Un sous-système de recherche géométrique et d'équivalence pour la CAO de circuits intégrés VLSI

Josette Toussan Berger

THESE

présentée par

Josette TOUSSAN ep. BERGER

pour obtenir le grade de **Docteur**

de l'INSTITUT NATIONAL POLYTECHNIQUE de GRENOBLE

(arrêté ministériel du 5 juillet 1984)

spécialité : Informatique

UN SOUS-SYSTEME DE RECHERCHE GEOMETRIQUE ET D'EQUIVALENCE POUR LA CAO DE CIRCUITS INTEGRES VLSI

Soutenue le 21 Mars 1988 devant la commission d'examen :

Jury

Jacques Mossière, *Président,*

Patrick Dewilde, *Rapporteur,*

Roland Gerber, *Rapporteur,*

Jacques Lecourvoisier,

Guy Mazaré

Laboratoire de Génie Informatique de l'Institut des Mathématiques Appliquées de Grenoble

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : Georges LESPINARD **Année 1988**

Professeurs des Universités

BARIBAUD Michel	ENSERG
BARRAUD Alain	ENSIEG
BAUDELET Bernard	ENSPG
BEAUFILS Jean-Pierre	ENSEEG
BLIMAN Samuel	ENSERG
BLOCH Daniel	ENSPG
BOIS Philippe	ENSHMG
BONNETAIN Lucien	ENSEEG
BOUVARD Maurice	ENSHMG
BRISSONNEAU Pierre	ENSIEG
BRUNET Yves	IUFA
CAILLERIE Denis	ENSHMG
CAVAIGNAC Jean-François	ENSPG
CHARTIER Germain	ENSPG
CHENEVIER Pierre	ENSERG
CHERADAME Hervé	UFR PGP
CHOVET Alain	ENSERG
COHEN Joseph	ENSERG
COUMES André	ENSERG
DARVE Félix	ENSHMG
DELLA-DORA Jean	ENSIMAG
DEPORTES Jacques	ENSPG
DOLMAZON Jean-Marc	ENSERG
DURAND Francis	ENSEEG
DURAND Jean-Louis	ENSIEG
FOGGIA Albert	ENSIEG
FONLUPTJean	ENSIMAG
FOULARD Claude	ENSIEG
GANDINI Alessandro	UFR PGP
GAUBERT Claude	ENSPG
GENTIL Pierre	ENSERG
GREVEN Hélène	IUFA
GUERIN Bernard	ENSERG
GUYOT Pierre	ENSEEG
IVANES Marcel	ENSIEG
JAUSSAUD Pierre	ENSIEG
JOUBERT Jean-Claude	ENSPG
JOURDAIN Geneviève	ENSIEG
LACOUME Jean-Louis	ENSIEG
LESIEUR Marcel	ENSHMG
LESPINARD Georges	ENSHMG
LONGEQUEUE Jean-Pierre	ENSPG
LOUCHET François	ENSIEG
MASSE Philippe	ENSIEG
MASSELOT Christian	ENSIEG
MAZARE Guy	ENSIMAG
MOREAU René	ENSHMG
MORET Roger	ENSIEG
MOSSIERE Jacques	ENSIMAG
OBLED Charles	ENSHMG
OZIL Patrick	ENSEEG
PARIAUD Jean-Charles	ENSEEG
PERRET René	ENSIEG
PERRET Robert	ENSIEG
PIAU Jean-Michel	ENSHMG
POUPOT Christian	ENSERG
RAMEAU Jean-Jacques	ENSEEG
RENAUD Maurice	UFR PGP
ROBERT André	UFR PGP
ROBERT François	ENSIMAG
SABONNADIERE Jean-Claude	ENSIEG
SAUCIER Gabrielle	ENSIMAG
SCHLENKER Claire	ENSPG
SCHLENKER Michel	ENSPG
SILVY Jacques	UFR PGP
SIRIEYS Pierre	ENSHMG
SOHM Jean-Claude	ENSEEG
SOLER Jean-Louis	ENSIMAG
SOUQUET Jean-Louis	ENSEEG
TROMPETTE Philippe	ENSHMG
VEILLON Gérard	ENSIMAG
ZADWORNY François	ENSERG

Professeur Université des Sciences Sociales (Grenoble II)

BOLLIET Louis

Personnes ayant obtenu le diplôme d'HABILITATION A DIRIGER DES RECHERCHES

BECKER Monique
BINDER Zdenek
CHASSERY Jean-Marc
CHOLLET Jean-Pierre
COEY John
COLINET Catherine
COMMAULT Christian
CORNUEJOLS Gérard
COULOMB Jean- Louis
DALARD Francis
DANES Florin
DEROO Daniel
DIARD Jean-Paul
DION Jean-Michel
DUGARD Luc
DURAND Madeleine
DURAND Robert
GALERIE Alain
GAUTHIER Jean-Paul
GENTIL Sylviane
GHIBAUDO Gérard
HAMAR Sylvaine
HAMAR Roger
LADET Pierre
LATOMBE Claudine
LE GORREC Bernard
MADAR Roland
MULLER Jean
NGUYEN TRONG Bernadette
PASTUREL Alain
PLA Fernand
ROUGER Jean
TCHUENTE Maurice
VINCENT Henri

Chercheurs du C.N.R.S
Directeurs de recherche 1ère Classe

CARRE René
FRUCHART Robert
HOPFINGER Emile
JORRAND Philippe
LANDAU Ioan
VACHAUD Georges
VERJUS Jean-Pierre

Directeurs de recherche 2ème Classe

ALEMANY Antoine
ALLIBERT Colette
ALLIBERT Michel
ANSARA Ibrahim
ARMAND Michel
BERNARD Claude
BINDER Gilbert
BONNET Roland
BORNARD Guy
CAILLET Marcel
CALMET Jacques
COURTOIS Bernard
DAVID René
DRIOLE Jean
ESCUDIER Pierre
EUSTATHOPOULOS Nicolas
GUELIN Pierre
JOUD Jean-Charles
KLEITZ Michel
KOFMAN Walter
KAMARINOS Georges
LEJEUNE Gérard
LE PROVOST Christian
MADAR Roland
MERMET Jean
MICHEL Jean-Marie
MUNIER Jacques
PIAU Monique
SENATEUR Jean-Pierre
SIFAKIS Joseph
SIMON Jean-Paul
SUERY Michel
TEODOSIU Christian
VAUCLIN Michel
WACK Bernard

Personnalités agrées à titre permanent à diriger des travaux de recherche (décision du conseil scientifique)

E.N.S.E.E.G

CHATILLON Christian
HAMMOU Abdelkader
MARTIN GARIN Régina
SARRAZIN Pierre
SIMON Jean-Paul

E.N.S.E.R.G

BOREL Joseph

E.N.S.I.E.G

DESCHIZEAUX Pierre
GLANGEAUD François
PERARD Jacques
REINISCH Raymond

E.N.S.H.G

ROWE Alain

E.N.S.I.M.A.G

COURTIN Jacques

E.F.P.

CHARUEL Robert

C.E.N.G

CADET Jean
COEURE Philippe
DELHAYE Jean-Marc
DUPUY Michel
JOUVE Hubert
NICOLAU Yvan
NIFENECKER Hervé
PERROUD Paul
PEUZIN Jean-Claude
TAIB Maurice
VINCENDON Marc

Laboratoires extérieurs

C.N.E.T

DEVINE Rodericq
GERBER Roland
MERCKEL Gérard
PAULEAU Yves

AVANT PROPOS

Le travail présenté dans cette thèse a été effectué au Centre National d'Etudes des Télécommunications de Meylan (CNET-CNS), dans le cadre d'une collaboration avec le Laboratoire de Génie Informatique, laboratoire rattaché à l'Institut des Mathématiques appliquées de Grenoble.

Je tiens à exprimer toute ma reconnaissance Monsieur G. MAZARE, Professeur à l'Institut National Polytechnique de Grenoble pour m'avoir accueillie dans son équipe de recherche, au sein du Laboratoire de Génie Informatique, et permis, par son encadrement scientifique, de mener à bien cette étude.

Je voudrais exprimer ma reconnaissance à Monsieur R. GERBER, Chef de la division "Conception de Circuits Intégrés" du CNET-CNS, ainsi que Monsieur J. LECOURVOISIER, Chef du département "Recherche en Conception Assistée" (RCA), pour m'avoir accueillie au sein du département RCA. Ils m'ont ainsi permis d'acquérir une bonne expérience dans un domaine passionnant.

Je tiens à remercier particulièrement les membres du jury :

Monsieur J. MOSSIERE, Directeur du Laboratoire de Génie Informatique, de me faire l'honneur de présider le jury de soutenance,

Monsieur P. DEWILDE, Professeur à l'Université Technologique de DELFT (Pays-Bas), pour l'attention qu'il a porté à ces recherches, et pour avoir bien voulu être rapporteur de cette thèse,

Monsieur R. GERBER, Chef de la division "Conception des Circuits Intégrés" du CNET-CNS et Professeur, pour avoir bien voulu accepter d'être rapporteur de cette thèse,

Monsieur J. LECOURVOISIER, Chef du département "Recherche en Conception Assistée", qui a relu effectivement le manuscript de ce mémoire.

Merci également à tous les membres du CNET-CNS qui, de près ou de loin, m'ont aidée pour la réalisation de mes études; merci à la direction du CNET-CNS en général, pour l'environnement exceptionnel, matériel et technique, qu'elle a offert à ces recherches.

SOMMAIRE

UN SOUS-SYSTEME DE RECHERCHE GEOMETRIQUE ET D'EQUIVALENCE POUR LA CAO DE CIRCUITS INTEGRES VLSI

INTRODUCTION

Comme la complexité des circuits VLSI croît, la tâche de vérification des masques devient de plus en plus longue. Heureusement, l'utilisation des descriptions hiérarchiques de ces masques, permet le développement de méthodes qui utilisent avantageusement de telles descriptions. Pour les conceptions actuelles, ces méthodes sont potentiellement plus rapides que les méthodes qui ignorent la hiérarchie et commencent par une expansion de la description.

Cette thèse explore une méthode de **recherche géométrique** par **hiérarchie de quadtrees** et l'utilisation hiérarchique de l'algorithme **union-find** [AHO 74] pour la recherche et la mise à jour des équivalences électriques. Des algorithmes hiérarchiques et géométriques utilisables pour l'édition, la vérification géométrique et topologique (connectique) de conceptions VLSI sont présentés [BER 86]. L'intérêt de telles méthodes dans la vérification et le compactage de ces conceptions est aussi abordé.

Des **quadtrees adaptatifs** particuliers (**mixed-quadtrees**) ont été conçus et testés pour optimiser les outils conventionnels d'extraction et de vérification du **layout**(1) de conceptions non structurées et pour permettre une **édition** rapide de conceptions structurées. L'utilisation de ces outils dans un système de CAO a largement contribué à la validation de cette méthode géométrique d'accès par les concepteurs.

Un **extracteur** et **vérificateur** géométrique et topologique **de conceptions structurées** est actuellement réalisé dans le but d'évaluer hiérarchiquement l'efficacité de tels algorithmes. Cependant, seule l'utilisation incrémentale et même interactive de tels outils de CAO pour la réalisation de conceptions très diverses, permettra une évaluation réelle de la réduction du temps global de conception (efficacité réelle).

Les méthodes hiérarchiques, géométriques et topologiques qui ont ainsi été explorées, ont les propriétés suivantes :

1. Elles sont applicables à un grand nombre de tâches, comme l'**extraction** des conceptions, leur **vérification**, leur **édition** et l'optimisation de leur surface par **compactage**.

2. Ces méthodes opèrent sur les descriptions des masques selon des méthodologies de description symboliques ou réelles (layouts stick ou layouts physiques).

3. Ces méthodes ont pour seule restriction une certaine fonctionnalité des blocs ou symboles utilisés dans la description hiérarchique des conceptions analysées et en conséquence peuvent être utilisées facilement par un grand nombre de systèmes de conceptions VLSI

---

(1) Description de l'ensemble des masques de la conception

4. Ces méthodes sont applicables à des tâches interactives et permettent l'utilisation **dynamique** - au fur et à mesure de l'entrée des données - d'un certain nombre d'algorithmes d'extraction, de vérification et d'adaptation des conceptions éditées à un ensemble de règles technologiques.

Ce document présente les arguments qui ont contribué à la conception d'un tel sous-système de recherche géométrique et topologique, le sous-système lui-même et certains algorithmes hiérarchiques en cours de développement, ainsi que les résultats déjà obtenus (mesures de performances). Un aperçu des principaux problèmes que ce sous-système peut également aider à résoudre est également discuté dans l'ensemble des outils d'aide à la conception des circuits intégrés VLSI.

Le **chapitre 1** présente la tâche de conception d'un circuit VLSI. Il explique le principe et l'intérêt de la conception structurée. Il aborde enfin sommairement les problèmes spécifiques de **l'analyse** de la description géométrique ainsi produite.

Le **chapitre 2** étudie les diverses méthodes déjà proposées dans la littérature scientifique pour l'**analyse hiérarchique** des conceptions. Ces méthodes ont en commun la nécessité de réduire la complexité de l'analyse. Ainsi certaines de ces méthodes analysent directement un layout structuré conçu selon des régles de conception restrictives. D'autres méthodes n'imposent au contraire aucun style de conception et procédent à une transformation du layout ainsi structuré par le concepteur en un layout structuré selon des régles restrictives. Ce chapitre présente aussi le principe des divers algorithmes d'analyse hiérarchique. On évalue sur une hiérarchie théorique le comportement global des **analyses incrémentales**, réalisées au fur et à mesure de la conception du circuit; divers critères permettent ainsi d'assurer l'efficacité globale de ces analyses. L'efficacité de l'analyse hiérarchique est ainsi déduite de l'efficacité des algorithmes de base et du filtrage hiérarchique des éléments du layout effectivement vérifiés, lors d'un test incrémental.

Le **chapitre 3** étudie de manière détaillée les divers **algorithmes géométriques** et **d'équivalence** qui peuvent être utilisés dans l'analyse et la vérification des masques d'un circuit. Ces algorithmes sont étudiés dans le but de trouver ceux qui peuvent être adaptés efficacement aux descriptions hiérarchiques. Il peut être intéressant de construire une solution dynamique tout au long du processus d'entrée des données. Les algorithmes qui s'adaptent à la conception structurée et permettent de plus une gestion dynamique des données sont donc particulièrement recherchés et analysés quand à leur efficacité.

