff *avec l'aide de la topologie τ^**.

On note SK la catégorie dont les objets sont les espaces topologiques stablement-compacts et dont les morphismes sont les fonctions continues.

Proposition 1 *Soit $(X, \tau, \leq)$ un po-espace Hausdorff compact (τ est la topologie, $\leq$ est l'ordre partiel). Construisons (X', τ') un espace topologique à partir de $(X, \tau, \leq)$ comme suit:*

– $X' = X$,
– τ', *l'ensemble des ouverts, est composé des éléments* U *de* τ *qui sont tels que* $\forall x \in U$, $\forall y \geq x$, $y \in U$ *("ensembles supérieurs").*

Alors (X', τ') *est un espace stablement-compact.*

SCHÉMA DE PREUVE. C'est une conséquence du théorème de convexité locale (voir [Nac65] ou [Joh82], Théorème 1.4, Chapitre VII, page 272) qui dit que les ensembles de la forme $U \cap V$, où U est un ouvert supérieur et V est un ouvert inférieur, forment une base pour la topologie de X. Il suffit alors de prendre pour τ^* (requis par la définition 16) l'ensemble des ouverts inférieurs. L'axiome de "séparation" faible de la définition 16 est le corollaire 1.2, Chapitre VII, page 271 de [Joh82].

Bien sûr, en général, (X', τ') n'est pas du tout Hausdorff, donc n'est pas un objet "géométrique" standard.

Les difonctions entre des po-espaces compacts Hausdorff sont évidemment mappées sur des fonctions continues entre leurs espaces stablement-compacts correspondants.

Le foncteur α de $\mathcal{PO}$ vers $\mathcal{SK}$ a un adjoint à droite que l'on peut maintenant décrire:

Définition 17 *Soient* (X, τ) *un espace topologique,* A *l'ensemble de ses ouverts compacts (c'est-à-dire les ouverts de* A *qui sont compacts en tant que sous-espaces topologiques de* X*),* A^* *l'ensemble des compléments (dans* $\wp(X)$*) des éléments de* A*. La topologie "patch" sur* X *est la topologie* $\kappa(\tau)$ *ayant pour base* $C = \{U \cap V \mid U \in A, V \in A^*\}$.

Proposition 2 *Soit* (Y, σ) *un espace stablement compact. On peut lui associer une structure*

$$(X, \tau, \leq) = \gamma(Y, \sigma)$$

avec,

– $X = Y$,
– $\tau = \kappa(\sigma)$,
– *pour tous* $x, y \in X$, $x \leq y$ *si pour tous* $U \in \sigma$ *avec* $x \in U$, y *est aussi dans* U.

Alors $(X, \tau, \leq)$ *est un po-espace compact Hausdorff.*

Qui plus est, γ définit un foncteur de SK vers PO (transformant les fonctions continues de SK en des difonctions de PO).

Considérons maintenant une dihomotopie H entre deux dichemins f et g avec les mêmes débuts et fins dans X. C'est tout simplement une difonction de $I \times \vec{I}$ vers X telle que $H(0, .) = f$ et $H(1, .) = g$.

Ainsi $\alpha(H)$ est une fonction continue de $\alpha(f)$ vers $\alpha(g)$, qui sont elles-mêmes des fonctions continues de $\alpha(I \times \vec{I})$ vers $\alpha(X)$. Mais α est un foncteur adjoint à gauche, donc commute avec les petites limites, en particulier les produits cartésiens. Donc $\alpha(I \times \vec{I}) = \alpha(I) \times \alpha(\vec{I})$. $\alpha(I) = I$ car tout sous-ensemble de I est supérieur. On en conclut que $\alpha(H)$ est une homotopie (classique) entre $\alpha(f)$ et $\alpha(g)$. On a donc prouvé un critère d'obstruction à la dihomotopie:

Proposition 3 *Soient* f *et* g *deux dichemins dans le po-espace compact Hausdorff* X*. Alors* $\alpha(X)$ *et* $\alpha(Y)$ *ne sont pas homotopes (au sens classique du terme) implique que* f *et* g *ne sont pas dihomotopes.*

La question qui se pose naturellement est de savoir s'il peut y avoir une réciproque à ce résultat; soit H' une homotopie entre $\alpha(f)$ et $\alpha(g)$. A quelle condition existe-t-il une dihomotopie H entre f et g? A-t-on de surcroit $\alpha(H) = H'$?

Une première idée pour étudier les homotopies entre $\alpha(f)$ et $\alpha(g)$, étant donnés f et g deux dichemins dans un po-espace compact Hausdorff X, est d'étudier les homomorphismes de groupes entre les groupes d'homotopies de $\alpha(\vec{I})$ vers les groupes d'homotopie correspondants de $\alpha(X)$. Tout d'abord, on peut voir assez facilement que $\alpha(\vec{I})$ est un espace topologique compact et connexe.

Comme $\alpha(\vec{I})$ est connexe, on peut définir son groupe fondamental $\pi_1(\alpha(\vec{I}))$ en choisissant n'importe quel point base, par exemple 0. Malheureusement, en étudiant les fonctions continues de I to $\alpha(\vec{I})$, qui sont les fonctions semi-continues inférieurement, on s'aperçoit que $\alpha(\vec{I})$ est un espace topologique simplement connexe.

Donc il n'y a aucun homomorphisme de groupe intéressant de $\pi_1(\alpha(\vec{I}))$ vers $\pi_1(\alpha(X))$.

Pire encore, on peut montrer que l'on peut déformer continûment (pour la topologie de $\alpha(\vec{I} \times \vec{I})$) n'importe quel chemin continu (pour la topologie de $I \times I$) maximal en n'importe lequel autre, en passant par des chemins discontinus, pour la topologie de $I \times I$; ce qui implique que l'homotopie des fonctions de $\alpha(\vec{I})$ vers $\alpha(\vec{I} \times \vec{I})$ ne permet même pas de distinguer la présence d'un trou dans la surface $I \times I$!

Mais il est possible que l'on arrive à des résultats bien meilleurs si l'on était parti non pas de l'ordre partiel $\prec$ mais de l'ordre partiel $<$ (accessibilité par des dichemins) sur un graphe d'avancement, mais ceci reste encore à examiner. L'autre question importante est de savoir s'il existe un pendant aux espaces localement partiellement ordonnés. S. Vickers examine cette possibilité avec un de ses étudiants à Milton Keynes.

1.8.3 Autres problèmes

Il existe un certain nombre d'autres travaux "géométriques" s'appliquant à des modèles du calcul. En particulier, il y a les travaux de F. Métayer, [Mét94] et [Mét95] et de K. Basu [Bas97a] et [Bas97b] qui, me semble-t-il, peuvent avoir un rapport avec ce que je tente de faire pour le parallélisme.

Ce qui me paraît plus proche de façon immédiate est le rapport avec les systèmes de réécriture, et en particulier le théorème de Squier [Squ87]. J'y ai pensé dès ma thèse mais je n'ai hélas pas eu l'occasion de me pencher dessus à nouveau récemment. Je présente néanmoins quelques idées sur le sujet, car je crois que la formulation ω-catégorique nous permettra peut-être d'aider à comprendre le phénomène.

Un monoïde $(M, 1, .)$ est finiment présenté (respectivement, par un système de réécriture canonique) s'il existe un ensemble fini de symboles S et un ensemble fini R de relations (respectivement de règles orientées) sur S^* tels que $M \cong S^*/R$ (respectivement tel que M soit isomorphe à S^* quotienté par la congruence engendrée par R). On dit de plus qu'un monoïde $(M, 1, .)$ a un problème de mot décidable s'il existe un algorithme (qui termine toujours) permettant de décider si deux mots de M sont égaux. Le théorème de Squier prouve que tous les monoïdes finiment présentés avec un problème de mot décidable ne peuvent pas forcément être présentés par un système de réécriture canonique fini. En fait, il dit plus: si M peut être présenté par un système de réécriture canonique fini, alors le troisième groupe d'homologie de M est de rang fini. Plus tard, on a remarqué qu'en fait tous les groupes d'homologie devaient être de rang fini (voir par exemple [Kob90], et [SOK94]).

J'avais remarqué dans ma thèse que tout cela ressemblait fortement à un problème d'homotopie dirigée (par exemple la résolution construite dans [Gro91] est un espace cubique). En fait, l'explication de [SOK94] et surtout celle plus récente donnée avec des 2-catégories dans [Laf95] me laisse à penser que l'approche ω-catégorique (et aussi l'approche topologique sans doute) doivent nous permettre d'obtenir des caractérisations plus précises de ce qu'est un monoïde présenté par un système de réécriture canonique fini. En effet, dans le cas du parallélisme on se contente de caractériser par dihomotopie certains monoïdes "partiellement commutatifs" (comme dans la théorie de Mazurkiewicz), c'est-à-dire où les seules relations possibles sont des relations de commutations particulières (un peu plus compliquées dans le cas de n-sémaphores, $n > 1$, où il faut considérer des cycles de longueur $n+1$). Dans le cas de [Laf95], les relations du monoïde sont présentées par des 2-cellules dans une 2-catégorie. J'espère pouvoir revenir plus tard sur ce sujet.

En attendant, ce qui m'a plus préoccupé, en rapport avec la logique et la réécriture, c'est l'étude, initiée par mon frère Jean Goubault-Larrecq, de la logique modale S4, et de ses preuves. En fait, on peut remarquer que la modalité $\Box$ de S4 engendre une comonade. Or, une comonade engendre naturellement un espace simplicial augmenté (c'est la résolution Bar "abstraite"), voir [ML71]. Il est alors intéressant d'examiner comment les preuves S4 transforment les espaces simpliciaux augmentés. C'est l'objet de notre article [GLG99] (présenté mais non encore publié). Nous sommes en train de réfléchir à des extensions de ce travail, car le phénomène semble général: selon la comonade choisie, on obtient des espaces simpliciaux augmentés différents qui ont chacun leur intérêt.

Chapitre 2

Analyse statique de programmes parallèles

La communauté des sémanticiens a redécouvert (après les travaux sur les graphes d'avancement, essentiellement le fait d'algorithmiciens) les considérations géométriques que je viens de développer.

Cette redécouverte date de l'opposition entre les partisans des sémantiques "vraiment parallèles" et les partisans de sémantiques par entrelacement. La base du contentieux est que, dans une sémantique par entrelacement, on représente le parallélisme par le mélange de toutes les actions en parallèle, c'est-à-dire par une simulation sur un processeur. Cela implique de confondre parallélisme et choix non-déterministe, donc avec l'exclusion mutuelle également[1].

Certains tenants des sémantiques par entrelacement prétendent que l'asynchronie n'existe pas réellement. Mon point de vue est - en me faisant un peu l'avocat du diable - exactement le contraire: la synchronie est physiquement non observable en général, mais est une bonne approximation dans un certain nombre de cas pratiques. En tous les cas, il y a des problèmes pratiques pour lesquelles on aurait du mal à faire des hypothèses synchrones, comme par exemple pour de vrais systèmes distribués (internet par exemple), pour lesquelles il est peu probable qu' on puisse un jour disposer d'une horloge globale, accessible de façon simultanée par tous.

La sémantique par entrelacement crée de plus un certain nombre de problèmes pratiques pour l'analyse statique. Les entrelacements construisent un grand nombre d'états qui sont complètement inintéressants pour la sémantique et la vérification (il ne s'y passe rien). Par exemple, quand toutes les actions de deux processus en

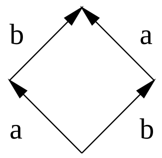


FIG. 2.1 – $a \mid b$ (entrelacement

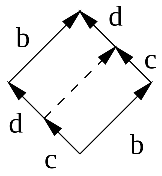


FIG. 2.2 – Un raffinement de $a \mid b$

parallèle sont indépendantes, il suffit de considérer l'exécution de l'un d'eux avant l'autre et non tous les entrelacements possibles pour comprendre le système[2].

Ce problème est connu dans la littérature sous le sobriquet "explosion combinatoire". Il existe des techniques pour éviter cela autant que faire se peut, la plupart d'entre elles étant dérivées de considérations sémantiques "vraiment parallèles". Par exemple la méthode des "stubborn sets" de [Val90] vient de considérations sur les réseaux de Pétri, et celle des "persistent sets" de [GW91] est basée sur la théorie des traces de Mazurkiewicz.