Le **chapitre 4** présente le **sous-système de recherche géométrique** conçu pour l'édition et l'analyse de layouts structurés. Les **mixed-quadtrees** ainsi proposés sont également évalués.

Le **chapitre 5** présente le **sous-système de recherche d'équivalence électrique** conçu pour l'analyse **dynamique** de layouts structurés. L'efficacité théorique de l'algorithme d'équivalence **union-find**, présenté au chapitre 3, est bien connue [TAR 75]. Une extension de l'algorithme *union-find* aux hiérarchies de conception est donc ici proposée afin d'obtenir une **représentation compacte** et hiérarchique du **réseau électrique** d'un layout structuré. Cette représentation associée au sous-système de recherche géométrique permet de reconnaître, de

manière hiérarchique, si deux éléments du layout, plus ou moins imbriqués dans la hiérarchie de conception, sont électriquement connectés.

Le **chapitre 6** présente l'utilisation des sous-systèmes de recherche géométrique et d'équivalence proposés pour les diverses fonctions de la CAO. Il propose ainsi un algorithme général d'analyse du layout d'un circuit. On présente aussi la structure de données utilisée pour construire la *représentation compacte* du réseau des nœuds électriques d'un layout hiérarchique. Cette représentation du réseau électrique associé au layout préserve la hiérarchie à laquelle elle est associée; cette *représentation compacte* et hiérarchique est aussi appelée le **réseau compact** associé au layout. Un algorithme de vérification hiérarchique des régles de dessin est ensuite proposé. Cet algorithme prend en compte le facteur de régularité des layouts structurés. Afin d'améliorer cette vérification hiérarchique, divers types de vérification des régles de dessin sont considérés. Des algorithmes utilisant les sous-systèmes de recherche géométrique et d'équivalence, sont ensuite proposés pour réaliser diverses fonctions interactives d'édition, de contrôle de la connectivité electrique et d'arrangement de layouts structurés.

Le **chapitre 7** résume les divers facteurs d'efficacité de la méthode proposée pour l'analyse hiérarchique et incrémentale du layout. Le sous-système de recherche géométrique a permis une amélioration sensible des outils conventionnels d'analyse de layouts non structurés et a ainsi prouvé l'efficacité de ce sous-système pour l'analyse de ces layouts. Son intégration a été faite dans un éditeur généralisé de circuits pour le réalisation des fonctions de base; cet éditeur réalise l'une des interfaces de la base de données *COSMIC (projet Europeen CVS);* l'extraction de la connectivité électrique par recherche géométrique permet actuellement une édition simplifiée et efficace des diverses cellules qui composent un layout structuré.

L'évaluation expérimentale du comportement des algorithmes hiérarchiques sur des cas réels demande la réalisation d'une interface avec la base de données COSMIC, actuellement utilisable dans sa première version. L'étude du comportement des algorithmes d'analyse hiérarchique proposés sur un ensemble de circuits très divers confirmera leur efficacité théorique et en permettra une optimisation globale. L'intégration complète, dans un éditeur de circuit, des sous-systèmes de recherche géométrique et d'équivalence va permettre une extension interactive de l'analyse hiérarchique ainsi que l'adaptation dynamique de layouts flexibles à un ensemble de règles technologiques.

Chapitre 1

METHODES DE CONCEPTION ET CAO DE VLSI

1. INTRODUCTION

La complexité des circuits intégrés croît de manière prodigieuse, surpassant les capacités des concepteurs à utiliser efficacement les différents dispositifs qui peuvent être placés dans un circuit. Beaucoup d'efforts sont dépensés pour construire des systèmes de conception qui aident les concepteurs à produire des descriptions de circuits correctes et efficaces. Vérifier à la fin de la conception que le circuit produit correspond bien aux intentions du concepteur est un procédé coûteux à la fois en temps calcul et en temps de conception (homme x année), puisque la correction des erreurs devient alors une tâche fastidieuse. Les outils de CAO sont conçus pour permettre de produire rapidement une description correcte du circuit à partir des idées des concepteurs de systèmes.

Certains des problèmes clefs qui doivent être résolus dans la CAO des VLSI sont connus être **NP-complets**, c'est-à-dire pratiquement insolubles dans leur généralité [BEN 81 & OTT] (1). Cependant les descriptions des circuits VLSI produisent en moyenne des distributions de données assez uniformes, et il existe donc, pour la plupart des tâches, des algorithmes efficaces [BEN 80 & HAK]. Mais même avec de bons algorithmes, le temps consacré à la vérification des circuits VLSI actuels est exorbitant. A cause de la complexité de ces circuits, il est nécessaire de développer des méthodes de conception structurée et des outils d'analyse hiérarchique des descriptions hiérarchiques ainsi obtenues. Les algorithmes qui ont un bon comportement pour les conceptions non structurées peuvent avoir un comportement très différent pour les conceptions structurées. Ainsi la résolution des problèmes clefs doit être à nouveau complétement étudiée pour construire ces outils d'analyse hiérarchique.

2. ANALYSE DE LA TACHE DE CONCEPTION

La **conception** d'un circuit intégré est un processus itératif. L'architecture du système est spécifiée d'abord, suivie par la conception du circuit lui-même. Le découpage de la conception en **blocs fonctionnels**, leur **placement** et **connexion** (routing) utilisent l'expérience et le savoir-faire des concepteurs, ainsi que des outils de CAO spécifiques à ces tâches. Le résultat de ce processus de conception récursif est une description géométrique des formes, qui doivent apparaître sur la surface du silicium. Cette **description géométrique** permet de réaliser les **masques du circuit** [MEA 80]. Ces masques sont alors utilisés pour la réalisation des **motifs**(2) sur les différentes couches d'une **tranche de silicium**. La phase de **fabrication** conduit à une tranche de silicium contenant de nombreux exemplaires du circuit; ces exemplaires sont coupés en circuits individuels et chacun d'eux est monté dans un boîtier et testé.

Les concepteurs doivent produire une description des masques en accord avec les **règles de conception** liées à la technologie utilisée et aussi libre que possible des erreurs fonctionnelles, logiques, topologiques ou géométriques. Ils utilisent pour cela actuellement certains symbolismes de descriptions [MEA 80], [WES 81],[CHE 86].

(1) La complexité de manipuler des descriptions hiérarchiques est examinée par [BEN 81 & OTT], [SAH 80].

(2) motif : partie connexe d'une couche (layer) ou atome d'une telle partie.

De nombreux programmes opèrent sur la description symbolique ou physique d'un CI et aident à détecter les diverses erreurs. Ces programmes effectuent en particulier les tâches d'**extraction de graphe électrique** et de **vérification des règles de conception** topologiques et géométriques, ainsi que l'**extraction des paramètres électriques.**

La **vérification des règles de conception** (Design Rule Checking / **DRC**) signifie la vérification d'un ensemble de règles de conception liées à la technologie utilisée pour la fabrication des tranches de silicium. Ces règles spécifient habituellement largeur minimum, espacement, recouvrement et débordement minimum des divers motifs des différentes couches (layers) de la description des masques [LIN 76]. Ces règles de conception assurent ainsi la réalisation, les connexions et isolations des différents dispositifs électriques décrits de manière géométrique par leurs masques.

L'**extraction du circuit** est le processus de conversion de la description géométrique du circuit ainsi réalisée en une description électriquement équivalente et constituée de transistors et d'interconnexions. Cette description électrique peut être utilisée par les **simulateurs**. Plus de détails sur les performances du circuit décrit ainsi géométriquement doivent être obtenus pour des simulations de niveau circuit; l'**extraction des paramètres électriques** des éléments du circuit (résistance, capacité des nœuds d'interconnexion ; W/L des transistors ...) est alors nécessaire.

Ces programmes d'analyse de la conception traitent un très grand nombre de motifs géométriques et en extraient l'information utile. Dans les conceptions NMOS / CMOS, il y a environ dix formes géométriques, généralement rectangulaires, pour décrire chaque transistor. Pour une conception de 100 000 transistors, 1 000 000 de rectangles apparaissent alors sur l'ensemble des masques utilisés pour réaliser le circuit. Supposons les rectangles spécifiés par leurs coordonnées supérieure gauche et inférieure droite et un champ qui spécifie la couche / couleur de chaque rectangle ; 17 M bytes de mémoire sont alors utilisés par une telle description ! ((8 + 8 + 1) x 100000 x 10). Les algorithmes naïfs qui recherchent les interactions d'un rectangle courant avec les (N - 1) autres rectangles ont des temps d'exécution proportionnels à N^2. Même des "algorithmes intelligents" mais travaillant à partir d'une liste explicite de rectangles (ou polygones simples) exhibent des temps d'exécution prohibitifs pour des conceptions de taille assez modeste.

3. ROLE DE LA HIERARCHIE DANS LA CONCEPTION ET L'ANALYSE

Le partage d'ensembles complexes en sous-ensembles plus simples, constitue une méthode appropriée pour la conception et l'analyse de tels ensembles ; le processus de division itérative ramène à la conception ou l'analyse d'ensembles de plus en plus simples. C'est ainsi que les méthodes de "**construction itérative de la solution**" et de "**division du problème**", représentent les deux principales méthodes de conception et d'analyse de problèmes complexes. Chaque sous-ensemble est alors constitué de plus petits sous-ensembles et ainsi récursivement, jusqu'à ce que les plus petits sous-ensembles soient atteints.

De telles conceptions hiérarchiques permettent de concevoir des ensembles compliqués et de les manipuler comme de simples entités ; les détails de ces entités sont alors ignorés jusqu'à ce qu'on en ait besoin. Les hiérarchies ont donc été utilisées implicitement ou explicitement depuis longtemps pour la conception et la spécification de systèmes. Ces méthodes de conception, description et analyse sont donc aussi utilisées explicitement pour la conception des circuits intégrés VLSI.

Dans une description hiérarchique, le circuit est spécifié comme la composition d'un certain nombre d'éléments. Chacun de ses éléments constitutifs est soit une forme primaire non décomposable dite **élément primaire**, soit une **référence** à une collection d'éléments primaires, communément appelée **un symbole**, une **cellule** ou un **modèle**. Ces références ou **occurrences de symboles** sont appelées **instances du symbole ou sous-cellules**; elles spécifient la place, le nom et l'orientation du symbole. Une conception compliquée est ainsi construite par la hiérarchie des symboles en référence. Si nous considérons la liste d'**éléments primaires** ou **atomes de géométrie** (polygones ou rectangles) qui décrivent un circuit intégré, nous pouvons comparer la longueur de cette description sans hiérarchie à une description hiérarchique du même circuit. Pour un même circuit, les descriptions hiérarchiques peuvent être beaucoup plus courtes que leur description non hiérarchique, lorsque le circuit a un haut degré de régularité; les répétitions réalisées sous la forme de multiples instances ou appels du même symbole indiquent alors clairement quelles sont les parties de la conception qui sont identiques.

Une fois que la description hiérarchique d'un circuit est ainsi faite, certaines occasions peuvent se présenter de réduire **l'analyse** (extraction, vérification) de cette description. La clef est d'éviter tout travail redondant. Un symbole vérifié une fois n'a pas besoin de l'être à nouveau pour chacune de ses occurrences; pour chaque instance du symbole il suffira d'examiner son interaction dans le contexte d'appel (de sérieuses difficultés surviennent en général lorsqu'un élément est placé juste dessus d'une de ses occurrences). Le temps gagné par la non analyse des cellules ainsi répétées est souvent conséquent même lorsque ces cellules ne sont pas répétées un grand nombre de fois.

Les programmes d'analyse de la description géométrique d'un circuit sont assez simples dans leur **mode conventionnel** (non hiérarchique). Ils examinent chaque fragment / atome de la description géométrique pour chercher ses interactions avec les autres atomes. Les programmes d'**analyse hiérarchique** effectuent au contraire des comparaisons sur la collection des éléments primaires ou atomiques et des éléments construits (sous-cellules) d'une cellule et évitent ainsi les comparaisons entre éléments de sous-cellules trop éloignées; la structure hiérarchique évite à elle seule de multiples comparaisons.

4. PROBLEMES SPECIFIQUES

Tous les outils d'analyse de la description des masques d'un circuit posent le problème de **la recherche des interactions géométriques**. Etant donné un motif ou atome de géométrie, quels sont les autres atomes assez proches de lui pour interagir avec lui ? Ce problème de base est analysé au *chapitre 3. Le chapitre 4* propose une **méthode de recherche géométrique** efficace pour les descriptions de layouts avec ou sans hiérarchie.

Ces outils d'analyse posent aussi le problème de **l'équivalence électrique** de deux pièces de géométrie. Etant donnés deux morceaux / atomes de géométrie, sont-ils électriquement équivalents ? L'équivalence électrique est une relation d'équivalence dans l'ensemble des atomes de géométrie qui composent une conception. Le *chapitre 3* étudie divers algorithmes d'équivalences et leur application à l'analyse des layouts structurés de VLSI. *Le chapitre 5* propose au problème d'équivalence une solution hiérarchique.

Trouver une méthode générale qui permette d'analyser par des algorithmes hiérarchiques les layouts structurés des conceptions VLSI est le problème le plus difficile abordé par cette recherche. Les tâches de vérification, d'extraction de circuit et d'extraction des paramètres électriques à partir d'une description géométrique de la conception, sont distinctes mais très semblables [MCGR & WHI 80], [LOS 79]. Nous recherchons une méthode générale applicable à l'ensemble de ces tâches et aussi probablement à bien d'autres qui traitent la géométrie des conceptions et seront aussi discutées.

Une hypothèse qui a un impact significatif dans notre approche est que nous n'avons aucun contrôle sur le système de conception utilisé pour créer la description. Cela signifie qu'**il peut y avoir sans restriction des recouvrements dans le placement des symboles et des simples pièces / atomes de géométrie**. Certaines restrictions ont cependant été apportées afin de simplifier la "relation d'équivalence électrique" utilisée sur une description hiérarchique. Ces restrictions supposent implicitement une **certaine fonctionnalité(3) des symboles;** elles pourraient être levées moyennant un "surcroît de calcul" (difficile à évaluer : il peut être très important) qui complique l'étude hiérarchique des équivalences électriques. L'algorithme de vérification des règles de dessin lui-même est indépendant de toute fonctionnalité des symboles et vérifie chaque atome de géométrie avec son "voisinage" dans la profondeur de la hiérarchie.

Les descriptions hiérarchiques des masques fournissent pour ces nouveaux outils d'analyse hiérarchique, un même dénominateur commun, exactement comme le faisaient les descriptions complètement expansées pour les outils conventionnels. Les symboles sont construits de manière incrémentale avec des éléments primaires ou atomes de géométrie (rectangles, polygones, contacts, ...) et des instances de modèles ou symboles (sous-cellules et transistors). Une instance de symbole peut être réfléchie par rapport à chacun de ses axes, subir une translation ou tourner arbitrairement quand le symbole est placé. Les symboles sont éventuellement paramétrés par des changements d'échelle lorsqu'ils sont décrits avec des "unités" différentes ("pas" de grille, grille virtuelle). Ce facteur d'échelle est virtuel puisqu'il n'apparaît pas physiquement (masques réels au micron / surface du silicium).

La tâche d'analyse hiérarchique est étudiée au *chapitre 2* et une solution générale d'analyse hiérarchique est proposée. L'algorithme général d'analyse du layout d'un circuit est présenté au *chapitre 6.* Cet algorithme général utilise les sous-systèmes de recherche géométrique et d'équivalence adaptés aux descriptions hiérarchiques qui sont proposés aux *chapitres 4 et 5.* Les performances de cet algorithme doivent varier selon les caractéristiques de la description en entrée.

---

(3) règles d'interconnexion (implicites ou explicites) de ces symboles

La **qualité** d'une description hiérarchique intervient ainsi dans l'**efficacité de l'analyse** hiérarchique et différentes descriptions hiérarchiques peuvent produire pour un même algorithme des temps d'analyse fort différents. La **régularité** de la description structurée et le **non recouvrement des sous-cellules** produisent en général aux algorithmes hiérarchiques de bonnes performances *(chapitre 2)*. Les moins bonnes descriptions d'un circuit sont généralement pénalisées par un coût d'analyse supplémentaire. Le *chapitre 2* étudie les divers modèles hiérarchiques et le *chapitre 7* résume les principaux critères de qualité d'une description structurée. L'analyse d'une grande variété de conceptions hiérarchiques, constitue un développement de ce travail qui doit permettre l'optimisation des performances des algorithmes d'analyse hiérarchique développés.

Chapitre 2

LA DESCRIPTION ET L'ANALYSE HIERARCHIQUE DU LAYOUT

I. INTRODUCTION

Pendant de nombreuses années les descriptions hiérarchiques ont été utilisées seulement pour spécifier et réduire le travail de conception. Ainsi des descriptions structurées étaient faites pour les diverses représentations d'un circuit, mais les programmes d'analyse et de vérification n'utilisaient pas l'avantage de cette structuration des données. En 1980 un format hiérarchique d'échange en ***langage C.I.F.*** ou *Caltetch Intermediate Format* est spécifié [MEA 80]. Depuis cette date un certain nombre de stratégies d'analyse hiérarchique ont été proposées. Certaines de stratégies sont générales et permettent l'analyse de n'importe quelle hiérarchie, d'autres sont spécifiques à un type de hiérarchie, ce qui induit alors une restriction de la liberté des concepteurs.

Les méthodes d'analyse hiérarchique utilisent globalement moins d'espace mémoire, nécessitent moins de temps et permettent un meilleur report des erreurs que les méthodes d'analyse conventionnelle. Mais elles nécessitent souvent l'utilisation d'une hiérarchie restrictive puisque que [BEN 81 & OTT] ont montré qu'avec une hiérarchie non restrictive la plupart des problèmes géométriques à résoudre sur une collection de rectangles deviennent **NP-complets**, mesurés par rapport à la longueur de la description hiérarchique. Fort heureusement les descriptions hiérarchiques de layouts faites par les concepteurs ont plus de chance de générer de **bonnes hiérarchies** , puisque les concepteurs n'empilent généralement pas les symboles ou géométries de manière incohérente et illimitée mais au contraire avec une certaine régularité. Cependant, des **restrictions** explicites de la description du layout permettent d'améliorer les performances des analyses qui sont ensuite effectuées.

Il semble naturel que les outils qui analysent le layout d'un circuit exploitent la hiérarchie des données afin d'éviter tout travail redondant. Un concepteur peut créer une cellule une fois et l'utiliser un grand nombre de fois dans la description du layout. Ainsi les outils d'analyse hiérarchique doivent pouvoir reconnaître la régularité de la description et traiter ainsi une fois seulement une description de cellule, indépendamment des divers contextes qui l'utilisent; ils appliquent ensuite les résultats de cette analyse aux diverses occurrences de cette cellule. Cependant les occurrences de ces cellules peuvent avoir entre elles des interactions non négligeables. Diverses méthodes de description hiérarchique spécifient clairement la manière dont les définitions de cellules doivent être utilisées. Elles réduisent ainsi la complexité d'analyse des interactions de leurs occurences et la complexite globale de l'analyse du layout produit.

2. LA SPECIFICATION DU LAYOUT

La spécification du type de données à analyser constitue une première phase nécessaire à la spécification de l'analyse des layouts de circuits VLSI.

2.1. Les définitions du problème

Un **circuit** est une collection de **dispositifs électriques** primaires - ou composants - et d'**interconnexions**. Un **layout** est une collection de **figures à 2 dimensions** qui appartiennent aux différents niveaux de masques. Les outils de

spécification du layout aident le concepteur à spécifier cette collection de figures géométriques. L'analyse du layout permet de vérifier que les figures géométriques ont les propriétés désirées. Dans un layout , la **topologie** définie par la position relative des composants du circuit détermine la **fonction** du circuit; le **placement** et la taille des dispositifs électriques ainsi représentés détermine les **performances** du circuit.

2.2. Les descriptions hiérarchiques

Dans une conception structurée, on définit des **cellules,** ou **symboles** dont on utilise ensuite des **occurrences** pour construire, de manière ascendante, de nouvelles cellules. Les atomes ou **éléments primaires** qui servent à construire un layout structuré sont des *figures* ou parties de *figures.* Ainsi une cellule est généralement une combinaison d'**éléments primaires ou atomiques**, non décomposables, et d'**éléments structurés** ou **références** à d'autre cellules. Lorsque l'on définit une cellule, les occurences de définitions de cellules utilisées sont ses **sous-cellules** ou **instances de cellules**. Les éléments qui composent une cellule sont dits être d'un **niveau plus bas** dans la hiérarchie de conception que la cellule elle-même *(fig. 1).*

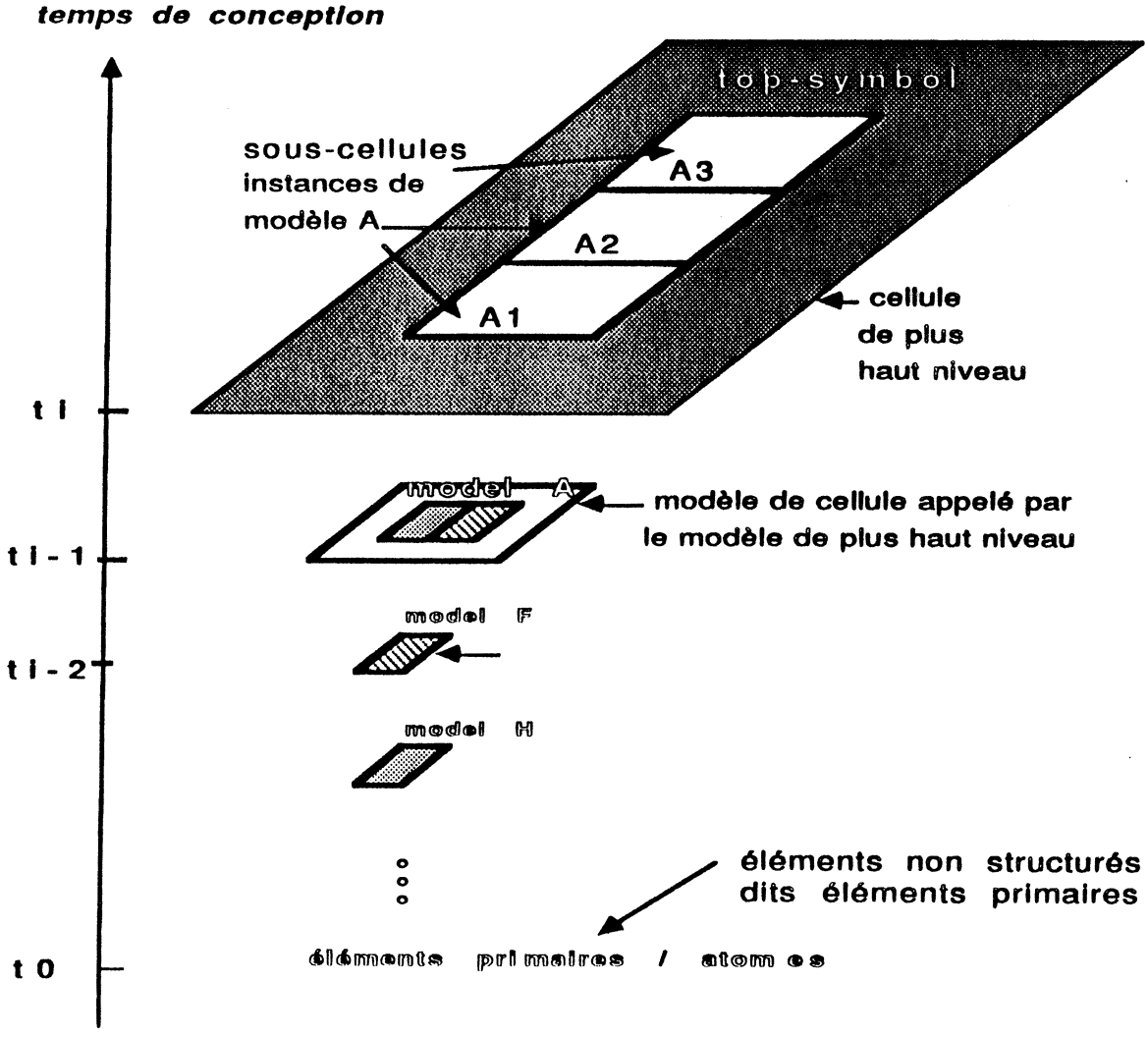


Figure 1 - *Exemple de* ***construction hiérarchique ascendante*** *-*

On peut de manière récursive, remplacer toutes les références d'une cellule hiérarchique par le contenu de leur définition jusqu'à obtenir une description constituée uniquement d'*éléments primaires*.. Ceci est une **mise à plat - ou expansion - de la hiérarchie**. Un **outil conventionnel d'analyse** du layout effectue préalablement une mise à plat de la hiérarchie de ce layout; il applique ensuite des algorithmes qui effectuent le traitement des *éléments primaires* ou *figures* du layout non hiérarchique ainsi obtenu.

Les données d'un layout sont partagées entre les cellules de diverses façons. Dans une hiérarchie **non restrictive** toutes les données sont accessibles par n'importe quelle cellule de n'importe quel niveau de la hiérarchie. Dans une **hiérarchie stricte** les données ne peuvent être partagées qu'entre des cellules de **niveaux adjacents**.

2.3. La complexité des analyses hiérarchiques

Supposons que les outils non hiérarchiques travaillent à partir d'une **liste de rectangles** qui spécifient les dessins des masques. Bentley et Ottman montrent que les problèmes fondamentaux de n'importe quelle **analyse conventionnelle** des layouts de VLSI - report de paires d'intersections de rectangles, etc. - sont en O(N*logN), et présentent un algorithme qui réalise cet optimum [BEN 81 & OTT]. Des algorithmes efficaces sont aussi présentés dans [BEN 80 & HAK], [BEN 80 & WOO]. Des modifications de ces algorithmes furent implémentées par U. Lauther [LAU 81]. Un temps de vérification en **O (N log N)** représente une limite inférieure de ce qui peut être fait dans l'optimisation des méthodes d'analyse non hiérarchiques, qui commencent par réaliser une mise à plat de la hiérarchie des layouts. Cette borne inférieure résulte en effet du tri de N rectangles [KNU 73].

En utilisant des **hiérarchies de rectangles**, Bentley et Ottman [BEN 81 & OTT] montrent que la plupart des problèmes sont **NP-complets**, par rapport à la **longueur N de la description hiérarchique**, lorsque ces hiérarchies sont conçues sans aucune contrainte.

Les approches de réduction de la complexité du problème général sont donc généralement restrictives; du choix de ces restrictions dépend l'efficacité des outils conçus.

2.4. Critères pour une solution du problème de l'analyse du layout

Un outil de vérification doit être jugé par l'aide qu'il apporte au concepteur dans la détermination et la localisation des erreurs.

Pour être efficaces les tests réalisés par un outil de vérification doivent être *complets* et ne pas signaler de *fausses erreurs*. Un test est *complet* s'il détecte effectivement toutes les erreurs à détecter. De plus les *diagnostics d'erreurs* doivent être faciles à analyser. Une erreur doit être détectée et signalée dès que possible. Elle doit être rapidement localisable. Enfin lorsque la description hiérarchique utilise une cellule, ou un groupe de cellules, de manière répétitive, chaque erreur ne devrait être signalée qu' une seule fois afin de faciliter le contrôle du concepteur.

Le **coût de vérification**, en terme de **ressources de calcul**, est aussi un important critère pour la spécification d'un outil de vérification. En effet, un outil de vérification rapide et pratique peut être utilisé fréquemment tout au long du processus de conception.

Un outil de vérification idéal devrait vérifier une conception par rapport à certaines *règles externes*, sans imposer d'autres contraintes au concepteur. Cependant dans le but de réduire la complexité des analyses hiérarchiques et d'obtenir des outils d'analyse efficaces, de nombreux systèmes de conception imposent des contraintes aux concepteurs pour la description des layouts structurés. Un critère de classement des systèmes de vérification peut être basé sur les **contraintes** qu'il impose au concepteur.

2.5. les approches de réduction du problème

Les vérificateurs hiérarchiques vérifient généralement séparément chaque modèle de cellule et testent ensuite les interactions entre les divers instances de ces cellules. Les diverses approches de ces vérificateurs (D.R.C.) diffèrent par la manière dont ils traitent les **juxtapositions** et les **recouvrements** de sous-cellules.

Une restriction courante est l'imposition du **non recouvrement des symboles** *(restriction géométrique)* [SCH 81], [WAG 84], [MCGR 80 & WHI]. Comme cela est dit dans [BEN 81 & OTT], cette restriction transforme beaucoup de problèmes NP-complets d'une analyse hiérarchique en problèmes qui ne le sont plus. La forme des symboles peut être polygonale tant que ces symboles ne se recouvrent pas. Les tâches d'analyse des descriptions géométriques peuvent ainsi être réalisées de manière hiérarchique et efficace.