Un autre problème est celui du *raffinement*, qui est une façon assez commode d'imaginer faire de l'analyse de programmes (ou plutôt sans doute de spécifications, à la SDL). Comme cela est en partie décrit dans [vGG89], c'est assez difficile à appliquer dans des sémantiques par entrelacement. Supposons par exemple que l'action a de la figure 2.1 ne soit en fait pas atomique, et soit composée des deux sous-actions c et d. Alors la sémantique par entrelacement de $a \mid b$ n'est plus équivalente à a puis b ou b puis a. Le

[1] Voir à ce propos la figure 2.1 qui code en même temps $a \mid b$ et $a.b + b.a$.

[2] Le lecteur avisé sent déjà poindre la notion d'homotopie.

chemin c, b puis d manque (voir la figure 2.2, la partie manquante étant représentée par la ligne en pointillé).

Cela implique qu'il faut avant même toute analyse statique basée sur une sémantique par entrelacement, poser comme hypothèse l'atomicité de toutes les actions potentiellement présentes. Cela peut rendre les choses très lourdes. On peut remarquer à ce propos la similairité avec le problème du choix de la "base de temps" dans les sémantiques opérationnelles de systèmes temps-réel[3].

En fait, le raffinement s'exprime géométriquement comme une subdivision comme on le voit dans les figures 2.1 et 2.2[4].

Il faut noter que les 2-transitions ne sont rien d'autre que des relations de commutation locales, comme dans la théorie des traces de Mazurkiewicz [Maz86], ou encore des relations d'indépendance, comme dans les systèmes de transitions asynchrones [Bed88] et [Shi85], ou dans les automates de traces [Kah74, KM77], ou encore dans les systèmes de transition avec indépendance [SNW94], ou indirectement avec la relation de "confluence" des systèmes de transition parallèles [Sta89].

Il y a néanmoins deux ingrédients supplémentaires aux HDA: l'élégance et le puissance des outils de formalisation géométrique (ainsi que l'intuition qu'elle peut dégager), et la généralisation naturelle à des phénomènes de dimension supérieure (c'est-à-dire, on l'a déjà vu, à des relations d'indépendance n-aires).

Dans [Gou93] j'ai essayé de montrer comment on peut modéliser des "vrais" programmes (pas seulement des P et des V) avec des HDAs. Je ne suis plus persuadé que la présentation que j'en avais faite soit très pratique. Il faudrait sans doute retravailler cela, peut-être dans le style de la sémantique de [FS00].

[3] A ce propos, la géométrie là encore permet d'être bien plus naturel. J'avais décrit mes premières idées sur ce sujet dans [Gou96c], mais idéalement, il faudrait les retravailler pour utiliser une version des po-espaces locaux. Il est clair que la structure topologique continue doit là être utilisée, surtout qu'elle colle parfaitement aux découpages combinatoires que l'on peut en faire en choisissant une base de temps, grâce aux remarques de la section 1.7. Pour ceux qui sont intéressés par les systèmes hybrides, les notions topologiques sont les mieux à même d'éviter de tomber dans les classiques paradoxes de Zénon.

[4] Et l'indépendance des propriétés homotopiques des ensembles simpliciaux et cubiques modulo subdivision nous assure que les invariants géométriques de la sémantique sont stables par raffinement.

Je m'étais ensuite essayé à un certain nombre d'analyses statiques par interprétation abstraite [CC77] à partir de sémantiques par HDA.

La première étape a été publiée dans [CG93], puis dans [Gou95a] et [Gou95b] j'ai décrit une première méthode pour déterminer un sur-ensemble des ordonnancements possibles à partir de sémantiques géométriques.

Au même moment, R. Cridlig utilisait une sémantique géométrique pour faire du model-checking dans [Cri95] (pour des machines à mémoire partagée) et dans [Cri96] (sur CML, c'est-à-dire des machines qui fonctionnent par passage de messages). Il a d'ailleurs implémenté un analyseur d'un langage Pascal parallèle, voir à ce propos

`http://www.dmi.ens.fr/~cridlig`.

D'autres propositions reliées à l'analyse statique ont été faites pour utiliser l'information sémantique présente dans la sémantique géométrique. Par exemple, Y. Takayama dans [Tak95] et [Tak96] a proposé d'aider à la parallélisation automatique de programmes séquentiels (il a appliqué cela à CCS).

Je décris dans les sections suivantes mes travaux les plus récents dans ce domaine, essentiellement extraits de [FGR98a] et de [FGR98b], qui concernent la détection de points morts et de façon équivalente (par inversion du temps) des états inatteignables. C'est un algorithme capital car il est au coeur d'un analyseur d'ordonnancement [Rau00]. Cela n'étonnera pas les habitués du model-checking, dans lequel on passe souvent par un algorithme de détection d'états inatteignables pour la vérification de systèmes parallèles, voir par exemple [GW91].

2.1 Analyse de points morts et d'états inatteignables

Pour donner des exemples, nous utiliserons le langage suivant; les processus seront:

$$\begin{array}{rclll} Proc & = & \epsilon & \mid \; Pa.Proc & \mid \; Va.Proc \\ & & & \;\;\; Proc + Proc & \mid \; Y \end{array}$$

où a est n'importe quel élément de O, ensemble d'objets partagés. Chacun de ces objets peut être partagé par au plus $s(a)$ processus (où s est une fonction de O vers $\mathbb{N}$). Y est une variable de processus, permettant la définition

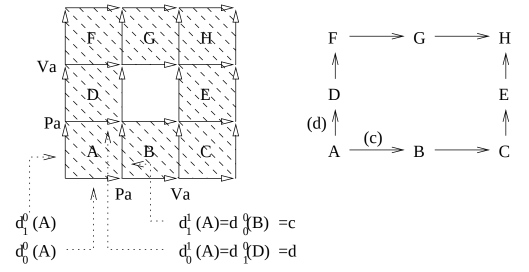


FIG. 2.3 – Première sémantique appliquée à $Pa.Va \mid Pa.Va$

de processus récursifs (par des équations récursives). Les programmes $Prog$ seront constitués d'un nombre fini de processus en parallèle.

2.1.1 Première sémantique

Un environnement d'exécution pour le langage considéré est une fonction $\rho : \mathbf{O} \to \mathbb{N}$ qui décrit pour chaque objet de $\mathbf{O}$ combien de processus peuvent encore y accéder.

On pose alors:

$$\begin{aligned}\llbracket X_1 \mid \cdots \mid X_k \rrbracket \rho = \ & (X_1 \mid \cdots \mid X_k, \rho) + \\ & \llbracket Y_1 \mid X_2 \mid \cdots \mid X_k \rrbracket \rho_1 + \cdots \\ & + \llbracket X_1 \mid \cdots \mid X_{k-1} \mid Y_k \rrbracket \rho_k\end{aligned}$$

où $(X_1 \mid \cdots \mid X_k, \rho)$ est un k-rectangle et où ρ_i, $1 \leq i \leq k$ est tel que $\rho_i(b) = \rho(b)$ pour tout $b \in \mathbf{O}$, $b \neq a_i$, et $\rho_i(a_i) = \rho(a_i) - 1$ si $Q_i = P$ ou $\rho_i(a_i) = \rho(a_i) + 1$ si $Q_i = V$. S'il existe $a \in \mathbf{O}$, $\rho(a) < 0$,

$$\begin{aligned}\llbracket X_1 \mid \cdots \mid X_k \rrbracket \rho = \ & \llbracket Y_1 \mid X_2 \mid \cdots \mid X_k \rrbracket \rho_1 + \cdots \\ & + \llbracket X_1 \mid \cdots \mid X_{k-1} \mid Y_k \rrbracket \rho_k\end{aligned}$$

avec les mêmes environnements ρ_i, $1 \leq i \leq k$.

On a ensuite les règles classiques:

(Elimination des variables de processus)

$$\begin{aligned}\llbracket X_1 \mid \cdots \mid Y. \quad Y_i \mid \cdots \mid X_k \rrbracket \rho = \\ \llbracket X_1 \mid \cdots \mid Proc_Y.Y_i \mid \cdots \mid X_k \rrbracket \rho\end{aligned}$$

(Elimination de plus)

$$\begin{aligned}\llbracket X_1 \mid \cdots \mid Y_i + \quad Z_i \mid \cdots \mid X_k \rrbracket \rho = \\ \llbracket X_1 \mid \cdots \mid Y_i \mid \cdots \mid X_k \rrbracket \rho +_i \\ \llbracket X_1 \mid \cdots \mid Z_i \mid \cdots \mid X_k \rrbracket \rho\end{aligned}$$

Un premier algorithme simple pour trouver les points morts, et les régions y conduisant inéluctablement (dites *dangereuses*) est le suivant. On commence par calculer la région interdite d'un système parallèle, $F = \bigcup_1^r R^i$ union d'hyperrectangles $R^i = [a_1^i, b_1^i] \times \cdots \times [a_n^i, b_n^i] \subseteq I^n$. Son complément dans I^n crée un graphe d'hyperrectangles (états globaux): deux hyperrectangles $R, R' \in \mathcal{R}$ sont reliés par un arc de R à R' – on écrira $R \to R'$ – si $R \leq R'$ et si R et R' ont une face en commun (c'est le graphe à droite à la figure 2.3).

Les points morts sont alors les feuilles de ce graphe (en fait celles qui ne sont pas l'état final "autorisé" $\mathbf{1} = (1, \ldots, 1)$). On peut alors propager en arrière F pour calculer la région dangereuse: on rajoute à chaque itération les hyperrectangles dont tous les bords fin sont dans la région interdite.

Un exemple de la construction de la sémantique sur le terme $Pa.Va \mid Pa.Va$ est donné à la figure 2.3. Un exemple d'application du premier algorithme de détection de points morts est donné pour le terme

```
A=Pa.Pb.Vb.Pc.Va.Pd.Vd.Vc
B=Pb.Pd.Vb.Pa.Va.Pc.Vc.Vd
PROG=A|B
```

à la figure 2.4.

C'est une méthode qui est coûteuse (un peu comme l'entrelacement, même si elle est souvent meilleure) - voir à ce propos les benchmarks du chapitre 2. Elle est assez similaire à la méthode utilisée pour le model-checking par Régis Cridlig dans son analyseur de Pascal Parallèle.

2.1.2 Deuxième sémantique

En fait, on peut directement donner dans notre cas une construction géométrique de la région interdite; c'est la deuxième sémantique $\llbracket . \rrbracket_2$ donnée par la règle:

$$[k_1, r_1] \times \cdots \times [k_n, r_n] \in \llbracket X_1 \mid \cdots \mid X_n \rrbracket_2$$

s'il existe une partition de $\{1, \cdots, n\}$ en $U \cup V$ avec $card(U) = s(a) + 1$ pour un objet a avec, $X_i(k_i) = Pa$, $X_i(r_i) = Va$ pour $i \in U$ et $k_j = 0$, $r_j = l_j$ pour $j \in V$.

On peut alors trouver un algorithme de calcul des points morts plus intelligent grâce aux remarques suivantes (voir [FGR98a]).

Pour tout ensemble d'indices non-vide $J = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$ on définit

$$R^J = R^{i_1} \cap \cdots \cap R^{i_k} = [a_1^J, b_1^J] \times \cdots \times [a_n^J, b_n^J]$$

avec $a_j^J = \max\{a_j^i | i \in J\}$ et $b_j^J = \min\{b_j^i | i \in J\}$.

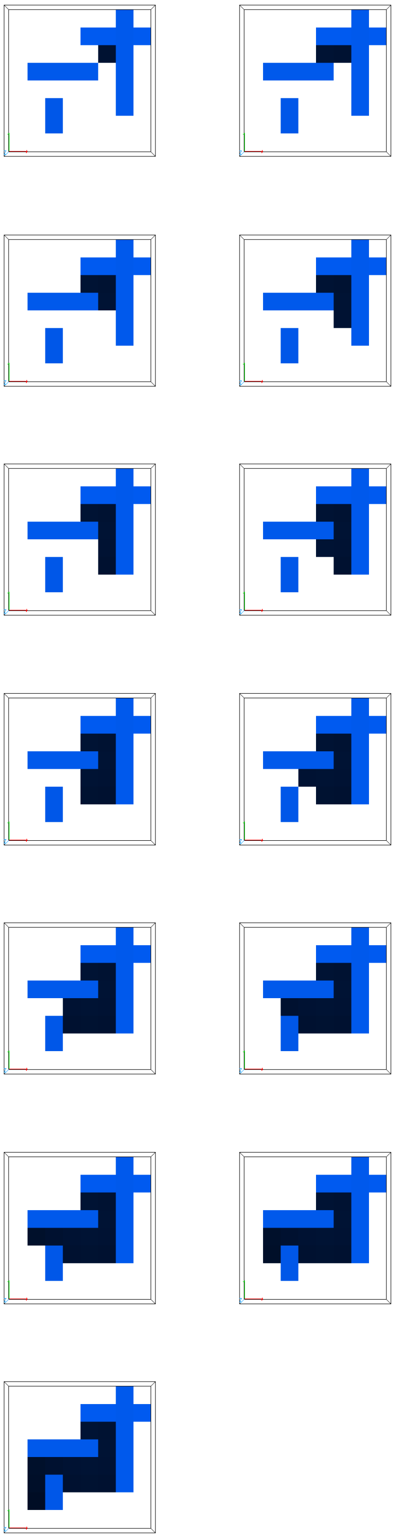


FIG. 2.4 – Les itérées successives calculant la région dangereuse.

Cet ensemble est encore un hyperrectangle sauf s'il est vide. Soit $\mathbf{a}^J = [a_1^J, \dots, a_n^J] = \min R^J$ le point *minimal* dans cet hyperrectangle.

Pour tout $1 \leq j \leq n$, soit $\widetilde{a_j^J}$ le "deuxième plus grand" des $a_j^{i_l}$, c'est-à-dire,

$\widetilde{a_j^J} = a_j^{i_s}$ avec $a_j^{i_l} \leq a_j^{i_s} < a_j^J$ pour $a_j^{i_l} \neq a_j^J$, et considérons l'hyperrectangle semi-ouvert

$$U^J =]\widetilde{a_1^J}, a_1^J] \times \cdots \times]\widetilde{a_n^J}, a_n^J]$$

"en-dessous" de R^J. Alors on a la propriété suivante:

Théorème 1 1. *Un point* $\mathbf{x} \in X = I^n \backslash F$ *est un point mort si* $\mathbf{x} \neq \mathbf{1}$ *et s'il y a un ensemble d'indices à* n *éléments* $J = \{i_1, \dots, i_n\}$*, avec* $R^J \neq \emptyset$ *et* $\mathbf{x} = \mathbf{a}^J = \min R^J$*.*

2. *Si* $\mathbf{x} = \min R^J$ *est un point-mort, alors l'hyperrectangle semi-ouvert* U^J *est dangereux, c'est-à-dire, tout dichemin* I^n *partant d'un point* $\mathbf{y} \in U^J$ *va s'arrêter à la frontière de* $\mathring{F}$*.*

On a alors un algorithme tout trouvé pour calculer la zone dangereuse. On commence par chercher l'ensemble $\mathcal{D}$ des points morts de X (c'est-à-dire les intersections non-vides d'exactement n hyperrectangles) et, pour chaque point-mort ainsi trouvé $\mathbf{a}^J \in \mathcal{D}$, on détermine l'hyperrectangle dangereux U^J.

On pose alors $F_1 = F \bigcup \cup_{\mathbf{a}^J \in \mathcal{D}} U^J$. On itère ce procédé en partant de la nouvelle région interdite F_1, ainsi de suite. On peut montrer que cet algorithme s'arrête avec un ensemble F_n tel que $I^n \setminus F_n$ ne contient aucun point mort. L'ensemble F_n caractérise la région *interdite* et la région *dangereuse*.

Par exemple, le terme plus haut, étudié à la figure 2.4 donne lieu à quatre itérées et non treize comme avec la première méthode, que l'on peut voir à la figure 2.5.

Il faut cependant faire attention aux "bords" avec cette méthode, comme on le montre à la figure 2.6. Il faut en effet considérer les bords du carré, cube etc. du graphe d'avancement, comme étant délimités par des hyperrectangles interdits, voir la figure 2.6, où j'ai représenté les quatre carrés entourant l'automate. Si un des hyperrectangles interdits touche le bord, alors il crée une intersection, qui peut à son tour créer une région dangereuse.

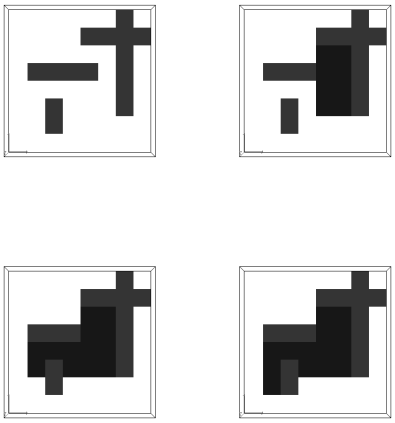


FIG. 2.5 – Itérées successives avec la deuxième méthode.

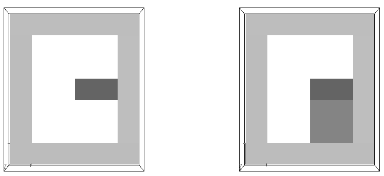


FIG. 2.6 – Une région dangereuse créee par l'interaction avec un bord.

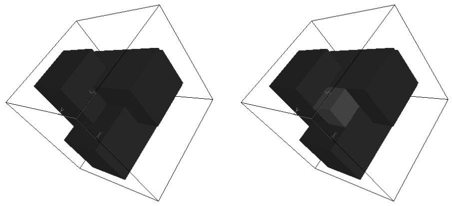


FIG. 2.7 – Les 3 philosophes.

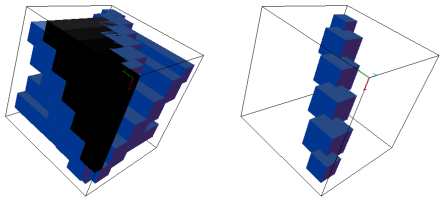


FIG. 2.8 – Un exemple où l'on modifie le niveau de partage.

L'exemple classique[5] des 3 philosophes A, B et C essayant de dîner ensemble autour d'une table ronde, en utilisant les trois fourchettes a, b et c est donné par le terme:

```
A=Pa.Pb.Va.Vb
B=Pb.Pc.Vb.Vc
C=Pc.Pa.Vc.Va
PROC=A|B|C
```

La figure 2.7 représente à gauche la sémantique (les trois cylindres à section carrée formant la région interdite) et à droite, la région dangereuse.

Un autre exemple intéressant est donné par le terme:

```
A=Pa.Pb.Va.Pc.Vb.Pd.Vc.Pe.Vd.Pf.Ve.Vf
B=Pf.Pe.Vf.Pd.Ve.Pc.Vd.Pb.Vc.Pa.Vb.Va
C=Pf.Pe.Vf.Pd.Ve.Pc.Vd.Pb.Vc.Pa.Vb.Va
PROG=A|B|C
```

Sa sémantique et région dangereuse sont données à la figure 2.8, à gauche quand on suppose que les sémaphore utilisés sont tous des 1-sémaphores, et à droite quand ce sont des 2-sémaphores: dans ce cas il n'y a plus de région dangereuse.

Un dernier exemple classique est celui de Lipsky et Papadimitriou référencé dans [Gun94]. Il correspond au terme,

[5] Tous ces exemples sont traités automatiquement par mon vieil analyseur dont l'implémentation est décrite dans le chapitre suivant.

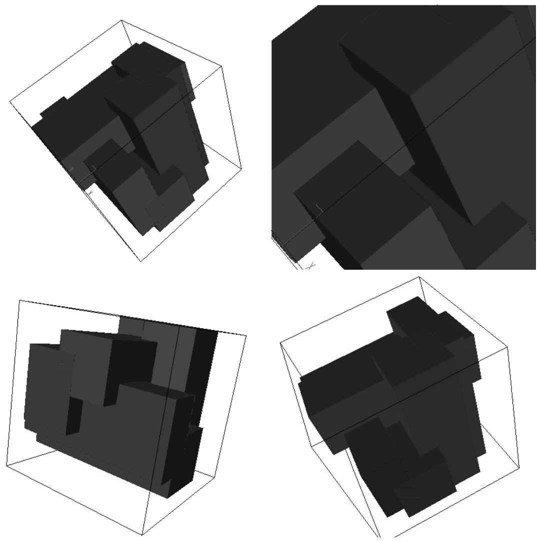


FIG. 2.9 – L'exemple de Lipsky et Papadimitriou.

```
A=Px.Py.Pz.Vx.Pw.Vz.Vy.Vw
B=Pu.Pv.Px.Vu.Pz.Vv.Vx.Vz
C=Py.Pw.Vy.Pu.Vw.Pv.Vu.Vv
PROG=A|B|C
```

dont on a les différentes vues à la figure 2.9. En fait, la région interdite est trouée, et peut laisser passer une classe de dihomotopie de dichemin, sans point mort.

2.1.3 Boucles

Nous avons vu ce qui se passe dans le cas des processus PV qui forment des graphes d'avancement. Qu'en est-il des boucles (et donc des po-espaces locaux)?

Prenons par exemple le terme:

```
A=Pa.(Pb.Vb.Pc.Vc)*.Va
B=Pc.Pb.Vc.Pa.Va.Vb
PROG=A|B
```

Sa sémantique, telle que définie dans [Faj00] ou dans [FS00] revient à représenter géométriquement le programme où on déroule une fois toutes les boucles (y compris internes), c'est-à-dire ici,

```
A=Pa.Pb.Vb.Pc.Vc.Va
B=Pc.Pb.Vc.Pa.Va.Vb
PROG=A|B
```

puis à replier sur chaque coordonnée où on a une boucle. Ce processus est décrit à la figure 2.10, où on passe du dessin de gauche à celui de droite

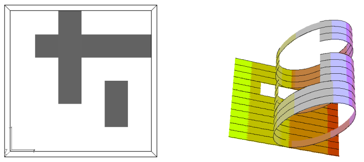


FIG. 2.10 – Repliage de la sémantique

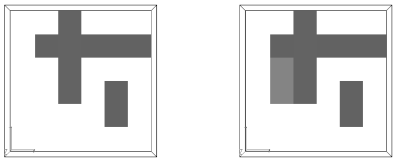


FIG. 2.11 – Un dépliage et la région dangereuse correspondante.

en repliant les points d'abscisse juste après `Pa` sur ceux juste après `Pa.Pb.Vb.Pc.Vc`.

On peut parfaitement généraliser les concepts de régions dangereuses et inatteignables. Ici, on dit qu'un état est dans la région dangereuse si il n'existe pas de chemin fini permettant d'atteindre l'état final. C'est le point de vue de [Faj00] que je suis ici. On peut aussi modifier la définition pour autoriser les chemins infinis[6], et c'est le point de vue de [FS00] qui aboutit aux mêmes types de résultats.

L'idée naturelle est d'essayer de trouver un critère simple permettant de dérouler de façon minimale les boucles, pour déterminer les régions dangereuses. A priori ce n'est pas évident, comme on le montre maintenant.

On s'aperçoit qu'un point juste en dessous de la région dangereuse pour un dépliage, pour l'exemple de la figure 2.10, n'est pas dans la région dangereuse, car il y a un chemin vers l'état final avec un ou zéro déroulement de la boucle (voir figure 2.11). Mais avec deux déroulements (figure 2.12), il semble appartenir à la région dangereuse.

Voici un exemple plus compliqué qui montre qu'inversement, on peut avoir l'impression en déroulant les boucles qu'un état est dans une région dangereuse, alors que ce n'est en fait pas le cas. Considérons la sémantique de

[6] Quand on s'intéresse aux systèmes d'exploitation, on ne veut considérer comme dangereux que les points qui n'ont qu'un futur fini qui n'atteint pas un état final.

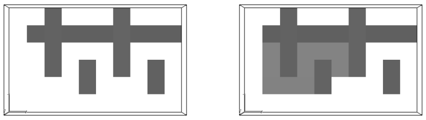


FIG. 2.12 – Région dangereuse après deux déroulements.

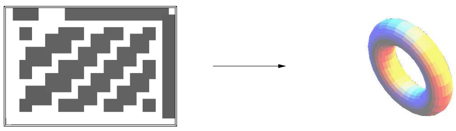


FIG. 2.13 – Repliage de la sémantique sur un tore.