Dans [JOH 82], le système (Bell Laboratories) restreint les concepteurs de manière légèrement différente. Dans ce système, le vérificateur examine seulement les interconnexions. Les transistors doivent être définis explicitement, plutôt que d'être formés par des recouvrements arbitraires de géométries. De plus, l'utilisateur doit donner une information de connectique dans le langage de spécification du circuit. Les géométries des transistors sont supposées être *prévérifiées* et ainsi le programme se concentre sur les interconnexions. Toutes les régions actives formées par le recouvrement d'interconnexions sont alors en erreur. Ces restrictions d'interconnexions des symboles *(restrictions topologiques)* permettent alors aux concepteurs des placements sans restriction *(géométrique)* de cellules et au vérificateur de bonnes performances (validation logique et DRC).

Les **approches restrictives**, *géométriques* ou *topologiques* - connectique, fonctionnalité des blocs - sont donc sans aucun doute bénéfiques aux algorithmes d'analyse hiérarchique, là où il est possible de contrôler le style de conception. Pour des raisons pratiques, ce n'est pas toujours possible *(§3.3.)*.

Mais le recouvrement des sous-cellules est souvent pratique, il n'est en effet pas possible d'isoler complètement des blocs fonctionnels sur la surface du silicium; ils ont besoin de recevoir les signaux d'entrée et sortie, de puissance et de masse; ainsi des cellules recouvrants ces blocs sont souvent utilisés pour assurer les transferts de signaux.

A l'inverse des approches restrictives, des **approches générales** essaient cependant aussi de réduire la complexité du problème général et de traiter n'importe quelle conception sans aucune hypothèse sur le mode de conception du circuit *(§3.4. §3.2.)*.

3. PANORAMA SUR L'UTILISATION DES HIERARCHIES

On peut se demander comment les hiérarchies ont été utilisées pour réduire la conception et l'analyse des layouts. Pendant de nombreuses années la hiérarchie était utilisée seulement pour les spécifications et pour la descriptions des diverses représentations d'un circuit, mais les programmes de vérification n'utilisaient pas l'avantage de cette structuration des données. En 1980, Mead et CONWAY spécifièrent alors un format hiérarchique d'échange en ***langage C.I.F.*** ou *Caltetch Intermediate Format* [MEA 80]. Cela provoqua le développement d'un certain nombre d'outils d'analyse hiérarchique.

3.1. L'analyse conventionnelle

Les premiers outils de vérification n'essayèrent pas de restreindre le type de hiérarchie utilisée par le concepteur. Le type de hiérarchie utilisé avait peu d'importance puisque la conception était complètement mise à plat avant d'être analysée. Les premiers vérificateurs ou D.R.C. (Design Rule Checkers) utilisaient cette philosophie. Les algorithmes de **scan line** étaient utilisés par les premiers D.R.C. commerciaux . [BAI 77] montra qu'avec les distributions particulières des données que produisent les layouts de VLSI, ces algorithmes de *scan line* réalisent la vérification d'un layout de **N rectangles** en un temps de l'ordre de O ($N^{3/2}$). L'amélioration de ces algorithmes permit d'observer des temps en O ($N^{1.2}$) [ALE78],[WIL 78], [YOS 77]. De plus récents travaux montrent qu'il est possible d'obtenir un temps optimal en **O(N logN)** avec cette approche [BEN 81 & OTT].

Les layouts ainsi *mis à plat* ont l'avantage de pouvoir être analysés par des algorithmes relativement simples. Aucune hypothèse sur le style de hiérarchie utilisée dans la conception n'est nécessaire. Ainsi les concepteurs peuvent réaliser des recouvrements sans restrictions; par exemple ils peuvent faire traverser les sous-cellules à des bus et des interconnexions.

Cependant la mise à plat des layouts structurés présente aussi des inconvénients. La quantité totale de données à traiter est très importante, ce qui exige une mémoire secondaire importante, et rend l'analyse du layout très lente. Mais un autre problème est celui du **report des erreurs**, puisque peu d'erreurs peuvent alors produire un grand nombre de reports, si les erreurs appartiennent à des cellules répétées un grand nombre de fois.

3.2. Le filtrage hiérarchique

A cause des problèmes posés par les layouts ainsi mise à plat, les concepteurs ont parfois vérifié des circuits dont ils avaient enlevé les parties les plus répétitives [RIV 80]. Ils ont ainsi fait une première analyse hiérarchique et ont produit un filtrage des données prenant en compte la régularité d'une description du layout.

T. Whitney prend avantage de la description hiérarchique pour réaliser un filtrage des instances de cellules. Si la description ainsi filtrée est sans erreur, alors la description originale l'est aussi. La description filtrée contient donc l'expansion de la première instance de chaque cellule - ou *symbole* - la cellule étant supposé vérifiée (marquage) lors d'un deuxième appel; lorsque deux instances de cellules sont en interaction (proximité, recouvrement), une *table des symboles* précédemment rencontrés est examinée pour savoir si une telle interaction relative a déjà été rencontrée; dans le cas contraire, l'aire d'interaction est examinée, c'est-à-dire que ses éléments sont réellement filtrés. Chaque cellule est vérifiée séparément, préalablement à la vérification de toutes les paires d'interactions de cellules. Les positions relatives de chaque paire de cellules sont alors sauvegardées. Cette approche a permis un gain de temps important pour des circuits très réguliers et utilisant un faible recouvrement de sous-cellules; mais l'analyse de layouts qui utilisaient extensivement les recouvrements de sous-cellules prit plus de temps que l'analyse du layout complétement mis à plat. De plus le filtrage des donnée qui est réalisé par l'analyse des paires d'interactions de sous-cellules ne garantit pas la détection de toutes les erreurs d'espacement *(figure 2).*

Mais T. Whitney considère elle-même que **l'auto-suffisance** des cellules est nécessaire pour réaliser un véritable programme d'analyse (ERC / DRC) [MCGR & WHI 80]. Par ailleurs, les filtres hiérarchiques produisent, à cause d'une information incomplète de connectivité, de fausses erreurs de connectivité et d'espacement (DRC). L'information filtrée est en général redondante et un traitement doit être fait pour éliminer les duplications de géométrie. Ces filtres produisent une image qui caractérise bien la qualité de la description hiérarchique; les mauvaises descriptions hiérarchiques donneront en effet un circuit résultant peu filtré et presque inchangé; les bonnes hiérarchies donneront un circuit résultant très "creux".

Le **filtrage hiérarchique** en général est une méthode inefficace pour les extractions de circuits, qui doivent recevoir en entrée soit le circuit complet, soit le circuit filtré augmenté de l'information de connectivité des parties ôtées du circuit complet. Un plus sérieux problème de cette méthode est la non détection de certaines erreurs comme l'explique le "contre exemple" de *la figure 2* [WHI 81], due au filtrage fait sur les paires de cellules déjà vérifiées dans la même *interaction relative* (et dûe à la vérification des interactions deux à deux en dehors du contexte complet d'interaction).

[LEU 86] propose une méthode de filtrage et d'analyse d'un layout hiérarchique qui ne présente pas les inconvénients des autres filtres hiérarchiques. Un algorithme hiérarchique filtre récursivement un layout quelconque et produit une collection de figures auxquelles sont associées un ensemble d'informations hiérarchiques, ce qui permet une meilleure analyse de leur *contexte d'utilisation hiérarchique.* Dans ce cas, l'information hiérarchique associée à chaque figure est suffisante pour produire une analyse correcte du layout de la cellule ainsi filtrée. Le traitement du layout ainsi produit est alors réalisé par un algorithme de *scan line* [FOK 83] capable d'analyser correctement ce layout filtré, mais en un sens hiérarchique. L'efficacité optimale d'un algorithme de *scan line* ainsi étendue à l'analyse hiérarchique, produit alors un outil d'analyse spécifiquement hiérarchique, et un temps global d'analyse proportionnel au nombre d'éléments du circuit complet.

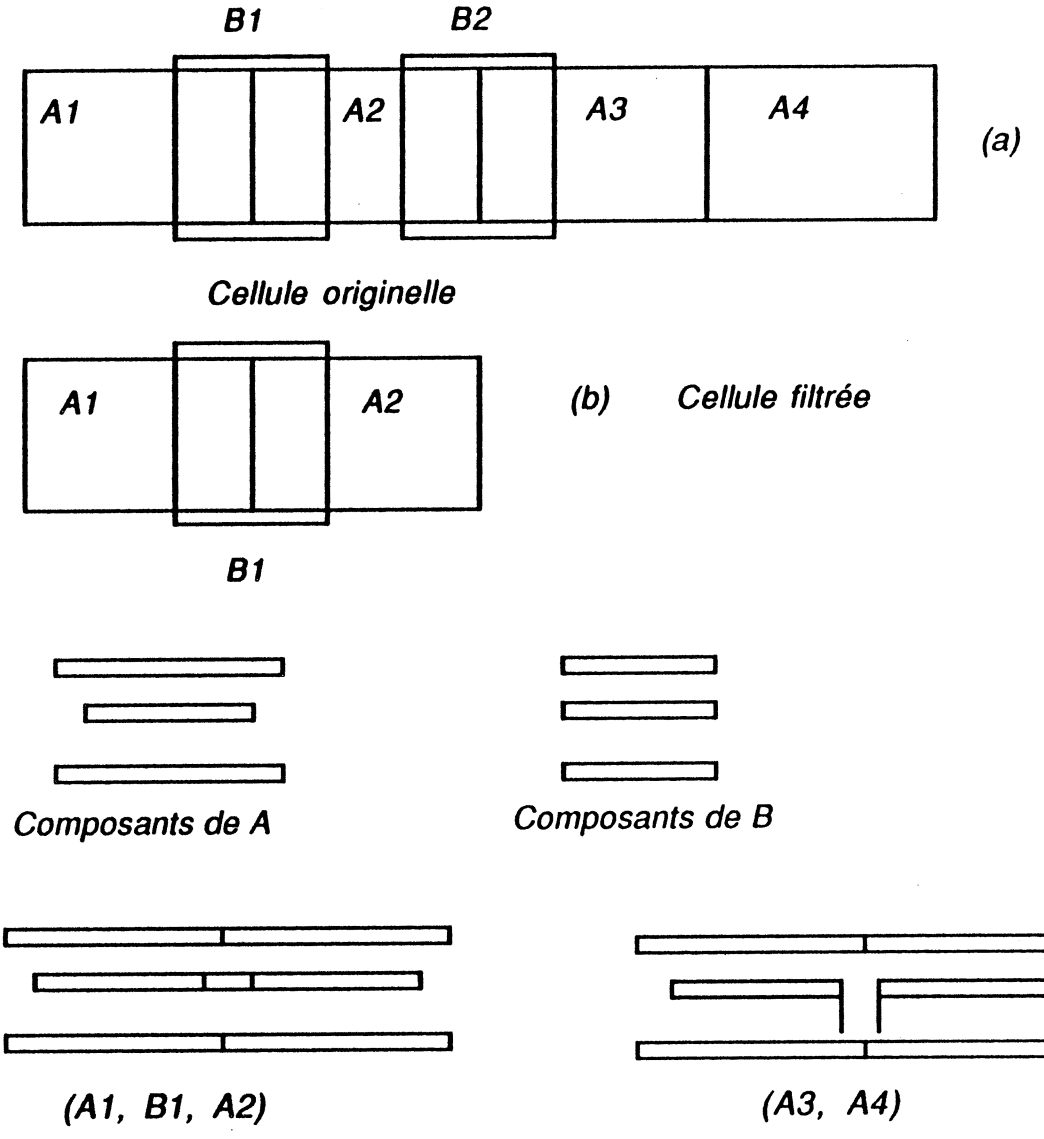


Figure 2 - **Filtrage hiérarchique** [WHI 81]- *Une sérieuse imperfection du filtrage hiérarchique pour le DRC. Si les cellules A et B contiennent la géométrie montrée en (c) alors la juxtaposition de A3 et A4 crée une violation d'espacement non reconnue par le DRC puisque le filtrage produit la figure (b);*

hypothèses : (A1, A2, A3, A4) instances de A; (B1, B2) instances de B;

méthode du filtrage hiérarchique : A1 est rencontré en premier par l'algorithme de filtrage et recopié en sortie du filtre, de même pour B1. Les interactions (A1, B1), (B1,B2), (A1, A2) sont aussi recopiées en sortie. les autre cellules sont déjà vérifiés , un post-processeur supprime les duplications.

3.3. L'analyse hiérarchique de conceptions restrictives

Le concept de non recouvrement des sous-cellules est utilisé par [WAG 84]. Le recouvrement des sous-cellules par des interconnexions est ainsi automatiquement transformé par la création de nouvelles cellules sans recouvrement *(checking cells, fig. 3).*

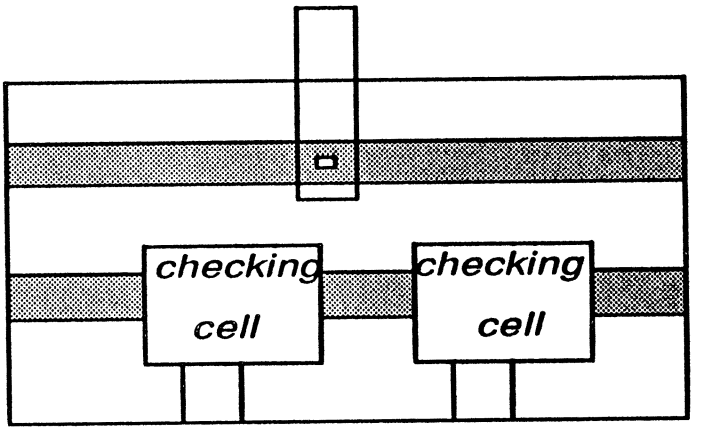


Figure 3 - ***Checking cells - Les recouvrements de sous cellules n'étant pas autorisés*** *lorsque des interconnexions passent au travers de sous cellules la portion d'interconnexion à l'intérieur doit faire partie de la cellule. Lorsqu'une cellule a été extraite elle est réduite à une frontière avec ses pins d'interconnexions pour utilisation dans un contexte [WAG 84].*

De manière incrémentale, des **halos** sont alors extraits à l'intérieur des bords de chaque cellule créée [WAG 84] [WON 85] [SCH 85]*(figure 4).* Le *halo* représente la cellule dans le contexte qui l'utilise, *halo* large de deux fois la règle maximale pour le DRC hiérarchique [WAG 84]. Le *halo* constitue un **résumé** de la cellule pour les contextes qui vont utiliser cette cellule. Sa construction dépend de l'analyse effectuée et des ressources nécessaires à une utilisation extensive de l'outil conventionnel d'analyse. Sa construction dépend aussi du type de méthodologie utilisée pour structurer la conception. Ainsi pour un layout structuré sans restrictions des recouvrements de sous-cellules, le *halo* de la cellule serait aussi large que la cellule, et, avec une telle méthode, l'analyse hiérarchique deviendrait alors inefficace.