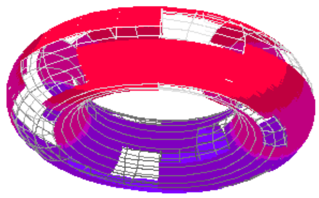


FIG. 2.14 – Une vue de la région interdite sur le tore.

```
A=Pd.Pa.(Pb.Va.Vd.Pc.Vb.Pa.Pd.Vc.Pb.Va.
        Pc.Vb.Pa.Vc.Pb.Va.Pc.Vb.Pa.Vc)*
                           .Va.Pe.Vd.Ve
B=Pe.Pa.(Pb.Va.Pc.Vb.Pa.Vc.Pb.Va.Pc.Vb.
         Pa.Vc)*.Va.Pd.Ve.Vd
PROG=A|B
```

Elle est obtenue, comme il y a une boucle sur chaque coordonnée, par un repliage d'une portion de la sémantique de,

```
A=Pd.Pa.Pb.Va.Vd.Pc.Vb.Pa.Pd.Vc.Pb.Va.
        Pc.Vb.Pa.Vc.Pb.Va.Pc.Vb.Pa.Vc
                         .Va.Pe.Vd.Ve
B=Pe.Pa.Pb.Va.Pc.Vb.Pa.Vc.Pb.Va.Pc.Vb.
        Pa.Vc.Va.Pd.Ve.Vd
PROG=A|B
```

sur un tore, comme indiqué à la figure 2.13.

Si on regarde la partie repliée sur le tore, et en représentant les trous en plein, on a la figure 2.14.

Si on déroule deux fois chacune des deux boucles, on obtient la figure 2.15 dans laquelle j'ai représenté côte à côte la vue dépliée et un chemin correspondant sur le tore.

En déroulant deux fois la boucle pour `A` et trois fois celle pour `B`, on obtient la figure 2.16, où on a également représenté la région dangereuse. Ce que l'on peut voir sur cette figure, c'est que l'on peut trouver des dichemins qui ne vont pas s'arrêter sur un point mort contrairement à ce que l'on avait pour les déroulements précédents.

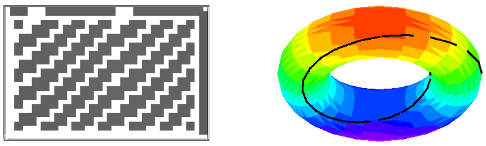


FIG. 2.15 – En déroulant deux et deux fois.

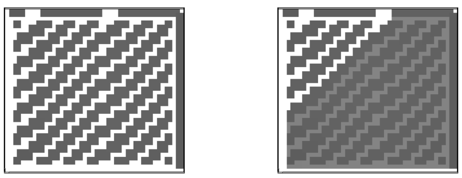


FIG. 2.16 – En déroulant deux et trois fois.

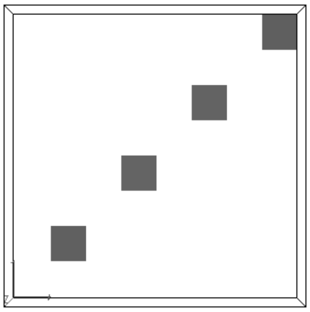


FIG. 2.17 – Cas le pire pour l'algorithme d'ordonnancement.

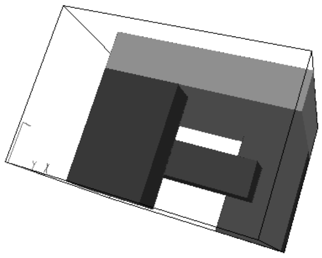


FIG. 2.18 – Un point critique pour la "chambre à trois trous".

Le résultat important, démontré par L. Fajstrup dans [Faj00] est que les points morts sont calculables en un déroulement. L. Fajstrup montre également que l'on peut calculer la région dangereuse en examinant certains déroulements seulement, en nombre fini, coordonnée par coordonnée. J'essaie maintenant d'améliorer mon analyseur statique pour tirer partie de ces résultats.

2.2 Analyse d'ordonnancements

Je ne reviendrai pas ici sur les travaux qui remontent à ma thèse, portant sur des méthodes homologiques pour approximer les ordonnancements, et publiés dans [Gou95b].

Le fait important, démontré par M. Raussen dans [Rau00] est que ce qui compte pour trouver les autres régions diconnexes que celles contenant la région dangereuse et la région inatteignable, ce sont les intersections d'exactement $n-1$ hyperrectangles de dimension n (appelés $(n-1)$-points critiques). En dimension deux, cela veut dire quelque chose d'assez simple, c'est que ces régions sont créées par les points minimaux et maximaux des hyperrectangles, comme on peut le voir avec l'exemple du "drapeau Suisse" de la figure 1.9.

M. Raussen donne, dans [Rau00], une méthode de classification complète des dichemins en dimension 2, qui n'est qu'une sur-approximation en dimension supérieure ou égale à trois.

Même si cela ne donne qu'une sur-approximation en dimension supérieure à trois, cette méthode permet de déterminer correctement les régions diconnexes pour l'exemple de la "chambre à trois trous". La figure 2.18 montre une intersection de deux hyperrectangles de dimension trois (ne pas oublier que les bords comptent!) qui crée l'obstruction à la dihomotopie entre les deux chemins du chapitre précédent.

L'algorithme qui en découle est exponentiel (au pire) en le nombre de composantes connexes de la région interdite (complétée par l'ajout des régions dangereuses et inatteignables), cette borne étant atteinte comme en atteste l'exemple de la figure 2.17. Là encore, je vais essayer de trouver un peu de temps pour compléter mon analyseur avec les algorithmes correspondants.

2.3 Combinaison d'analyses

2.3.1 Temps locaux abstraits

On s'aperçoit aisément que les algorithmes de détection de points morts, ou de classification des dichemins modulo dihomotopie ne dépendent que de la structure d'ordre partiel entre les états. On a besoin essentiellement de prendre des intersections et des unions de régions interdites.

On peut donc penser à approximer, par des treillis numériques, les régions par des rectangles "isothétiques" plus grands et plus facilement manipulables. Par exemple, on peut approximer un ensemble quelconque d'hyperrectangles par des hyperrectangles modulo un entier: $(\{[1,4],[9,12],[18,20]\}$ par $[1,4]+8k$ $(k \in \mathbb{Z}))$.

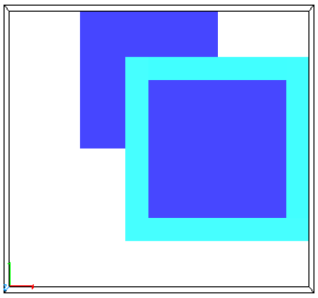


FIG. 2.19 – Le carré clair est une abstraction d'une partie de la région interdite.

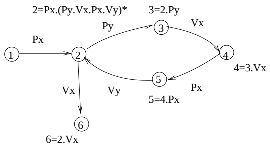


FIG. 2.20 – Le processus Y.

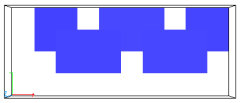


FIG. 2.21 – Les deux familles R_2 et R_3.

J'ai commencé à développer une abstraction d'ensembles d'hyperrectangles par des expressions régulières dans [FGR98b]. C'est une abstraction intéressante qui, par exemple, permet de déterminer de façon exacte la région dangereuse de la version récursive des trois philosophes qui dînent:

```
A=Pa.Va.Pb.Vb.A
B=Pb.Pc.Vb.Vc.B
C=Pc.Pa.Vc.Va.C
PROG=A|B|C
```

Par contre, avec l'exemple suivant,

```
X=Py.Vx.Px.Vy.X
Y=Px.X.Vx
Z=Px.Vx
PROG=Y|Z
```

les choses se gâtent. L'idée de l'abstraction par des expressions régulières est de remplacer les coordonnées temporelles entières (temps local sur chaque axe dans les graphes d'avancement) par des expressions régulières sur le langage `Pa`, `Va` (pour tout `a`). L'ordre (partiel) est l'ordre préfixe. Les "coordonnées" des états se trouvent sans peine en regardant les automates séquentiels représentant les processus (voir par exemple la figure 2.20).

Si je note par $[X, Y]$ "l'intervalle" d'expressions régulières ayant pour préfixe X et étant préfixes de Y, les mêmes méthodes que précédemment permettent de trouver deux hyperrectangles abstraits qui s'intersectent (voir la figure 2.21):

$$R^2 = Px.(Py.Vx.Px.Vy)^*.Py.Vx.Px.$$
$$[1, Vy.Py.Vx] \times Px.[1, Vx]$$
$$R^3 = Px.(Py.Vx.Px.Vy)^*.Py.Vx.Px.$$
$$[1, Vy.Vx] \times Px.[1, Vx]$$
$$R^2 \cap R^3 = Px.(Py.Vx.Px.Vy)^*.Py.Vx.Px.$$
$$[1, Vy] \times Px.[1, Vx]$$

et qui créent une région dangereuse (dûe au partage de `x`), qui n'existe pas dans la réalité. C'est en ce sens que l'on obtient qu'une sur-approximation des régions dangereuses et inatteignables.

2.3.2 Réduction de l'espace d'états

Pour utiliser ces algorithmes afin d'analyser de "vrais" programmes parallèles (par exemple, et c'est naturel, des **threads** JAVA) on peut faire comme on a dit plus haut: on commence par abstraire la coordination des processus: les temps locaux sont alors des éléments d'un treillis abstrait (environnement, trace séquentielle etc.). On calcule ensuite par l'algorithme de M. Raussen les traces essentielles. Ensuite on calcule la sémantique collectrice (abstraite) uniquement sur ces traces. C'est donc un procédé de réduction de l'espace d'états. En fait, on peut tenter de faire encore mieux; on peut maintenir en même temps l'abstraction de la coordination avec la sémantique collectrice abstraite (sorte de partitionnement dynamique). Je n'ai pas pour le moment eu le temps de tester ces idées, mais j'espère pouvoir le faire dans un avenir proche.

2.4 Systèmes distribués tolérants aux pannes

Commençons par prendre un exemple simple, traité en détail dans [Gou97], et dont les premières idées ont été décrites dans [Gou96b] puis

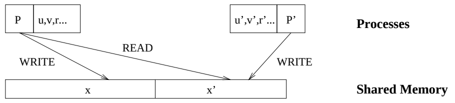


FIG. 2.22 – Une machine distribuée simple à mémoire partagée.

ont été exposées de façon plus générale dans [Gou96a].

On suppose que l'on a une machine distribuée à deux processeurs P et P' (que l'on a représentée à la figure 2.22) qui communiquent par lecture et écriture dans une mémoire partagée.

Par exemple, le processeur P (respectivement P') peut écrire, par l'action `scan`, dans une variable locale u (respectivement u') la valeur contenue dans une variable x (respectivement x'). P peut aussi copier x *et* x' dans sa mémoire locale, par une action *update*, respectivement dans u et v (et pour P' dans u' et v'). Chaque processeur peut exécuter également une instruction `case` et toutes les opérations arithmétiques et boucles qui font qu'ils peuvent calculer localement n'importe quelle fonction partielle récursive. Cette machine est en ce qui nous concerne, complètement équivalente à une machine à mémoire partagée avec lecture écriture atomiques.

Une telle machine a la propriété que tout calcul est fait "sans-attente", c'est à dire qu'il n'y a aucune synchronisation entre les processeurs (que l'on identifiera à leurs processus séquentiels respectifs). Cela implique que cette machine est en quelque sorte tolérante aux pannes au sens où si l'un des deux processeurs plante et s'arrête donc de calculer, l'autre peut continuer de calculer, avec les données partielles qu'il possède.

Une application importante est, encore une fois, le cas d'une base de données distribuée. Supposons que nos transactions, qui se déroulent dans des guichets très éloignés les uns des autres, veuillent modifier la même variable, mais en lui donnant des valeurs différentes. C'est par exemple le cas de deux clients voulant réserver la même place sur le même avion, la variable étant le nom de la place, et la valeur étant le nom du passager. La question est: peut-on trouver un protocole qui, étant donné une architecture de la base de données distribuées (gérant les places d'avions), assure que seul un des deux clients aura le billet souhaité, alors que l'autre saura de façon certaine qu'il n'a pas la place? C'est un cas particulier du problème fondamental appelé *consensus*: on veut que les deux processus (ou transactions) se mettent d'accord sur une valeur commune (le nom de la personne qui prend effectivement la place). Ce n'est un vrai problème que si on suppose qu'il peut y avoir une panne réseau, ou que l'une des deux transactions peut planter à tout moment, sans en avertir l'autre.

C'est évidemment un problème de calculabilité. Dans le cas de deux processeurs, cela a été complètement résolu dans [FLP85]: il n'est pas possible de résoudre le problème du consensus sans-attente sur notre simple petite machine.

Un cas plus général, quand on considère n processeurs avec $n > 2$, est la t-résilience (avec $1 \leq t \leq n-1$); on suppose maintenant que jusqu'à t parmi les n processeurs peuvent planter à tout moment. Plus généralement, on peut aussi considérer des architectures de machines distribuées plus réelles, avec des objets partagés plus complexes que des simples variables scalaires, par exemple des files ou des piles, ou communiquant par passage de messages, bloquant ou non bloquant. Le problème du consensus t-résilient est donc plus subtil: les processus s'exécutant sur les n processeurs doivent être aussi peu couplés que possible pour que même si t processus flanchent, les autres puissent terminer avec la même valeur, qui doit être une des valeurs de départ de l'un des n processeurs[7].

Si le résultat avec deux processus est assez ancien, et utilise des techniques de théorie des graphes, il a fallu attendre jusqu'à assez récemment pour avoir une caractérisation de ce qui peut être calculé sur notre machine simple avec n processeurs, n quelconque. Cela a été réalisé en utilisant des techniques de topologie algébrique combinatoire (des ensembles simpliciaux). C'est en fait encore une fois en partie un problème d'homotopie orienté comme je vais expliquer un peu plus bas.

Le principal argument ("similarity chain") de [FLP85] peut se comprendre comme un résultat de connexité. Si on représente les états locaux de chaque processeur P_i $(i = 1, 2)$, à un moment donné de l'exécution du programme, par un point (P_i, x_i) (x_i étant la valeur à ce moment donné de la variable privée de P_i), et les états globaux du système par un segment (non-orienté) liant les deux états locaux qui les composent, alors il est facile de voir que la sémantique des `scan`/`update` impose que les états

[7] Une bonne référence pour avoir une idée des nombreux protocoles existant dans le domaine est le livre [Lyn96].

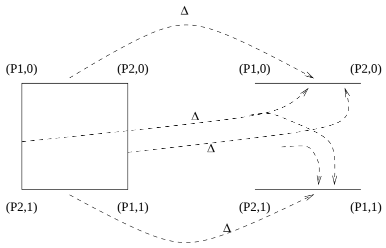


FIG. 2.23 – La fonction de décision pour le consensus binaire, des états globaux initiaux vers les finaux.

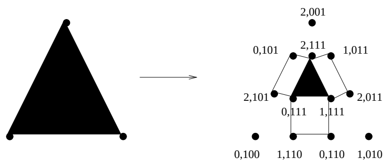


FIG. 2.24 – Le complexe de protocole dans le cas de trois processus synchrones effectuant une étape.

globaux atteignables à partir d'un état global initial, forment un graphe connexe. Il n'y a pas moyen qu'un programme sur une telle architecture casse la connectivité. Cela implique que le consensus ne peut pas être implémenté sur une telle machine, car il faudrait pour cela atteindre, depuis l'état global $((P_1,0),(P_2,1))$ deux états globaux disconnectés $((P_1,0),(P_2,0))$ et $((P_1,1),(P_2,1))$ comme on l'a dessiné à la figure 2.23.

D'autres travaux ont généralisé la preuve de [FLP85] à plus de deux processeurs. On trouve dans [BMZ88] une caractérisation de la classe de problèmes que l'on peut résoudre dans un système qui fonctionne par passage de messages non-bloquants dans le cas d'une panne au maximum (1-résilience). C'est la dernière généralisation que je connaisse qui utilise des techniques de théorie des graphes.

La conjecture [Cha90] que le problème de l'accord sur un ensemble de k valeurs ("k-set agreement"[8]) ne peut pas être implémenté dans certains systèmes asynchrones a finalement été prouvée dans trois papiers de façon indépendante, [BG93], [SZ93] et [HS93].

L'idée de base de [HS93] est de généraliser les représentations du type de celles de la figure 2.23 en utilisant des ensembles simpliciaux au lieu des graphes (qui ne sont que des cas particuliers).

Les ensembles simpliciaux sont formés de points, segments, et également de triangles, tétraèdres, simplexes en toute dimension, collés les uns aux autres le long de leurs faces. La formalisation est tout à fait similaire à celle des ensembles cubiques que l'on a vu auparavant (on pourra se reporter à [May67] et [GZ67]). Dans notre cas, un simplexe de dimension n représente un état global, à un moment de l'exécution, de n processus. Les points sont toujours des paires nom du processus et état local. Encore une fois, étant donné un état global initial, la sémantique des opérations de la machine distribuée étudiée est définie par les états globaux atteignables, au cours du temps. Par exemple, la figure 2.24 représente l'ensemble simplicial, appelé par M. Herlihy et ses collaborateurs, complexe de protocole ("protocol complex") après zéro (à gauche) et une (à droite) étape de communication, sur une machine distribuée synchrone (avec une horloge globale) communiquant par messages, qui donc envoie les états locaux de chaque processus (en tout 3) à chaque autre, à chaque étape. L'ensemble simplicial à droite est constitué par exemple de points isolés correspondants aux états locaux d'un des trois processus, les deux autres étant morts, le triangle central correspondant à l'état global où tous les processus sont encore vivants, et ont bien eu connaissance des messages envoyés par leurs alter ego.

On peut montrer que s'il existe une fonction simpliciale de ce complexe de protocole vers un certain ensemble d'états globaux (organisés non plus en graphe mais en ensemble simplicial), respectant la spécification du problème que l'on veut résoudre, alors il existe un protocole sans-attente satisfaisant à cette spécification. Ce type de méthode nous permet également de discuter non plus seulement de la calculabilité sur certaines architectures parallèles, mais aussi de la complexité de calcul, c'est-à-dire sur une architecture à passage de messages, du nombre minimal de messages à échanger, où sur une architecture à mémoire partagée, du nombre de lectures écritures nécessaires.

On peut reformuler ces arguments, au moins dans un cas simple, comme suit. Prenons l'exemple d'un programme *Prog* composé des deux

[8] C'est une forme de consensus faible dans laquelle tout ce qui est requis est que les processus qui ne plantent pas arrivent finalement à se mettre d'accord en un temps fini sur un sous-ensemble de au plus k valeurs parmi toutes les valeurs d'entrée des n processus de départ.

processus suivants s'exécutant en parallèle,

$$\begin{array}{rcl} P & = & update; \\ & & scan; \\ & & case\ (u,v)\ of \\ & & \ (x,y'):\ u=x'; update; \\ & & \ default:\ update \end{array}$$

$$\begin{array}{rcl} P' & = & update; \\ & & scan; \\ & & case\ (u,v)\ of \\ & & \ (x,y'):\ v=y; update; \\ & & \ default:\ update \end{array}$$

On a principalement les trois ordonnancement possibles suivants, car les seules actions pouvant interagir sont `scan` et `update`.

(i) Supposons que l'action `scan` de P se termine avant que l'action `update` de P' ne démarre: P ne connaît pas y donc P choisit d'écrire x. $Prog$ termine dans l'état global $((P,x),(P',y))$.

(ii) Cas symétrique: $Prog$ termine dans l'état $((P,x'),(P',y'))$.

(iii) L'action `scan` de P vient après l'action `update` de P' et l'action `scan` de P' vient après l'action `update` P. $Prog$ termine dans l'état $((P,x'),(P',y))$.

On voit bien que chacun des trois ordonnancements possibles correspond à un ordonnancement des opérations `scan` et `update` uniquement.

Si on représente la commutation de deux transitions par une 2-transition, et la non-commutation, par l'absence de 2-transition, comme on l'a déjà fait pour les processus PV, on retrouve nos trois ordonnancements comme étant les trois dichemins modulo dihomotopie de la partie gauche de la figure 2.26. Cela revient à identifier le flot de programme de notre langage asynchrone avec celui d'un programme avec sémaphores, pour lequel la paire (`scan`, `update`) est remplacée par une interaction entre des actions P et V. C'est une bonne abstraction, quand on ne s'intéresse qu'aux effets de l'historique des communications (voir figure 2.25).

On a maintenant deux types de configurations[9] de trous dans un carré, quand on examine les classes de dihomotopie des dichemins, comme on le voit à la figure 2.26.

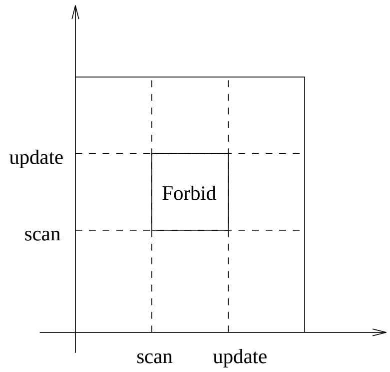


FIG. 2.25 – L'analogue en termes de PV de `scan` et `update`.

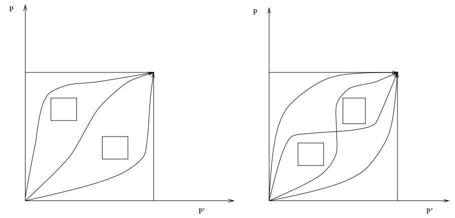


FIG. 2.26 – Les deux configurations possibles de trous.

Dans la première configuration, il y a trois ordonnancements possibles (c'est-à-dire classes de dichemins), alors que dans la deuxième configuration, il y a quatre ordonnancements possibles. Si on veut connaîre l'ensemble des ordonnancements possibles dans des cas plus complexes, comme celui de la figure 2.27, on obtient une structure de type arbre (aussi représentée à la figure 2.27).

En fait, on peut facilement décrire un surensemble des ordonnancements possibles, en toute situation.

Formellement, deux trous sont "comparables" s'il existe un dichemin de l'état final du premier trous vers l'état initial du deuxième trou. Cette "comparaison" est un ordre partiel, et n'importe quelle linéarisation de cet ordre partiel (obtenue en déformant la configuration relative des trous)

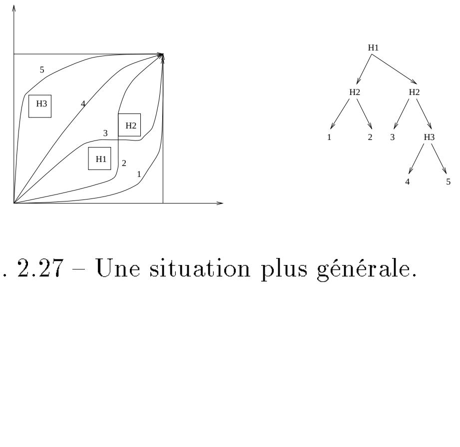


FIG. 2.27 – Une situation plus générale.

[9] Il ne peut pas y avoir de recouvrement entre les trous produits par la sémantique ici.

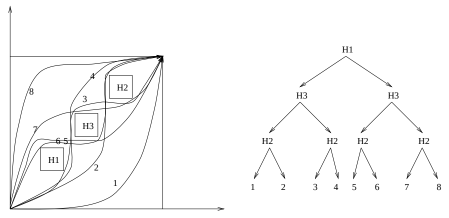


FIG. 2.28 – Le “type de dihomotopie” d'une chaîne de trous.

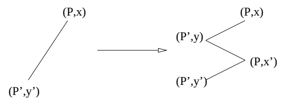


FIG. 2.29 – Subdivision d'un segment en trois segments.

nous donne un sur-ensemble des ordonnancements possibles. Il suffit de remarquer maintenant qu'une chaîne de trous a exactement comme “type de dihomotopie”, celui d'un arbre binaire (voir figure 2.28).

A chaque feuille de cet arbre, on peut associer un segment comme suit:

- Un point d'un de ces segments est n'importe quel état local, c'est-à-dire (P, x) ou (P', x'),
- Un segment entre (P, x) et (P', x') représente l'état global des deux processus.

Une feuille de l'arbre correspond à un des ordonnancements possibles, et ne peut aboutir qu'à un unique état global, c'est-à-dire à un segment.

Par exemple, le programme $Prog$ défini plus haut prend un segment (l'état global de départ) et l'amène sur trois segments, comme on le représente à la figure 2.29. C'est un cas particulier d'un phénomène plus général. Dans la sémantique du langage `scan`/`update`, si on se déplace d'une feuille de l'arbre binaire au suivant (de gauche à droite par exemple), on doit toujours partager un point des deux segments (un point P ou P'), donc le graphe que l'on peut atteindre à partir d'un segment doit toujours être connexe. Cela implique le résultat plus formel suivant: soit $\{e_1, \ldots, e_k\}$ l'ensemble des états globaux terminaux d'un programme à partir de l'état global initial $e = ((P, u), (P', v'))$. Alors le graphe engendré par l'identification des états locaux, comme plus haut, à partir des ses états globaux, vus comme des segments, est un graphe connexe. De fait, il y a une réciproque à ce résultat, et il est possible, comme on l'explique en détail dans [Gou96a], étant donné une spécification d'une fonction distribuée à calculer respectant l'hypothèse plus haut, de trouver un protocole écrit dans notre langage, l'implémentant (comme cela a été expliqué également et pour la première fois dans [HS94]).