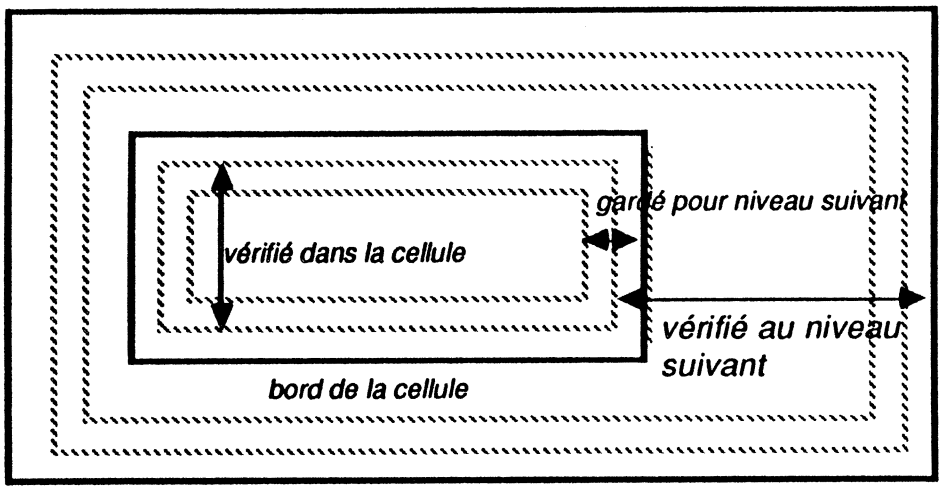
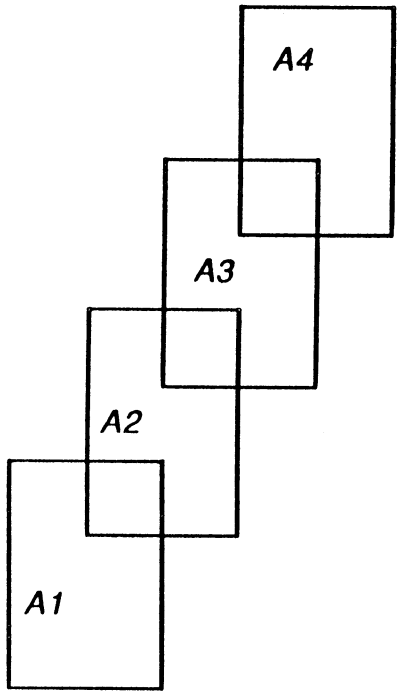


Figure 4 - ***Une méthode de halos utilisée pour la vérification des règles de dessin*** - *Extraction d'un* ***halo*** *représentant la cellule dans le contexte qui l'utilise, halo large de deux fois la règle maximale pour le DRC hiérarchique [WAG 84]. Le* ***halo*** *constitue un* ***résumé*** *de la cellule pour les contextes qui vont utiliser cette cellule. Sa construction dépend de* ***l'analyse effectuée*** *et des ressources nécessaires à une utilisation extensive de l'outil conventionnel d'analyse. Sa construction dépend aussi du type de méthodologie utilisée pour structurer la conception. Ainsi pour un layout structuré sans restrictions des recouvrements de sous-cellules, le* ***halo*** *de la cellule est aussi large que la cellule, et l'analyse hiérarchique devient alors inefficace.*

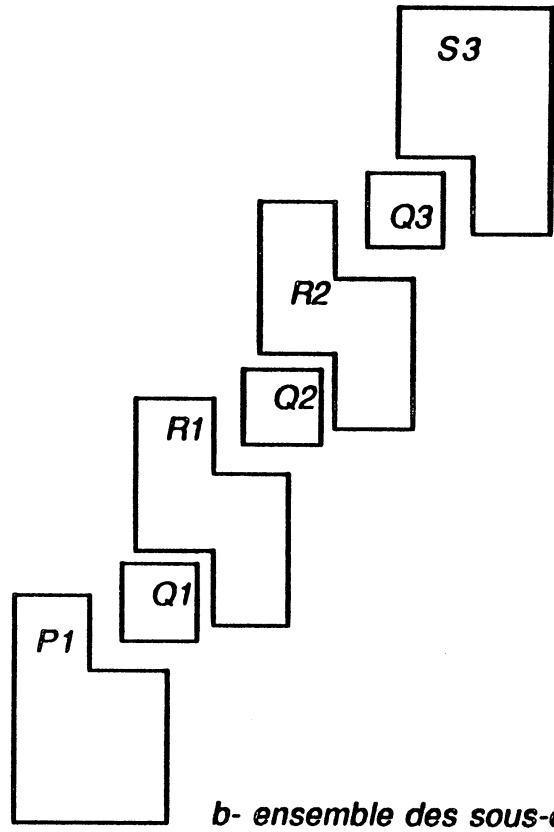
3.4. L'analyse hiérarchique par des méthodes générales

Dans [CHA 80], un programme d'extraction de circuit commence en traitant d'abord les **cellules terminales**, qui contiennent seulement des motifs de masques, et remonte vers la racine, en essayant de trouver les cellules recouvertes par aucune autre. Une fois qu'une telle cellule est trouvée, son contenu est extrait et cette cellule peut à son tour être représentée comme une **cellule terminale.** Cette approche de recherche, dans la hiérarchie, du *plus proche ancêtre commun* qui contienne complètement les cellules qui se chevauchent, peut cependant conduire à de pires cas lorsque cet ancêtre est la racine du circuit et qu'il faut alors réaliser l'expansion de toute la description. Avec cette méthode, les concepteurs qui limitent les recouvrements, obtiennent de meilleures performances de vérification que les autres, qui paient ainsi un coût supplémentaire de contrôle.

Newell et Fitzpatrick [NEW 82] utilisent une technique qui prend une hiérarchie quelconque, et génère une nouvelle hiérarchie dans laquelle les sous-cellules n'ont aucun recouvrement, appelée alors une **hiérarchie disjointe**. Cette hiérarchie est alors vérifiée en examinant séparément chaque cellule. Ici encore, cette technique permet de réaliser des gains importants pour des circuits qui ont peu de recouvrements, mais elle devient très coûteuse pour des circuits qui utilisent extensivement les recouvrements de sous-cellules.



a- tableau originel d'instances du symbole A.



b- ensemble des sous-cellules créées

Figure 5 - ***La transformation disjointe*** *- (A1, A2, A3, A4) instances de A forment un tableau (a) qui est transformé en (b) par la recherche des intersections de ces instances avec la création de nouvelles cellules (Q1, Q2, Q3, Q4) de manière à former un ensemble disjoint d'instances de cellules. La transformation disjointe s'applique récursivement à toute conception jusqu'à ce qu'il n'y ait plus aucun chevauchement d'instances de cellules; les nouvelles cellules sont alors juxtaposés sans recouvrement ni superposition.*

Une autre méthode générale transforme n'importe quel circuit en une hiérarchie sans recouvrements de cellules; elle est proposée par [HON 83]. Le circuit est ainsi **fracturé** en **tuiles disjointes** par un algorithme qui analyse la géométrie des circuits (de haut en bas) indépendamment de toute hypothèse sur les méthodes de conception utilisées (description CIF). La méthode s'est révélée coûteuse en temps de calcul pour certaines descriptions aux nombreux recouvrements.

4. LES STRATEGIES DE L'ANALYSE HIERARCHIQUE

4.1. Les diverses hiérarchies proposées

Les analyses hiérarchiques peuvent être énormément simplifiées si les layouts qu'elles analysent sont structurés selon certaines contraintes, qui restreignent ainsi la liberté du concepteur. Divers types de restrictions ont été utilisées dans ce but.

L'une des restrictions dans la structuration du layout est réalisée par l'utilisation de la **hiérarchie cohérente** (consistent hierarchy) proposée par [BEN 81 & OTT]. Dans ce type de hiérarchie les *rectangles enveloppes* des sous-cellules ne doivent avoir **aucun recouvrement**. T. Whitney et McGrath [McG 80] proposent aussi l'**auto-suffisance** des cellules; chaque cellule doit ainsi être légale par elle-même. Selon les mêmes principes, de moins strictes hiérarchies sont aussi proposées par [LOS 79].

Les **hiérarchies cohérentes** [BEN 81 & OTT] éliminent les recouvrements entre les *rectangles enveloppes* des sous-cellules. Cela signifie qu'il ne peut y avoir des connexions, des dispositifs primaires ou des sous-cellules, qui recouvrent au moins partiellement le rectangle minimum qui enveloppe une sous-cellule. Pour une telle hiérarchie beaucoup de problèmes élémentaires deviennent polynomiaux par rappport à la longueur de la description hiérarchique; la localisation d'un point par rapport à un ensemble de rectangles, devient ainsi de complexité O (N), N étant la longueur de la description hiérarchique. Certains problèmes, ainsi mesurés, restent cependant NP-complets, et ne sont polynomiaux que si on les mesure par rapport à la longueur de la description non hiérarchique équivalente. Les hiérarchies cohérentes de [BEN 81& OTT] sont trop restrictives pout être utilisées sur les cas réels que constituent les layouts de VLSI.

D'autres type de hiérarchies qui interdisent les recouvrements de sous-cellules sont ainsi proposées. [WAG 84] propose ainsi l'utilisation d'une hiérarchie restreinte dans laquelle les cellules peuvent avoir des formes polygonales mais ne peuvent être recouvertes par d'autres cellules, symboles ou interconnexions.

[TRIM 80 & ROW] proposent une **hiérarchie séparée** dans laquelle seulement les **cellules terminales** du niveau le plus bas - **cellules feuilles** - de la hiérarchie contiennent des motifs appartenant aux masques *(figure 6).* Les sous-cellules sont ainsi juxtaposées pour introduire des interconnexions. Les superpositions et juxtapositions de sous-cellules ne doivent pas produire de nouvelles fonctions. Cette approche conduit à des hiérarchies qui ont des propriétés intéressantes. Lorsque la technologie est changée, il suffit de modifier les *cellules feuilles* du plus bas niveau. Toute l'information sur les interconnexions est contenue dans la hiérarchie et la hiérarchie peut ainsi être transformée par des règles algébriques et formelles en une nouvelle hiérarchie plus simple, qui utilise les mêmes **cellules terminales**. Mais il n'existe pas de D.R.C. ou E.R.C. qui analyse ce style de hiérarchie, car il n'existe pratiquement pas de layouts ainsi structurés.

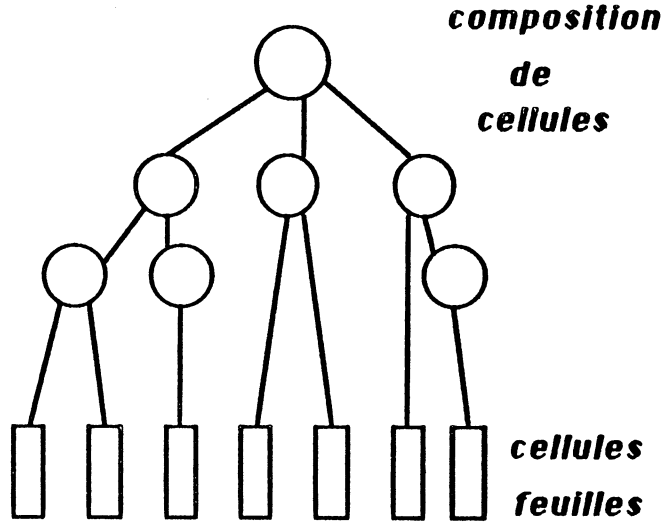


Figure 6 - ***Hiérarchie séparée*** *dans laquelle seules les* ***cellules terminales (cellules feuilles)*** *de la hiérarchie contiennent des motifs appartenant aux masques. Les sous-cellules sont ainsi juxtaposées pour introduire des interconnexions. Lorsque la technologie est changée, il suffit de modifier les cellules terminales (cellules feuilles).*

Dans [JOH 82], le système (Bell Laboratories) restreint les concepteurs de manière légèrement différente. Dans ce système, le vérificateur examine seulement les interconnexions. Les transistors doivent être définis explicitement, et ne peuvent être formés par des recouvrements arbitraires de géométries. De plus, l'utilisateur doit donner une information de connectique dans le langage de spécification du circuit. Les géométries des transistors sont supposées être *prévérifiées* et ainsi le programme se concentre sur les interconnexions. Toutes les régions actives formées par le recouvrement d'interconnexions sont alors en erreur. Ces restrictions d'interconnexion des cellules - **restrictions topologiques** - permettent alors aux concepteurs des placements sans restriction **géométrique de cellules** et assurent au vérificateur de bonnes performances.

4.2. Comparaison des stratégies de l'analyse hiérarchique

Les **restrictions géométriques** ou **topologiques** (connectique, fonctionnalité des blocs) sont donc sans aucun doute bénéfiques aux algorithmes d'analyse hiérarchique, là où il est possible de contrôler **le style de conception**. Pour des raisons pratiques, ce n'est pas toujours possible. A l'inverse des approches restrictives, des **approches générales** - *filtrage, expansion des interactions* - essaient cependant aussi de réduire la complexité du problème général et de traiter n'importe quelle conception sans aucune hypothèse sur le mode de conception du circuit.

Quatre stratégies de base permettent ainsi de réduire l'analyse hiérarchique des conceptions VLSI.

La première stratégie consiste à **ignorer la hiérarchie** et à **mettre à plat** la description à analyser [BAI 77].