Cela démontre en particulier ce que l'on avait affirmé au début de cette section, à savoir que le consensus sans-attente ne peut se résoudre sur notre machine simple. En démarrant par l'état global $((P, 0), (P', 1))$ les résultats que l'on veut pouvoir obtenir sont $\{((P, 0), (P', 0)), ((P, 1), (P', 1))\}$ qui ne forment pas un graphe connexe.

Il existe bien évidemment des résultats plus intéressants que celui-là dans la littérature. Si j'ai une bonne intuition du lien entre les travaux de M. Herlihy et ses collaborateurs dans le cas général, il reste des trous dans la formalisation, car de part et d'autre, la théorie n'est pas tout à fait assez mûre. Le projet GOODS s'il voit finalement le jour, répondra j'espère à ces manques. Dans [HS93] est prouvé également le fait que le problème de l'accord sur k valeurs (“k-set agreement”) sur une machine à mémoire partagée avec lecture/écriture atomique requiert au moins $\lfloor f/k \rfloor + 1$ étapes (où chaque processus commence par lire puis écrire), où f est le nombre de processus qui peuvent planter (c'est donc la f-résilience dont on parle ici).

La tâche de renommage (les processus doivent s'accorder entre eux pour se donner des noms, dans un plus petit ensemble de noms que l'ensemble de nom d'origine), d'abord proposée dans [ABND$^+$90] a été finalement résolue dans [HS93]. Il y a un protocole sans-attente pour le problème du renommage dans certains systèmes asynchrones si l'espace de choix de noms est suffisament grand. C'était déjà connu pour $2n + 1$ noms ou plus quand on a $n + 1$ processus asynchrones, et on savait également qu'il n'y en avait pas si on ne disposait pas d'au moins $n + 2$ noms. M. Herlihy et N. Shavit (voir aussi [HR95] et [HR00]) ont raffiné ces résultats en montrant qu'il n'y avait pas non plus de solution avec un espace de nom de cardinalité inférieure strictement à $2n + 1$.

Plus généralement, les types des données partagées importent pour les résultats de calculabilité dans une architecture à mémoire partagée. On sait depuis assez longtemps (L. Lamport) que le consensus pour deux processus peut se résoudre avec lecture, écriture atomique et une file (FIFO), ou avec une opération test&set atomique.

Il est donc naturel de définir ce que M. Herlihy a appelé le nombre de consensus ("consensus number") d'un type de données partagées, comme étant le nombre maximal de processus asynchrones (ayant de toutes façons la lecture, écriture atomique au moins) pour lequel il existe un protocole sans-attente pour résoudre le consensus. On sait par exemple que,

- les registres atomiques (lecture, écriture atomiques) ont le nombre de consensus 1,
- les variables permettant d'effectuer test&set et fetch&add, les files et piles ont pour nombre de consensus 2,
- L'affectation simultanée atomique à n registres a pour nombre $2n-2$,
- les registres load-locked, store-conditional et compare-and-swap ont pour nombre de consensus ∞.

Cela motive l'introduction du problème encore plus général suivant. On dit qu'un type de données partagées est un objet (m, j)-consensuel s'il permet à n'importe quel ensemble de m processus de résoudre le problème d'accord sur j valeurs. M. Herlihy et S. Rajsbaum dans [HR94] (voir aussi [BG93]) ont prouvé qu'il est impossible d'implémenter un objet $(n+1, k)$-consensuel en utilisant des objets tous (m, j)-consensuels si $n/k > m/j$.

On peut trouver beaucoup de résultats dans cette optique dans [Jay93], [Jay97] et [Sch97]. En particulier [Jay97] identifie un problème qui peut apparaître assez étrange dans un premier temps, qui est que cette "hiérarchie" des objets n'en est en fait pas une. Il est en effet possible d'implémenter un objet de nombre de consensus k en utilisant plusieurs autres types de données différents de nombre de consensus tous strictement inférieurs à k

De fait, cela est dû à des interactions subtiles entre les différents types de données. Pour moi, c'est exactement le même problème d'interaction que l'exemple de la figure 1.7. Mon espoir est que la classification plus systématique des ordonnancements d'actions dans des architectures distribuées complexes doit permettre de classifier plus finement la calculabilité et complexité des protocoles distribués tolérants aux pannes, en s'abstrayant de la contrainte nécessaire jusqu'à présent, de devoir utiliser des modèles des systèmes distribués trop simples (qui permettent sans étude fine, de déterminer l'ensemble des états atteignables à tout instant, c'est-à-dire permettant d'étudier le problème en partant du complexe de protocole).

Chapitre 3

Projets en cours

3.1 Analyseurs statiques

3.1.1 Analyseur de points morts

J'ai écrit, alors que j'étais au CNRS, une librairie générale en C de manipulation de HDA. Les HDA y sont représentés par des matrices d'incidence; un HDA $\mathcal{R}$ est défini par des 4-uplets (pour tout n)

$$(S^0_{n-1}, S^1_{n-1}, R^0_n, R^1_n)$$

R^i_n est la matrice (creuse) dont les lignes $R^i_n(x)$ sont indicées par les n-rectangles x et contiennent les bords début (si $i = 0$) et fin (si $i = 1$) de x. S^i_{n-1} est la matrice de "co-incidence" dont les lignes $S^i_{n-1}(y)$ sont indicées par les faces y et sont constituées par les n-rectangles dont les bords début (pour $i = 0$) ou fin (pour $i = 1$) contiennent y. Algorithmiquement, les lignes et les colonnes des matrices creuses sont des listes ordonnées doublement chaînées.

Au départ, cette librairie avait été développée pour faire des calculs homologiques d'ordonnancement, comme ceux exposés dans [Gou95b] et [Gou95a] (en utilisant des algorithmes de calculs de relations de dépendances linéaires rapides, comme ceux utilisés en cryptographie). De fait, je ne l'ai utilisé pour l'instant que pour tester l'idée de la section 2.1.1 pour le calcul des points morts, qui est le parcours exhaustif que je décris plus bas.

On suppose pour le calcul des points morts que l'on maintient une liste F de rectangles interdits. L'analyseur implémente la sémantique naïve complète des termes PV de la section 2.1.1.

A cause de la présence de termes récursifs, il faut vérifier à la création des n-rectangles et de ses faces, qu'ils n'ont pas déjà été créés. Un algorithme rapide de recherche d'état, basé sur une fonction de hachage particulière, permet d'atteindre des performances honorables. Il faut également remarquer que l'on ne crée que les faces des n-rectangles qui sont nécessaires à la sémantique (c'est-à-dire qui sépare au moins deux n-rectangles distincts).

On calcule ensuite très naïvement (un peu comme on ferait en model-checking, sans algorithme de réduction de l'espace d'états) le sous-ensemble D de l'ensemble d'états ascendants d'un ensemble S donné d'états, tels que tous leurs descendants atteignent S (et seulement S) au bout d'un temps fini.

On suppose que S est organisé en une pile FIFO q. On peut effectuer les opérations *empty*?, *enq* (pour mettre un nouvel élément dans q) et *deq* (pour enlever un élément de q).

On associe également à chaque n-rectangle x un entier m_x, initialisé lors de la construction du HDA, tel que.

- Pour tout n-rectangle x de S, l'entier m_x est initialisé à 0,
- Pour tout autre n-rectangle, m_x est initialisé à son nombre de successeurs distincts.

Alors, le multi-ensemble P_x des n-rectangles parents d'un n-rectangle x donné est l'union ("merge") des listes $S^1_{n-1}(y)$ pour tous les $y \in R^0_n(x)$. L'algorithme pour calculer D est comme suit. D est vide au début, puis,

(1) Si *empty*? alors on s'arrête,
(2) On décrémente m_z de un pour tous les $z \in P_{deq}$,
(3) Si un de ces m_z atteint 0, alors on rajoute z à D et on fait $enq(z)$,
(4) On boucle en (1).

Pour le deuxième algorithme de détection de points morts, celui de la section 2.1.2, j'ai également écrit une librairie C pour manipuler des unions finies de n-rectangles, alors que j'étais invité pour un mois à l'Université d'Aalborg. J'en ai également profité pour interfacer l'analyseur avec `geomview` qui est un outil de visualisation de formes géométriques assez performant. A l'époque, j'avais décidé de prototyper rapidement, et de représenter les n-rectangles par des listes de n intervalles. Les régions, c'est-à-dire les unions finies de n-rectangles, sont représentées par des listes de n-rectangles. Pour les besoins de l'algorithme, j'ai aussi du définir les n-rectangles étiquetés (par une région), et les régions étiquetées (listes de n-rectangles étiquetés). Le prototype de 1997 peut être essayé, à travers une petite interface CGI[1] que l'on trouvera à l'adresse,

`www.dmi.ens.fr/~goubault/analyse.html`

L'algorithme de détection de points morts maintient à tout instant une suite `pile` de régions étiquetées, telle que,

- `pile[0]` contient l'ensemble des hyper-rectangles formant la région interdite,
- `pile[1]` contient l'intersection d'exactement deux régions distinctes de `pile[0]`,
- etc.,
- `pile[n-1]` contient l'intersection d'exactement n régions distinctes de `pile[0]`.

Afin de calculer l'effet de l'ajout d'un nouvel hyperrectangle S, le programme d'analyse appelle la procédure `complete(S,∅)`[2].

```
complete(S,l)
  if S is included into a X in pile[0]
    return
  for i=n-2 to 0 by -1 do
    pile[i+1]=intersection(pile[i]\l,S)
  pile[0]=union(pile[0],S)
  for all X in pile[n-1] do
    pile[n-1]=pile[n-1]\X
    derive(X)
```

`derive(X)` s'occupe d'inférer une nouvelle portion de la région dangereuse à partir d'une intersection `X` of `n` hyperrectangles interdits `X1,...,Xn`:

```
derive(X)
  for all i do
    yi=max({Xj(i) / j=1,...,n}\{X(i)})
  Y=[y1,X(1)]x...x[yn,X(n)]
  if Y is not included in one of the Xj
    complete(Y,(X1,...,Xn))
```

A l'époque, le code écrit en C avait fait l'objet de quelques benchmarks. Compilé avec `gcc -O2` sur une Ultra Sparc 170E avec 496 Mbytes de mémoire vive, 924 Mbytes de cache, j'ai obtenu les résultats suivants, en comparaison avec l'analyse naïve (publiés dans [FGR98a]).

Les performances de la méthode naïve sur de simples petits exemples sont résumées dans le tableau suivant (P est le nom du programme, dim sa dimension, c'est-à-dire le nombre de processus en parallèle, $\#fbd$ est le nombre de n-rectangles interdits effectivement représentés, $t\ s$ est le temps pour représenter le HDA, en secondes, $t\ uns$ est le temps mis pour en déduire la région dangereuse, $\#uns$ est le nombre de n-rectangles dangereux trouvés par l'algorithme):

P	dim	#fbd	t s	t uns	#uns
ex	2	14	0	0	3
s2	2	16	0.01	0	15
s3	3	290	0.18	.01	4
s3'	3	80	0.64	0.02	0
l	3	158	0.08	0	0
3p	3	32	0	0	1
4p	4	190	0.09	0	1
5p	5	1048	0.82	0.02	1
6p	6	5482	5.82	0.13	1
7p	7	27668	42.35	0.86	1
16p	16	NA	NA	NA	1
32p	32	NA	NA	NA	1
64p	64	NA	NA	NA	1
128p	128	NA	NA	NA	1

Les processus `3p` à 128p sont les n philosophes avec n valant de 3 à 128. `l` est l'exemple de Lipsky et Papadimitriou (figure 2.9), `s3` et `s3'` ceux de la figure 2.8 (et `s2` en est une version bidimensionnelle), et `ex` est celui de la figure 2.5.

NA veut dire que l'analyseur prenait trop de temps ou même s'arrêtait sans pouvoir plus trouver de mémoire supplémentaire. On voit qu' au delà de sept processus, même simples, en parallèle, la méthode n'est plus pratiquable.

Quant à la deuxième méthode:

[1] Grâce à l'aide de Régis Cridlig.