La seconde stratégie utilise la hiérarchie comme un moyen d'accélérer l'analyse par un **filtrage des données**; ce filtrage élimine les **redondances** de vérification par la *reconnaissance de la régularité* des descriptions ainsi que par la *non réanalyse des cellules de la hiérarchie déjà vérifiées.* Ainsi l'analyse est souvent **incrémentale** et chaque nouvelle cellule est analysée *parallélement au processus de conception.* Certaines de ces méthodes permettent de considérer les cas de recouvrement de sous-cellules, mais les performances ne sont vraiment bonnes que lorsque les circuits sont bien structurés [WHI 81]. La **régularité** et un **faible recouvrement** des sous-cellules caractérisent alors les bonnes hiérarchies.

La troisième stratégie analyse des layouts **à priori** structurés selon des méthodes restrictives. La **hiérarchie restreinte** de ces layouts a alors une forme spécifique. L'analyse est alors beaucoup plus simple et plus complète, mais elle impose des contraintes au concepteur [BEN 81], [WAG 84], [WON 85], [SCH 85]. Les **restrictions** imposées sont de nature **géométrique** [BEN 81 & OTT] ou **topologogique** et **fonctionnelle** [JOH 82], ou purement **structurelles** [TRIM 80 & ROW].

La quatrième stratégie **transforme, à postériori, des layouts structurés** selon des méthodes non restrictives afin d'obtenir une **hiérarchie restreinte** qui en permette une analyse facile, [NEW 82], [HON 83]. La nouvelle hiérarchie ainsi obtenue est souvent très différente de la hiérarchie de départ, ce qui constitue un problème pour le report des erreurs détectées.

Toutes les stratégies d'analyse hiérarchique conviennent aux conceptions bien structurées. La **régularité** et un **faible recouvrement** des sous-cellules caractérisent les **bonnes hiérarchies** qui peuvent être ainsi analysées de manière efficace. En effet, les diverses stratégies d'analyse hiérarchique produisent des expansions partielles du layout, plus ou moins importantes, pour examiner les interactions entre les diverses occurences de cellules, et ces expansions sont d'autant plus importantes qu'il y a plus de recouvrements dans le layout analysé.

La plupart des problèmes qui doivent être résolus dans une analyse hiérarchique sont NP-complets si on les mesure par rapport à la longueur N de la description hiérarchique, c'est-à-dire pratiquement insolubles dans leur généralité. Cependant la plupart de ces problèmes sont polynomiaux si on les mesure par rapport à la longueur N de la description non hiérarchique équivalente. Les analyses hiérarchiques utilisent donc la **faisabilité de l'analyse conventionnelle sur de petites parties de layout**. Les **algorithmes conventionnels** d'analyse sont ainsi utilisés sur de plus ou moins importantes, voir de très **petites parties de layouts**. Pour un même layout, plus la stratégie d'analyse hiérarchique est efficace et plus petites sont les parties de layout réellement expansées. Pour une même stratégie, parmi toutes les descriptions hiérarchiques équivalentes la meilleure est celle qui produit une **expansion globale minimum** - en définissant cette expansion globale comme la somme de toutes les expansions partielles.

Des **expansions partielles,** plus ou moins importantes selon **l'efficacité des stratégies** et la **qualité des descriptions**, se substituent à une mise à plat ou expansion complète de la description hiérarchique pour en assurer une analyse plus rapide. La description hiérarchique est généralement utilisée pour réduire le **nombre d'interactions** de sous-cellules à **examiner**; certaines stratégies l'**analysent** alors de manière **ascendante et incrémentale**; d'autres l'analysent de manière **descendante et récursive.**

Des algorithmes utilisés pour l'**analyse conventionnelle** de layouts non structurés sont généralement utilisés pour analyser les expansions partielles de layouts structurés *(figure 6 bis)*. Cependant les parties de layouts ainsi filtrés ou expansées peuvent produire des comportements de pire cas à certains algorithmes qui travaillent bien sur un certain type de distribution des données [BEN 80 & HAK]; le comportement de ces algorithmes est ainsi à reconsidérer.

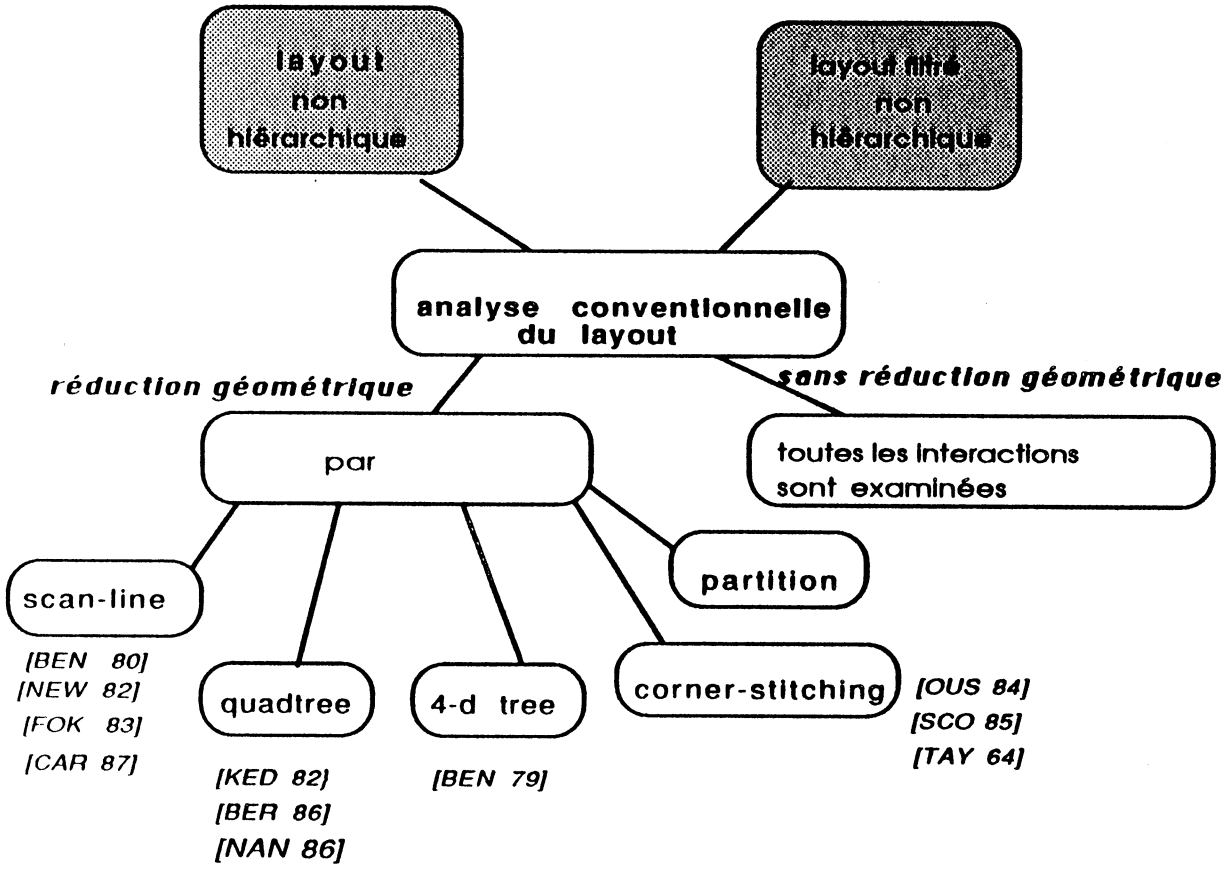


Figure 6 bis - ***Les divers algorithmes de l'analyse conventionnelle.***

Une **classification** comparative des diverses méthodes de réduction de la complexité des hiérarchies à analyser, est présentée sur la *planche 1.* Les diverses **restrictions hiérarchiques** de conception induisent **différentes hiérarchies** et **différents styles d'analyse** indiqués sur les branches de ce schéma arborescent. Certaines analyses hiérarchiques utilisent **a priori** des hiérarchies restreintes, tandis que d'autres analysent une hiérarchie quelconque, soit par expansion de la description, soit par **transformation de la hiérarchie de départ, à postériori,** en une hiérarchie restreinte permettant une analyse moins complexe.

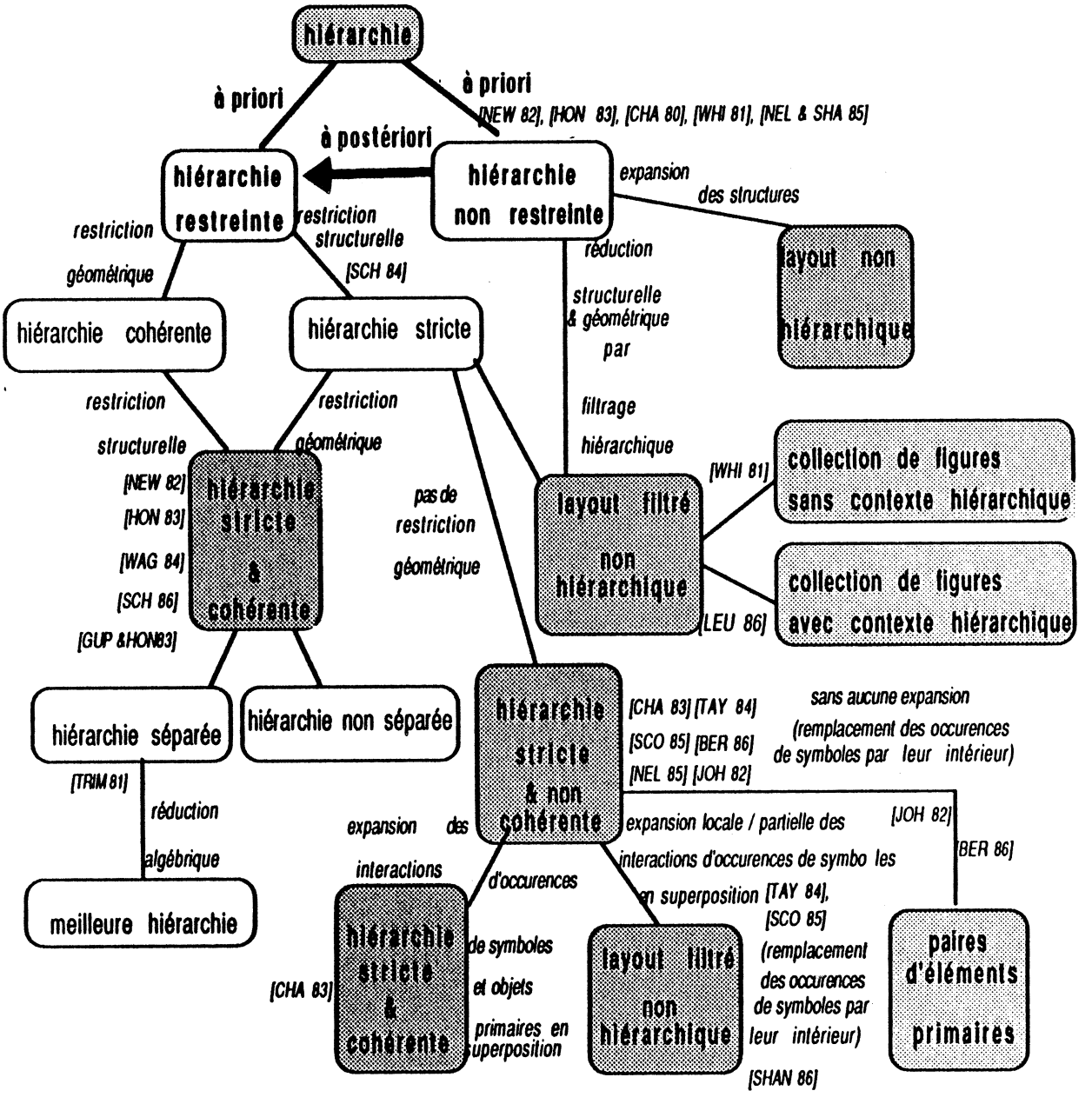


Planche 1 - **Les diverses restrictions hiérarchiques de conception induisent différentes hiérarchies et différents styles d'analyse** indiqués sur les branches de ce schéma arborescent - Certaines analyses hiérarchiques utilisent **à priori** des **hiérarchies restreintes**, tandis que d'autres analysent une **hiérarchie quelconque**, soit par **expansion** de la description, soit par **transformation à postériori** de la hiérarchie de départ en une **hiérarchie restreinte** permettant une analyse moins complexe.

Une **hiérarchie est** dite **stricte** lorsque les données ne peuvent être partagées qu'en deux niveaux adjacents. Cette restriction est une **restriction structurelle**. Elle peut cependant être réalisée de **manière géométrique**; ainsi la hiérarchie séparée **[TRIM 81]** réalise la connexion électrique des sous-cellules par juxtaposition (aboutement).

Une **hiérarchie** est dite **cohérente** - au sens large - lorsque les données d'une cellule ne peuvent pas être recouvertes par les données d'une autre cellule - de même niveau hiérarchique ou de niveau hiérarchique plus élevé. Cette restriction est une **restriction géométrique**.

Une **hiérarchie** sera dite **stricte** si les données, appartenant aux définitions de cellules, ne peuvent être partagées qu'en deux **niveaux adjacents**. Cette restriction est une **restriction structurelle**. Elle peut cependant être réalisée de manière géométrique; ainsi la **hiérarchie séparée** [TRIM 81] réalise la connexion électrique des sous-cellules par juxtaposition *(aboutement)*.

Une hiérarchie sera dite **cohérente** - *au sens large* - lorsque les données d'une cellule ne peuvent pas être recouvertes par les données d'une autre cellule - de même niveau hiérarchique ou de niveau hiérarchique plus élevé. Cette restriction est une **restriction géométrique**.

5. L'ANALYSE DU MODELE HIERARCHIQUE

On va examiner les diverses méthodes de l'analyse hiérarchique sans considérer les détails de l'analyse à réaliser. Trois aspects de **l'analyse** de conceptions structurées vont ainsi être examinés : la technique de mise à plat de la hiérarchie ou **technique conventionnelle**, une **technique hiérarchique** et une **technique incrémentale.** Les meilleures analyses conventionnelles sont en O (NlogN) mesurées par rapport au nombre d'éléments de la conception analysée.

5.1. L'analyse conventionnelle

Une cellule est une combinaison de sous-cellules, ou *éléments structurés*, et d'*éléments primaires* non décomposables. Une cellule peut être **mise à plat** ou **expansée** en remplaçant chaque sous-cellule par la définition de la cellule ainsi référencée. Un algorithme d'expansion travaille en remplaçant ainsi toutes les instances de cellules, jusqu'à ce qu'il n'y ait plus que des *éléments primaires.* Une analyse qui nécessite l'expansion préalable des hiérarchies sera appelée une **analyse conventionnelle**.