[2] Cela fait l'objet d'une explication détaillée dans [FGR98b].

P	dim	#fbd	t s	t uns	#uns
ex	2	4	0.02	0	3
s2	2	6	0.02	0	15
s3	3	18	0.01	0	4
s3'	3	6	0.03	0	0
l	3	6	0.02	0	0
3p	3	3	0.02	0	1
4p	4	4	0.03	0	1
5p	5	5	0.03	0	1
6p	6	6	0.03	0	1
7p	7	7	0.04	0	1
16p	16	16	0.03	0.03	1
32p	32	32	0.03	0.42	1
64p	64	64	0.04	1.52	1
128p	128	128	0.10	26.49	1

On arrive assez facilement à 128 processus en parallèle. Mais on peut faire beaucoup mieux: avec l'implémentation de l'époque, on perd beaucoup de temps dans la recherche d'intersections d'hyperrectangles, alors qu'il existe de bien meilleures méthodes, étudiées essentiellement en imagerie.

3.1.2 Amélioration de l'implémentation

J'ai décrit les premières idées d'amélioration dans [FGR98b]. Deux étudiants de l'ENSTA en stage au CEA, I. Aparici et S. Morezuelas, en ont implémenté l'essentiel, mais il reste encore quelques éléments d'interface avant de pouvoir tester le gain pratique de la méthode. Je reprends pour l'essentiel les explications de [AM99] et de [FGR98b].

L'idée est de trouver une meilleure représentation d'ensemble de n-rectangles (inspirée de [SW82], [KO93] et de [PS93]). Plusieurs options se présentent:

- Arbre m-aire de recherche,
- Arbre d'intervalle ("Interval Tree"),
- Méthode conduisant à un problème du type "*Dominance Merge*",
- Arbre de Segments Statique ("Segment Tree"),
- Arbre de Segments Dynamique ("Dynamic Segment Tree").

On a assez vite abandonné les algorithmes à base d'arbre m-aire ou les problèmes de type *Dominance Merge* car la complexité des opérations élémentaires était trop importante. On n'a retenu par la suite que les arbres de segments et d'intervalles.

De fait, les arbres d'intervalles et les arbres de segments se ressemblent beaucoup. La différence entre les versions dynamiques et statiques est que dans la version statique, il faut décider une fois pour toute l'intervalle maximal représentable dans chaque coordonnée. On va poser pour toute la suite de la section d=nombre de processus et n=nombre d'hyperrectangles interdits. Pour illustrer ces méthodes, on se reportera à l'exemple en 2 dimensions de la figure 3.2.

Un arbre d'intervalles est une structure de données dont le squelette est un arbre binaire (appelé structure primaire), défini statiquement pour un ensemble donné de points (les extrêmes des intervalles) et qui stocke un nombre indéterminé d'intervalles. Chaque nœud de l'arbre a un discriminant (qui est la demi-somme du nœud plus à droite de son sous-arbre gauche et du plus à gauche de son sous-arbre droit). Il lui est aussi associé deux listes où on met les extrêmes des intervalles qui contiennent le discriminant (voir [PS93], section 8.8.1).

On a commencé par vouloir utiliser des arbres d'intervalles pour garder en mémoire les coordonnées des hyperrectangles de la région interdite en chaque dimension. Dans la figure 3.1 on a représenté le stockage pour la coordonnée x correspondant au problème de la figure 3.2.

Puis on s'est tourné vers les arbres de segments, très proches, qui ont l'avantage d'avoir une version dynamique connue [KO93]. Un arbre de segments est encore une fois un arbre binaire; chacun de ses nœuds contient un intervalle. L'union des intervalles des deux fils d'un nœud est égal à l'intervalle du nœud père. Chaque nœud possède également une liste de segments. L'insertion d'un segment dans un arbre de segments se fait comme suit: si le segment coïncide avec l'intervalle du nœud, on rajoute son nom à la liste du nœud. S'il est plus grand, il y a une erreur. S'il est plus petit, on insère dans le fils de gauche l'intersection entre l'intervalle du fils et le segment à insérer, tout en gardant le même nom, et de même avec celui de droite. Le résultat est que chaque nœud contient le nom du segment si et seulement si l'intervalle du nœud est contenu dans le segment mais pas celui du père du nœud. Evidemment, seuls les segments dont les bornes coïncident avec celles d'un intervalle d'une feuille de l'arbre peuvent être insérés. C'est un problème qui peut être résolu avec la version dynamique. Dans la figure 3.3 on peut voir la façon dont on stocke les intervalles dans l'axe X de l'exemple de la figure

3.2.

On peut montrer qu'alors la complexité d'insertion d'un hyperrectangle dans une collection de d arbres d'intervalles est de l'ordre de

$$O((2^d)^{log(2n)}) = O((2n)^d)$$

L'arbre de segments dynamique a pour structure sous-jacente un arbre Rouge-Noir. Les arbres Rouge-Noir permettent la scission et la concaténation, nécessaires pour incorporer des nouveaux extrema dans les feuilles en même temps que l'arbre reste équilibré. Pourtant, le besoin de faire des rotations fait que la structure sous-jacente n'est pas tout à fait un arbre de segment mais un arbre de segments faible, en ce sens que pour tout intervalle A, pour tout chemin allant de la racine à une feuille contenue dans A, il doit y avoir un nœud et un seul qui contienne A. Dans un arbre de segments on retrouve cette condition mais en plus le nœud doit être le plus près possible de la racine. Ainsi, avant chaque rotation, les intervalles contenus dans les nœuds à déplacer sont transférés à leurs fils. Dans la figure 3.4, on voit un exemple de rotation à gauche.

Un arbre de segments dynamique fait aussi appel à un dictionnaire pour effacer les intervalles et à une structure *Union-Copy*. L'appartenance d'un ensemble d'intervalles à un nœud pourrait se faire sous forme de listes. Pourtant cette structure est très peu rapide pour les opérations d'union et de copie, qui sont très importantes dans notre arbre. C'est pour cela que cette appartenance est matérialisée avec une structure beaucoup plus adaptée: l'*Union-Copy* qui est composée de deux sous-structures de type *Union-Find*, qui peuvent être implémentées de plusieurs façons (celle de [Tar72], mais on a un peu mieux pour l'Union-Copy, voir [Pou90]).

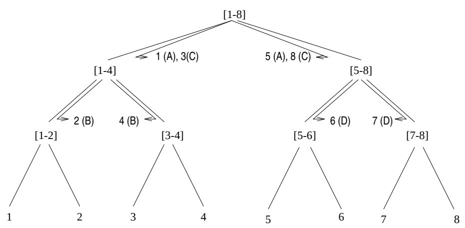


FIG. 3.1 – Exemple d'*arbre d'intervalles*

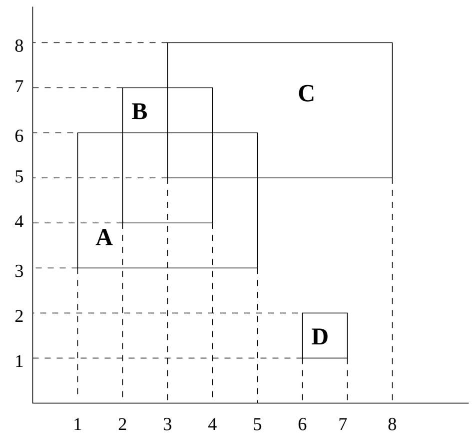


FIG. 3.2 – Un exemple de rectangles dans le plan.

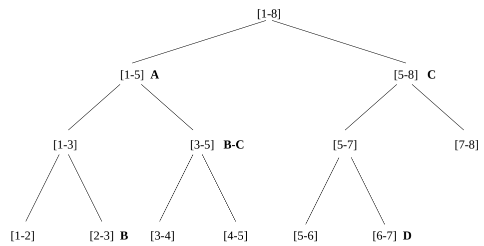


FIG. 3.3 – Exemple d'*arbre de segments*

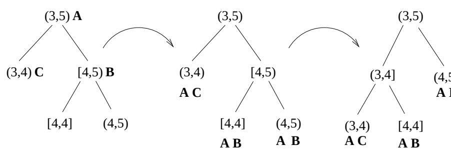


FIG. 3.4 – Rotation à gauche dans un arbre de segment dynamique

Bibliographie

[ABND+90] H. Attiya, A. Bar-Noy, D. Dolev, D. Peleg, and R. Reischuk. Renaming in an asynchronous environment. *Journal of the ACM*, 37(3):524–548, July 1990.

[Ait86] Ian R. Aitchison. The geometry of oriented cubes. Macquarie Mathematics Reports 86-0082, 1986.

[AM99] I. Aparici and S. Morezuelas. Détection rapide de points morts. Technical Report 99-056, CEA/LETI, 1999.

[Bas97a] K. Basu. The geometry of sequential computation I: A simplicial geometry of interaction. Technical report, Institutsbericht, Technische Universitaet Muenchen, Institut fuer Informatik, 1997.

[Bas97b] K. Basu. The geometry of sequential computation II: A simplicial geometry of interaction. Technical report, Institutsbericht, Technische Universitaet Muenchen, Institut fuer Informatik, 1997.

[Bed88] M. A. Bednarczyk. *Categories of asynchronous systems.* PhD thesis, University of Sussex, 1988.

[BG93] E. Borowsky and E. Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *Proc. of the 25th STOC.* ACM Press, 1993.

[BH81a] R. Brown and P. J. Higgins. Colimit theorems for relative homotopy groups. *Journal of Pure and Applied Algebra*, (22):11–41, 1981.

[BH81b] R. Brown and P. J. Higgins. On the algebra of cubes. *Journal of Pure and Applied Algebra*, (21):233–260, 1981.

[BH81c] Ronald Brown and Philip J. Higgins. The equivalence of ∞-groupoids and crossed complexes. *Cahiers Topologie Géom. Différentielle*, 22(4):371–386, 1981.

[BMZ88] O. Biran, S. Moran, and S. Zaks. A combinatorial characterization of the distributed tasks which are solvable in the presence of one faulty processor. In *Proc. 7th Annual ACM Symposium on Principles of Distributed Computing*, pages 263–275. ACM Press, August 1988.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximations of fixed points. *Principles of Programming Languages 4*, pages 238–252, 1977.

[CES71] E. G. Coffman, M. J. Elphick, and A. Shoshani. System deadlocks. *Computing Surveys*, 3(2):67–78, June 1971.

[CG93] R. Cridlig and E. Goubault. Semantics and analyses of Linda-based languages. In *Proc. of WSA'93*, number 724 in LNCS. Springer-Verlag, 1993.

[Cha90] S. Chaudhuri. Agreement is harder than consensus: set consensus problems in totally asynchronous systems. In *Proc. of the 9th Annual ACM Symposium on Principles of Distributed Computing*, pages 311–334. ACM Press, August 1990.

[Cri95] R. Cridlig. Semantic analysis of shared-memory concurrent languages using abstract model-checking. In *Proc. of PEPM'95*, La Jolla, June 1995. ACM Press.

[Cri96] R. Cridlig. Semantic analysis of Concurrent ML by abstract model-checking. In *Proceedings of the LOMAPS Workshop*, 1996.

[CRJ87] S. D. Carson and P. F. Reynolds Jr. The geometry of semaphore programs. *ACM Transactions on Programming Languages and Systems*, 9(1):25–53, January 1987.

[Dij68] E.W. Dijkstra. *Cooperating Sequential Processes.* Academic Press, 1968.

[Evr] M. Evrard. Homotopie des complexes simpliciaux et cubiques. Technical report, Université Paris 6.

[Faj00] L. Fajstrup. Loops, ditopology, and deadlocks. *Mathematical Structures in Computer Science*, 2000.

[FBS00] A. A. Al-Agl Fahd, R. Brown, and R. Steiner. Multiple categories: the equivalence of a globular and a cubical approach. Technical report, arXiv:math.CT/0007009, July 2000.

[FGR98a] L. Fajstrup, E. Goubault, and M. Raussen. Detecting deadlocks in concurrent systems. In *Proceedings of the 9th International Conference on Concurrency Theory, also available at http://www.dmi.ens.fr/~goubault.* Springer-Verlag, 1998.

[FGR98b] L. Fajstrup, E. Goubault, and M. Raussen. Detecting deadlocks in concurrent systems. Technical report, CEA, 1998.

[FGR99] L. Fajstrup, E. Goubault, and M. Raussen. Algebraic topology and concurrency. *submitted to Theoretical Computer Science, also technical report, Aalborg University*, 1999.