5.2. Extension d'un outil d'analyse conventionnel pour l'analyse hiérarchique

On suppose que l'on puisse construire pour chaque cellule un **résumé** et que ce **résumé** puisse **se comporter exactement comme la cellule qu'il résume** dans les divers contextes d'utilisation de cette cellule. Un niveau plus haut dans la hiérarchie de conception l'analyse d'une cellule est réalisée en utilisant les *résumés* des sous-cellules. Ce processus continue ainsi jusqu'à atteindre la cellule racine de la hiérarchie. Cette cellule qui englobe toute la hiérarchie est alors analysée par une **analyse conventionnelle** avec cependant le remplacement des sous-cellules par leurs *résumés.*

Ainsi une analyse hiérarchique qui utilise la méthode du remplacement des sous-cellules par leur **résumé** devra :

1. ***analyser les modèles des sous-cellules et en résumer éventuellement le résultat***
2. ***analyser la cellule***
3. ***créer éventuellement un résumé de la cellule ainsi analysée***

L'étape 2 ***analyser la cellule*** *est réalisée en 3 étapes :*

2.a. lire ou calculer les résumés des sous-cellules

2.b. expanser (instancier) les résumés des sous-cellules dans la cellule

2.c. analyser la cellule avec l'outil d'analyse conventionnelle

Lorsque l'on effectue l'**analyse hiérarchique** d'une cellule à l'aide d'un outil conventionnel, on **analyse** de manière récursive les définitions de ses **sous-cellules imbriquées** et **on en résume éventuellement le résultat** avant d'**analyser** la cellule par l'**outil d'analyse conventionnel**, *comme en figure 7.*

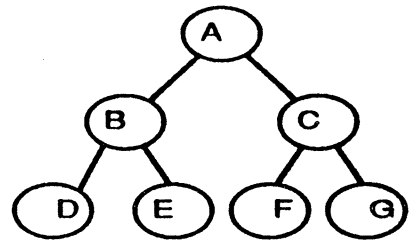


*Figure 7 - Soit la hiérarchie des instances de cellules A, B, C, D, E, F, G. Pour effectuer une analyse de A, il est nécessaire de créer les **résumes** de D, E, avant de créer un **résumé** de B, puis les **résumés** de F et G avant de créer un **résumé** de C. Avec les **résumés** de B et C, il est alors possible d'analyser A.*

5.3. Efficacité et limites dans l'extension d'un outil d'analyse conventionnel

On construit généralement *le résumé* d'une cellule, après l'avoir analysée, en supprimant tous éléments qui sont corrects et ne peuvent produire des erreurs dans les contextes qui les utilisent. Ainsi le résumé d'une cellule est **généralement plus simple** que la cellule elle-même. De plus **le résumé d'une cellule** doit **effectivement résumer les résultats de l'analyse** effectuée sur cette cellule et en être le porteur. Dans ce cas seulement le *résumé* de l'analyse peut remplacer complètement la cellule dans les contextes qui l'utilisent Ainsi le *résumé* d'une cellule dépend de la nature de l'analyse et aussi du type de hiérarchie utilisée [SCH 85].

Ainsi, sans restriction de la hiérarchie, le *résumé* d'une cellule **peut ne pas être plus petit** que la cellule elle-même. Supposons que nous ayons une conception dans laquelle les recouvrements de sous-cellules soient interdits. Alors le périmètre constitue la seule portion d'une cellule qui ait une interaction avec le contexte qui l'utilise. Alors le *résumé* de la cellule pour un D.R.C. hiérarchique est formé par le ***halo*** de largeur μ construit à l'intérieur de la frontière de la cellule; μ étant calculé en fonction de la valeur de la plus grande règle de dessin. Pour un D.R.C. hiérarchique qui opère sur des hiérarchies non restreintes, le *résumé* d'une cellule est peut-être alors la cellule entièrement expansée, et alors un D.R.C. hiérarchique qui résume les cellules analysées, ne produit pas de gain. Cette stratégie devra, dans ce cas , être légèrement modifiée car il sera alors certainement plus avantageux d'extraire les éléments de la cellule en les recherchant directement dans la profondeur de la hiérarchies des cellules que l'on a déjà analysées.

La sauvegarde des résultats de l'analyse de chaque cellule, *éventuellement résumée*, permet un gain de temps dans l'analyse hiérarchique. Si une cellule est modifiée, elle est alors réanalysée. Si le *résumé* de l'analyse d'une cellule est modifié, cela provoque une modification dans les diverses cellules qui l'utilisent; ces cellules utilisatrices doivent alors être réanalysées. La sauvegarde des *résumés* des cellules d'une conception n'est réaliste que si ces *résumés* sont plus petits que les cellules ells-mêmes. Il n'est en effet pas concevable de conserver les expansions complètes de toutes les cellules d'une conception structurée. Dans certains cas, il est préférable de reconstruire ces résumés de manière dynamique, au moment où les modèles de cellules interviennent et tout en propageant les résultats de leur analyse *(annexe 2.§5).*

Le report des résultats de l'analyse de chaque définition de cellule, la sauvegarde ou la construction dynamique de son résumé, assurent alors une analyse incrémentale de la description hiérarchique d'une conception. Le temps dépensé dans l'analyse complète de la conception est alors fragmenté et réparti tout au long du processus de conception. Lorsque la dernière cellule est conçue et analysée alors l'analyse est complète.

5.4. L'analyse hiérarchique et incrémentale

Dans une analyse incrémentale, chaque modèle de cellule utilisé dans la hiérarchie de conception est analysé parallélement au processus de conception. Le principal avantage de cette méthode est une détection précoce des erreurs; une cellule peut être ainsi analysée dès qu'elle est conçue et que ses descendants le sont aussi. Le principal désavantage de cette méthode est la détection possible de **fausses erreurs** qu'il est nécessaire de distinguer, en attendant une analyse plus complète de la hiérarchie de conception. Une cellule peut ne pas être correcte par elle-même et être correcte dans le contexte qui l'utilise. Une fois reconnues, ces fausses erreurs peuvent éventuellement être rajoutées aux *résumés* des cellules, afin de pouvoir être réanalysées un niveau plus haut dans la hiérarchie de conception.

5.5. L'exploitation des répétitions

Lorsqu'une cellule a été analysée et éventuellement *résumée*, le résultat ainsi obtenu évite la réanalyse de toutes les occurrences de cette cellule. Le remplacement des références à cette cellule par le résultat de son analyse, assure alors l'analyse hiérarchique des contextes qui l'utilisent. Un gain de temps est ainsi réalisé lorsque ces *résumés* sont plus petits que la cellule elle-même.

Pour une analyse telle que le D.R.C. hiérarchique, la reconnaissance des occurrences d'une même **interaction** - proche voisinage - de sous-cellules permet d'éviter de nombreuses vérifications.

La **reconnaissance de la régularité** des conceptions joue ainsi un rôle primordial dans l'amélioration des performances - en mémoire et en temps - des analyses hiérarchiques.

6. PERFORMANCES THEORIQUES DE L'ANALYSE INCREMENTALE

6.1. Estimation globale et faisabilité de l'analyse incrémentale

Soit un layout structuré selon une hiérarchie séparée dans laquelle seule les cellules terminales ont des éléments primaires. Soient les hypothèses d'une ***analyse conventionnelle*** réalisée avec un ***temps*** t d'ordre au plus $O(N^{\beta})$, et d'une ***analyse incrémentale*** portant sur un nombre d' ***éléments primaires*** n d'ordre au plus $O(N^{\alpha})$, pour l'utilisation d'une cellule contenant **N éléments primaires**. Les lettres grecques seront par défaut des réels positifs.

- il existe alors k_{α} réel et N_0 entier tel que pour $N > N_0$ on ait $n \leq k_{\alpha} * N^{\alpha}$
- il existe alors k_{β} réel et N_1 entier tel que pour $N > N_1$ on ait $t \leq k_{\beta} * N^{\beta}$

Il est possible, avec ces hypothèses, de donner une estimation théorique du **temps total** pris par cette ***analyse incrémentale***, pour analyser ***des hiérarchies sans recouvrement de sous-cellules.***

Supposons qu'un circuit de N *éléments primaires* soit organisé avec des **cellules composées de $\mathcal{P}$ éléments**. Les cellules de niveau 1 contiennent $\mathcal{P}$ ***éléments primaires,*** les cellules de niveau 2 contiennent $\mathcal{P}$ cellules de niveau 1, et ainsi de suite. Il y a alors $N / \mathcal{P}$ plus basses cellules et chacune peut être vérifiée indépendamment; ainsi cela prendra un temps de $K_1 * (N / \mathcal{P})$, K_1 étant le temps de vérification d'une cellule; ce temps est proportionnel à N. Au niveau suivant de la hiérarchie, il ya $N / \mathcal{P}^2$ cellules, et chacune d'elles peut être est vérifiée indépendamment; ainsi le temps total nécessaire pour vérifier ce niveau de la hiérarchie est de $K_2 * (N / \mathcal{P}^2)$, K_2 tant le temps de vérification d'une cellule de ce niveau. Ce temps est donc aussi proportionnel à N. Si chaque niveau supérieur prend un facteur γ **fois moins de temps que pour l'analyse du niveau précédent**, avec $\gamma < 1$, alors le temps total pour vérifier la hiérarchie complète est borné par :

$$T \leq K_1 * N * \mathcal{P}^{-1} + K_2 * N * \mathcal{P}^{-2} * [1 + \gamma + \gamma^2 + \gamma^3 + \ldots]$$

$$\leq [[K_1 * \mathcal{P}^{-1} + K_2 * \mathcal{P}^{-2} * [1-\gamma]^{-1}] * N$$

$$\leq \mu (\mathcal{P}, \gamma) * N$$

<u>Calcul de γ</u>- On va comparer le temps pris pour tester deux niveaux hiérarchiques adjacents. Les cellules de chacun de ces niveaux ont P sous-cellules. Cependant les sous-cellules du niveau le plus haut seront plus grosses d'un facteur $\mathcal{P}$. Ainsi *l'analyse incrémentale* sera faite sur *un nombre d'éléments primaires* $\mathcal{P}^{\alpha}$ plus grand, ce qui produira un temps $(\mathcal{P}^{\alpha})\beta$ plus long. Cependant il y a $\mathcal{P}$ fois moins de cellules sur le niveau le plus haut. Ainsi le le temps total demandé pour pour vérifier le plus haut niveau, comparé au temps demandé pour vérifier le plus bas niveau, est $\mathcal{P}^{\alpha\beta} / \mathcal{P}$, ou $\mathcal{P}^{\alpha\beta - 1}$. Ainsi $\gamma = \mathcal{P}^{\alpha\beta - 1}$; $\gamma < 1$ si $\alpha\beta < 1$

Par exemple pour un D.R.C. incrémental qui utilise une technique de halos pour extraire les éléments primaires à vérifier, en O ($N^{1/2}$), et un algorithme de scan-line pour effectuer la vérification, en O ($N^{3/2}$) pour N éléments, on a $\alpha = 1/2$ et $\beta = 3/2$. Supposons que chaque cellule contienne $\mathcal{P}=16$ sous-cellules.. Le temps total demandé pour vérifier chaque niveau de la hiérarchie est alors divisé par 2 à chaque niveau. Ainsi, le temps demandé pour vérifier une hiérarchie de n'importe quelle taille n'exédera pas 2 fois le temps le temps demandé pour vérifier le niveau 2 de la hiérarchie. De plus le temps demandé pour vérifier le niveau 2 est proportionnel à N. Ce qui assure alors, moyennant ces hypothèses, la vérification de toute la hiérarchie en temps en un temps meilleur que O (N), et ainsi la faisabilité de la vérification.

6.2. Estimation plus détaillée et intérêt de l'analyse incrémentale

Des hypothèses plus détaillées sur le type de hiérarchie analysée, vont permettre une estimation plus détaillée.

Soit **N** le nombre ***d'éléments primaires*** du circuit complet.
Soit $\mathcal{P}$ le nombre d'***éléments (primaires ou structurés)*** qui **définissent** une cellule.
Soit **Q** le nombre de fois qu'une ***cellule est utilisée*** *d'abord supposé indépendant de N - par exemple Q = 2.*

On fait les mêmes hypothèses qu'*en §6.1.* pour le **temps d'une analyse conventionnelle de N éléments**, et le **nombre d'*éléments primaires* à considérer** dans l'*analyse incrémentale* d'une sous-cellule de **N éléments.**

Le niveau terminal des cellules composées uniquement d'éléments primaires est le niveau 1.

Soit t_i **le temps de vérification d'une cellule de niveau** i de hiérarchie
Soit $\mathcal{N}_i$, le nombre total de cellules de niveau i, $\mathcal{N}_i = N / \mathcal{P}^i$
Soit $n_i = \mathcal{P}^i$ le nombre total *d'éléments primaires* ***inclus*** dans chaque cellule de **niveau** i

<u>Calculons le temps T_i nécessaire à l'analyse de toutes les cellules de niveau i de la hiérarchie.</u>

1. On a $t_1 \leq K_\beta \mathcal{P}^\beta$; d'où $T_1 \leq \mathcal{N}_1 t_1 \leq (N / \mathcal{P}Q) K_\beta \mathcal{P}^\beta$

2. Pour $i > 1$: chaque cellule contient $\mathcal{P}$ sous-cellules, chacune ayant $\mathcal{P}^{i-1}$ *éléments primaires* , à l'intérieur.