[FLP85] M. Fisher, N. A. Lynch, and M. S. Paterson. Impossibility of distributed commit with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[FS00] L. Fajstrup and S. Sokolowski. Infinitely running processes with loops from a geometric view-point. *Electronic Notes in Theoretical Computer Science, Proceedings of GETCO'00*, 2000.

[Gau00a] P. Gaucher. About the globular homology of higher dimensional automata. preprint available at math.CT/0002216, 2000.

[Gau00b] P. Gaucher. Combinatorics of branchings in higher dimensional automata. preprint available at math.CT/9912059, 2000.

[Gau00c] P. Gaucher. From concurrency to algebraic topology. *Electronic Notes in Computer Science*, 39, 2000.

[Gau00d] P. Gaucher. Homotopy invariants of higher dimensional categories and concurrency in computer science. *Mathematical Structures in Computer Science*, 2000.

[GJ92] E. Goubault and T. P. Jensen. Homology of higher-dimensional automata. In *Proc. of CONCUR'92*, Stonybrook, New York, August 1992. Springer-Verlag.

[GLG99] J. Goubault-Larrecq and E. Goubault. Order-theoretic, geometric and combinatorial models of intuitionistic S4 proofs. In *IMLA'99*, 1999.

[Gou93] E. Goubault. Domains of higher-dimensional automata. In *Proc. of CONCUR'93*, Hildesheim, August 1993. Springer-Verlag.

[Gou95a] E. Goubault. *The Geometry of Concurrency.* PhD thesis, Ecole Normale Supérieure, 1995. also available at http://www.dmi.ens.fr/~goubault.

[Gou95b] E. Goubault. Schedulers as abstract interpretations of HDA. In *Proc. of PEPM'95*, La Jolla, June 1995. ACM Press, also available at http://www.dmi.ens.fr/~goubault.

[Gou96a] E. Goubault. The dynamics of wait-free distributed computations. Technical report, Research Report LIENS-96-26, December 1996.

[Gou96b] E. Goubault. A semantic view on distributed computability and complexity. In *Proceedings of the 3rd Theory and Formal Methods Section Workshop.* Imperial

College Press, also available at http://www.dmi.ens.fr/~goubault, 1996.

[Gou96c] E. Goubault. Transitions take time. In *Proceedings of ESOP'96*, number 1058, pages 173–187. Springer-Verlag, 1996.

[Gou97] E. Goubault. Optimal implementation of wait-free binary relations. In *Proceedings of the 22nd CAAP*. Springer Verlag, 1997.

[Gou00] E. Goubault. Geometry and concurrency: A users' guide. *Mathematical Structures in Computer Science*, 2000.

[Gro91] J. R. J. Groves. Rewriting systems and homology of groups. In L. G. Kovacs, editor, *Groups – Canberra 1989*, number 1456, pages 114–141. Lecture notes in Mathematics, Springer-Verlag, 1991.

[Gun94] J. Gunawardena. Homotopy and concurrency. In *Bulletin of the EATCS*, number 54, pages 184–193, October 1994.

[GW91] P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In *Proc. of the Third Workshop on Computer Aided Verification*, volume 575, pages 417–428. Springer-Verlag, Lecture Notes in Computer Science, July 1991.

[GZ67] P. Gabriel and M. Zisman. Calculus of fractions and homotopy theory. In *Ergebnisse der Mathematik und ihrer Grenzgebiete*, volume 35. Springer Verlag, 1967.

[HR94] M. Herlihy and S. Rajsbaum. Set consensus using arbitrary objects. In *Proc. of the 13th Annual ACM Symposium on Principles of Distributed Computing*. ACM Press, August 1994.

[HR95] Maurice Herlihy and Sergio Rajsbaum. Algebraic spans (preliminary version). In *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 90–99, Ottawa, Ontario, Canada, 2–23 August 1995.

[HR00] M. Herlihy and S. Rajsbaum. Algebraic spans. *Mathematical Structures in Computer Science*, 2000.

[HS93] M. Herlihy and N. Shavit. The asynchronous computability theorem for t-resilient tasks. In *Proc. of the 25th STOC*. ACM Press, 1993.

[HS94] M. Herlihy and N. Shavit. A simple constructive computability theorem for wait-free computation. In *Proceedings of STOC'94*. ACM Press, 1994.

[Jay93] Prasad Jayanti. On the robustness of Herlihy's hierarchy. In *Proceedings of the Twelth Annual ACM Symposium on Principles of Distributed Computing*, pages 145–157, Ithaca, New York, USA, August 1993.

[Jay97] Prasad Jayanti. Robust wait-free hierarchies. *Journal of the ACM*, 44(4):592–614, July 1997.

[Joh82] P. T. Johnstone. *Stone Spaces*. Cambridge University Press, 1982.

[Kah74] G. Kahn. The semantics of a simple language for parallel programming. *Information Processing*, (74), 1974.

[KM77] G. Kahn and D. B. MacQueen. Coroutines and networks of parallel processes. *Information Processing*, (77), 1977.

[KO93] Marc J. Van Kreveld and Mark H. Overmars. Union-copy structures and dynamic segment trees. *Journal of the Association for Computing Machinery*, 40(3), July 1993.

[Kob90] Y. Kobayashi. Complete rewriting systems and homology of monoid algebras. *Journal of Pure and Applied Algebra*, 65:263–275, 1990.

[KV91] M. M. Kapranov and V. A. Voevodsky. ∞-groupoids and homotopy types. *Cahiers Topologie Géom. Différentielle Catégoriques*, 32(1):29–46, 1991. International Category Theory Meeting (Bangor, 1989 and Cambridge, 1990).

[Laf95] Y. Lafont. A new finiteness condition for monoids presented by complete rewriting systems. *Journal of Pure and Applied Algebra*, 98, 1995.

[LP81] Lipski and Papadimitriou. A fast algorithm for testing for safety and detecting deadlocks in locked transaction. *ALGORITHMS: Journal of Algorithms*, 1981.

[Lyn96] N. Lynch. *Distributed Algorithms.* Morgan-Kaufmann, 1996.

[May67] J. P. May. *Simplicial objects in algebraic topology.* D. van Nostrand Company, inc, 1967.

[Maz86] A. Mazurkiewicz. Trace theory. In G. Rozenberg, editor, *Petri Nets: Applications and Relationship to Other Models of Concurrency, Advances in Petri Nets 1986, PART II, PO of an Advanced Course*, volume 255 of *LNCS*, pages 279–324, Bad Honnefs, September 1986. Springer-Verlag.

[Maz88] A. Mazurkiewicz. Basic notions of trace theory. In *Lecture notes for the REX summer school in temporal logic*. Springer-Verlag, 1988.

[MC85] J. Mc Cleary. *User's guide to spectral sequences.* Publish or perish, Mathematics lecture series 12, 1985.

[Mét94] F. Métayer. Homology of proofnets. *Archive of Mathematical Logic*, 33:169–188, 1994.

[Mét95] F. Métayer. Volume of multiplicative formulas and provability. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 297–306. Cambridge University Press, 1995. Proceedings of the Workshop on Linear Logic, Ithaca, New York, June 1993.

[ML71] S. Mac Lane. *Categories for the working mathematician.* Springer-Verlag, 1971.

[Nac65] L. Nachbin. *Topology and Order.* Van Nostrand, Princeton, 1965.

[Pap79] Christos H. Papadimitriou. The serializability of concurrent database updates. *Journal of the ACM*, 26(4):631–653, October 1979.

[Pap83] Christos H. Papadimitriou. Concurrency control by locking. *SIAM Journal on Computing*, 12(2):215–226, 1983.

[Pen72] R. Penrose. *Techniques of Differential Topology in Relativity*, volume 7 of *Conference Board of the Mathematical Sciences, Regional Conference Series in Applied Ma thematics.* SIAM, Philadelphia, USA, 1972.

[Pou90] J. L. Poutre. New techniques for the union-find problem. In *Proceedings of the first Annual Symposium on Discrete Algorithms*, pages 54–63, 1990.

[Pra91] V. Pratt. Modeling concurrency with geometry. In *Proc. of the 18th ACM Symposium on Principles of Programming Languages.* ACM Press, 1991.

[Pra94] V. Pratt. Chu spaces: Automata with quantum aspects. Technical report, Stanford University, 1994.

[PS93] F. P. Preparata and M. I. Shamos. *Computational Geometry, an Introduction.* Springer-Verlag, 1993.

[Rau00] M. Raussen. On the classification of dipaths in geometric models for concurrency. *Mathematical Structures in Computer Science*, August 2000.

[SC70] A. Shoshani and E. G. Coffman. Sequencing tasks in multiprocess systems to avoid deadlocks. In *Conference Record of 1970 Eleventh Annual Symposium on Switching and Automata Theory*, pages 225–235, Santa Monica, California, Oct 1970. IEEE.

[Sch97] Eric Schenk. The consensus hierarchy is not robust. In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, page 279, Santa Barbara, California, 21–24 August 1997.

[Ser51] J.P. Serre. *Homologie Singulière des Espaces Fibrés. Applications.*

PhD thesis, Ecole Normale Supérieure, 1951.

[Shi85] M.W. Shields. Concurrent machines. *Computer Journal*, 28, 1985.

[SNW94] V. Sassone, M. Nielsen, and G. Winskel. Relationships between models of concurrency. In *Proceedings of the Rex'93 school and symposium*, 1994.

[SOK94] C. C. Squier, F. Otto, and Y. Kobayashi. A finiteness condition for rewriting systems. *Theoretical Computer Science*, 131:271–294, 1994.

[Sok98a] S. Sokolowski. Homotopy in concurrent processes. Technical report, Institute of Computer Science, Gdansk Division, 1998.

[Sok98b] S. Sokolowski. Investigation of concurrent processes by means of homotopy functors. Technical report, Institute of Computer Science, Gdansk Division, 1998.

[Sok98c] S. Sokolowski. Point glueing in cpo-s. Technical report, Institute of Computer Science, Gdansk Division, 1998.

[Sok99] S. Sokolowski. Classifying holes of arbitrary dimensions in partially ordered cubes. Technical report, Personal Communication, 1999.

[Spa66] E. J. Spanier. *Algebraic Topology*. McGraw Hill, 1966.

[Squ87] Craig Squier. Word problems and a homological finiteness condition for monoids. *J. of Pure and Applied Algebra*, 49:201–217, 1987.

[SSW85] E. Soisalon-Soininen and D. Wood. An optimal algorithm for testing for safety and detecting deadlocks in locked transaction systems. In *Symposium on Principles of Database Systems (PODS '82)*, pages 108–116, March 1985.

[Sta89] A. Stark. Concurrent transition systems. *Theoretical Computer Science*, 64:221–269, 1989.

[Ste91] Richard Steiner. Tensor products of infinity-categories. University of Glasgow, 1991.

[Str87] R. Street. The algebra of oriented simplexes. *J. Pure Appl. Algebra*, (49):283–335, 1987.

[SW82] W. H. Six and D. Wood. Counting and reporting intersections of d-ranges. *IEEE Transactions on Computers*, 31(3):181–187, March 1982.

[SZ93] M. Saks and F. Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. In *Proc. of the 25th STOC*. ACM Press, 1993.

[Tak95] Y. Takayama. Cycle filling as parallelization with expansion law. In *submitted to publication*, 1995.

[Tak96] Y. Takayama. Extraction of concurrent processes from higher-dimensional automata. In *Proceedings of CAAP'96*, pages 72–85, 1996.

[Tar72] Robert André Tarjan. On the efficiency of a good but not linear set-union algorithm. Technical Report TR 72-148, Computer Science Dept. Cornell University, Ithaca, N.Y. 14850, November 1972.

[Val90] A. Valmari. A stubborn attack on state explosion. In *Proc. of CAV'90*. Springer Verlag, LNCS, 1990.

[vGG89] R. van Glabbeek and U. Goltz. Partial order semantics for refinement of actions. *Bulletin of the EATCS*, (34), 1989.

[WN94] G. Winskel and M. Nielsen. *Models for concurrency*, volume 3 of Handbook of Logic in Computer Science, pages 100–200. Oxford University Press, 1994.

[YPK79] Mihalis Yannakakis, C. H. Papadimitriou, and H. T. Kung. Locking policies: Safety and freedom from deadlock. In *20th Annual Symposium on Foundations of Computer Science*, pages 286–297, San Juan, Puerto Rico, 29–31 October 1979. IEEE.

Table des matières