Ainsi le nombre **d'éléments primaires de niveau** i **à analyser** est :

$$c_i \leq \mathcal{P}\, k_\alpha (n_{i-1})^\alpha \text{ , soit } c_i \leq \mathcal{P}\, k_\alpha (\mathcal{P}^{i-1})^\alpha$$

d'où $t_i \leq k_\beta (c_i)^\beta$; soit $t_i \leq K_\beta (\mathcal{P}\ K_\alpha (\mathcal{P}^{i-1})^\alpha)^\beta$

$$T_i = (\mathcal{N}i / Q) * t_i \ ;$$

d'où $T_i \leq (N / Q) * K_\alpha{}^\beta * K_\beta * \mathcal{P}^{\beta(1-\alpha)} * \mathcal{P}^{i(\alpha\beta - 1)}$

En sommant pour tous les niveaux, de 1 à h, de la hiérarchie on obtient :

$$T_{total} = \Sigma_{i=1..h}\, T_i$$

$$T_{total} \leq (N / \mathcal{P}Q) * K_\beta \mathcal{P}^\beta + (N / Q) * K_\alpha^\beta K_\beta * \mathcal{P}^{\beta(1-\alpha)} \sum_{i:2..h} \mathcal{P}^{i(\alpha\beta-1)}$$

Ainsi **si** $\alpha\beta - 1 < 0$ **, le temps total est borné par :**

$$T_{total} \leq (N/ \mathcal{P} Q) * K_\beta \mathcal{P}^\beta + (N/ Q) * K_\alpha^\beta K_\beta * \mathcal{P}^{(\beta+\alpha\beta-2)/(1-\mathcal{P}^{\alpha\beta-1})}$$

soit : $T_{total} \leq f(\alpha, \beta, Q, \mathcal{P}) * N$

Le ratio T_{total} / N est borné par une **fonction** $f(\alpha, \beta, Q, P)$

1. ***décroissante par rapport à Q, régularité de la description***
2. ***Il existe une valeur de $\mathcal{P}$, nombre d'éléments définissant une cellule, qui la minimise***
3. ***pour $P = 1$ ou $P = \infty$, cette fonction n'est pas borné.***

Ainsi, pour le D.R.C. incrémental, si $\alpha = 1/2$ alors le temps total de vérification sera au plus en O (N) pourvu que le D.R.C. de base soit légèrement meilleur que $O(N^2)$.

Si on utilise les meilleurs algorithmes de base pour le D.R.C., en O (N log N).

Pour $\varepsilon > 0$ on a $O(N \log N) < O(N^{1+\varepsilon})$; ainsi le temps total T de vérification est en O (N) dès que $\alpha < 1 / (1+\varepsilon)$. Il suffit dans ce cas d'un léger gain par cellule analysée, par rapport à une analyse conventionnelle, pour que temps total de vérification soit en O (N).

Ainsi la plupart des algorithmes de base utilisés dans une analyse de cette hiérarchie théorique et **sans recouvrements**, conduisent ainsi à un temps total d'analyse en O (N).

<u>Calcul du temps total d'analyse par rapport à la longueur de la description hiérarchique</u>

Soit U_i le nombre de cellules dessinées au $i^{ème}$ niveau de la hiérarchie.

$U_i = N / Q / \mathcal{P}^i$ cellules doivent être dessinées, et chacune d'elles est définie par la donnée, ou le dessin, de $\mathcal{P}$ éléments. Ainsi la longueur totale de la description est

$$\mathcal{E} = \sum_{i=1..h} U_i * \mathcal{P} = \sum_{i=1..h} (1 + \mathcal{P}^{-1} + \mathcal{P}^{-2} + \ldots + \mathcal{P}^{-h+1}) * N / Q$$

$$= (1 - \mathcal{P}^{-h}) * N / Q (1 - 1/\mathcal{P})$$

$$= N * (1 - N^{-1}) / Q(1 - \mathcal{P}^{-1})$$

d'où $\mathcal{E} = \mathcal{L}(Q, \mathcal{P}) * N * (1 - N^{-1})$ avec $\mathcal{L}(Q, \mathcal{P}) = 1 / Q(1-\mathcal{P}^{-1})$

$$\Rightarrow \mathcal{E} < \mathcal{L}(Q, \mathcal{P}) * N$$

$\Rightarrow \mathcal{E} \approx \nu * N / Q$ pour N grand, ν réel

$\mathcal{E}$, longueur de la description hiérarchique, caractérise ainsi le temps de conception - dessin - de layouts ainsi structurés.

Ainsi le **ratio $T_{total} / \mathcal{E}$ est une fonction $\mathcal{L}$ qui dépend très peu de la taille de la conception**. Il dépend de la manière $\mathcal{L}$ (Q, $\mathcal{P}$) dont cette conception est structurée. Ainsi un gros circuit très répétitif et un petit circuit peuvent demander le même effort de conception et aussi des temps d'analyse comparables. Ceci n'est pas le cas pour les analyses conventionnelles, qui effectuent une expansion complète de la description, et une analyse de complexité moins bonne que O (N).

Ces **analyses incrémentales** des **layouts hiérarchiques** permettent ainsi une **analyse dynamique** de la conception, au fur et à mesure de l'entrée des données. Ainsi on peut réaliser des **vérifications interactives** et effectuer des analyses qui impliquent un nombre de données suffisamment restreint pour être effectuées en mémoire principale. La résidence des données en mémoire permet alors l'utilisation d'algorithmes "sophistiqués" de géométrie numérique ou d'équivalence *(cf. chap. 3, 4 & 5)*.

Nombre d'éléments primaires à analyser lors de l'analyse incrémentale d'une cellule

On peut calculer le plus grand nombre ***d'éléments primaires*** qui doivent ainsi être **traités** au cours d'un test incrémental :

1. au niveau i de la hiérarchie : $C_i \leq \mathcal{P} * K_\alpha (\mathcal{P}^{i-1})^\alpha$
2. au plus haut niveau de la hiérarchie :

$$\mathcal{P}^i = N \quad \text{d'où} \quad C_{max} \leq \mathcal{P} * K_\alpha (N / \mathcal{P})^\alpha$$

$$C_{max} \leq \mathcal{P}^{1-\alpha} * K_\alpha N^\alpha$$

Si ce nombre *d'éléments primaires* peut être analysé en mémoire principale, alors toutes les cellules peuvent être analysées avec l'hypothèse que toutes les données qui doivent être traitées, pour réaliser un test incrémental, peuvent être résidentes.

Cas où le facteur de répétitivité Q est une fonction de N :

En général le facteur de répétitivité Q, c'est-à-dire le nombre de fois qu'une cellule est utilisée, est une fonction croissante du nombre d'éléments du circuit - plutôt qu'une constante. La plupart des calculs précédents restent cependant corrects dans l'hypothèse où Q est une fonction de N.

Soit **$Q(N) = K_\eta * N^\eta$ le facteur de répétitivité**

Soit $\alpha\beta < 1 \Rightarrow T_{total} \leq f(\alpha, \beta, Q, \mathcal{P}) * N \leq \theta(\alpha, \beta, \mathcal{P}) * N/Q$

$\Rightarrow T_{total} \leq \theta(\alpha, \beta, \mathcal{P}) * N^{1-\eta}$

Une loi courante est $Q(N) = K_\eta * N^{1/2} \Rightarrow T_{total} \leq \theta(\alpha, \beta, \mathcal{P}) * N^{1/2}$

On a montré que le ratio $T_{total} / \mathcal{E}$ ne dépend pratiquement pas de N, $\mathcal{E}$ étant la longueur de la description hiérarchique. S'il suffit de dessiner une seule cellule par niveau hiérarchique alors on aurait $\mathcal{E} = \nu * \log_\beta N$

Ainsi : $\phi(\alpha, \beta, \mathcal{P}) * \log N \leq T_{total} \leq \theta(\alpha, \beta, \mathcal{P}) * N^{1-\eta}$

Exemple numérique

Etudions le cas d'une analyse hiérarchique par la méthode des halos pour une hiérarchie cohérente. Soit une **cellule** de surface S, de forme carrée, qui **contient** N ***éléments primaires*** plus ou moins enfoncés dans la profondeur de la hiérarchie.

Cette cellule produit un halo d'aire $8\mu * S^{1/2}$ si μ **est la règle de dessin maximum**. Le nombre d'éléments primaires dans cette aire est de $8\mu * \delta * S^{1/2}$ **avec** δ **densité des cellules terminales.**

Ainsi **pour l'un des tests du D.R.C. incrémental** le nombre *d'éléments primaires* **n à revérifier** (dans le contexte d'utilisation) pour chacune des **sous-cellules** obéit à la loi :

$n \leq k_\alpha * N^\alpha$ avec $k_\alpha = 8\mu * \delta^{1/2}$ et $\alpha = 0.5$

$= n \leq 8\mu * \delta^{1/2} * N^{1/2}$;

Le **nombre maximal** ***d'éléments primaires*** à vérifier au cours d'un test incrémental de D.R.C. est :

$C_{max} \leq \mathcal{P}^{1/2} * 8\mu * \delta^{1/2} * N^{1/2}$; ainsi si le nombre d'éléments primaires (rectangles / transistors...) de la conception est de l'ordre de 10^6, alors le nombre maximal *d'éléments primaires* à vérifier au cours d'un test incrémental de D.R.C. est au plus de l'ordre de $\kappa * 10^3$ avec $\kappa = \mathcal{P}^{1/2} * 8\mu * \delta^{1/2}$.

Le **temps d'un test incrémental** du D.R.C. obéit à la loi :

$t_i \leq k_\beta * N^\beta$ **avec** $1.2 \leq \beta \leq 1.5$ pour **N éléments à vérifier**; k_β dépend de l'ordinateur et de l'ensemble des règles à vérifier.

On a ainsi $\alpha\beta \leq 0.75 < 1$. Ce qui assure un **temps de vérification,** T_{total} **, en O (N)** pour des circuits composés de cellules assez hiérarchiques comme l'était la hiérarchie qui a servi à l'estimation.

Cas réel - Pour certaines cellules comme des tableaux (RAM, etc.) de cellules le D.R.C. par la méthode des hallos que l'on vient d'examiner est à elle seule inefficace, bien que l'on ait $\alpha\beta \leq 0.75 < 1$, car les hallos de ces cellules peuvent être aussi grands que les cellules elles-mêmes. En effet la réduction du nombre d'éléments à examiner n'est effective que partir d'une certaine valeur N_0 de N. Cette méthode provoque alors l'expansion complète des tableaux de cellules. Le groupage d'un certain nombre de cellules, comme le propose [WAG 84], assure alors la transformation de la hiérarchie des tableaux en une autre hiérarchie plus efficace, et plus proche du modèle théorique précédent, pour ce type de méthode.

7. SPECIFICATIONS POUR UNE ANALYSE HIERARCHIQUE

Les diverses stratégies de l'analyse hiérarchique sont ainsi très dépendantes de la méthodologie utilisée pour décrire les layouts hiérarchiques. Certains systèmes qui ne limitent pas les recouvrements de sous-cellules, restreignent cependant le partage des données à des niveaux de hiérarchie adjacents et utilisent ainsi des hiérarchies strictes. La plupart des hiérarchies de conceptions peuvent être transformées en hiérarchies strictes *(cf. §7.1.)*.

7.1. La réalisation de hiérarchies strictes

Pour une **hiérarchie stricte** les données ne peuvent être partagées qu'entre deux niveaux adjacents de la hiérarchie. Pour ce type de hiérarchie, un extracteur de connectivité électrique peut alors effectuer une analyse hiérarchique par utilisation d'un algorithme d'équivalence conventionnel. Un *résumé* est alors construit pour chaque cellule analysée. Ce *résumé* contient des éléments particuliers appelés **ports** - qui constituent justement *l'interface* de la cellule pour les contextes qui l'instancient - et leur graphe d' équivalence. L'analyse de la cellule est alors effectuée selon l'algorithme décrit *en §5.2.*

[MCCA 84], [STEV 84] extraient de façon incrémentale pour chaque cellule créée une **cellule Shell** constituée par tous ***éléments porteurs des signaux d'entrée et de sortie de la cellule.*** Les éléments de ces *cellules Shell* sont alors les porteurs de ***l'information de connectivité électrique*** extraite dans la cellule. Ces *cellules Shell* constituent ainsi les *résumés* des cellules dont elles sont extraites pour une certaine analyse de connectivité électrique. Les auteurs de cette méthode désirent en effet utiliser ces *cellules Shell* d 'une part pour observer et analyser plus facilement la connectivité électrique globale d'une cellule à travers la hiérarchie de conception; d'autre part ils désirent pouvoir établir un contact électrique sur des éléments *porteurs des signaux d'entrée et de sortie de la cellule,* mais à priori cachés dans la profondeur de la hiérarchie, c'est-à-dire entre des éléments primaires qui n'appartiennent pas, à priori, à des niveaux de hiérarchie adjacents. Pour l'analyse de la connectivité électrique, le *résumé* d'une cellule doit alors contenir *tous les éléments porteurs des signaux d'entrée et de sortie de la cellule*, ainsi que leur *graphe d'équivalence électrique (cf. chap. 3).* Ce *résumé* de la cellule assure alors une extension de la notion de *ports* de connexions et représente alors *l'interface* de la cellule pour les contextes qui l'utilisent. Une **hiérarchie stricte** est ainsi induite à partir d'une hiérarchie qui, à priori, ne paraissait pas l'être.

La définition des éléments terminaux ou ports d'une cellule peut être implicite. Sa reconnaissance est alors soit géométrique [TRIM 81 & ROW], soit structurelle et déterminée par la manière dont la cellule est utilisée dans la hiérarchie de conception. Dans le second cas, il est nécessaire d'étudier la manière dont ces cellules sont utilisées dans les divers contextes en analysant alors de manière ascendante la hiérarchie de conception *(figure 8)* [NEL 85], [SCO 86], [SHAN 86]. Une définition assez large de la notion de *port* et d'*interface d'une cellule* permet alors la transformation d'un certain nombre de hiérarchies quelconques en hiérarchies stricte. L'ensemble des ports d'une cellule et une information complémentaire constituent ainsi un *résumé* de la cellule pour le partage de ses données, *résumé* qui spécifie bien l'usage que l'on en fait.

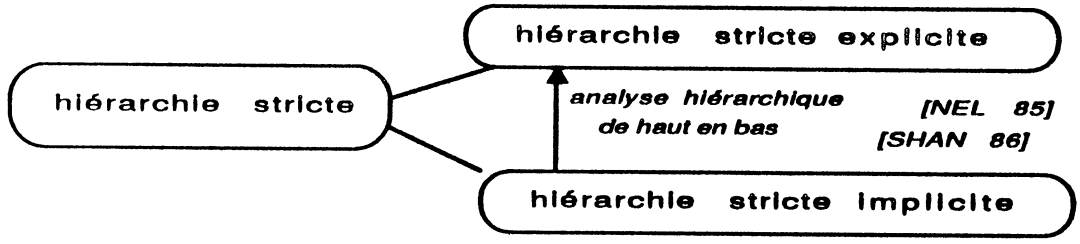


Figure 8. Une hiérarchie stricte peut être définie implitement par la manière dont les modèles de cellules sont utilisés dans les divers contextes qui les instancient. Ainsi l'observation de l'utilisation d'une cellule permet de reconnaître les éléments qui assurent le partage des données entre deux niveaux adjacents de la hiérarchie . Cette reconnaîssance est réalisée par une analyse de l'utilisation d'un symbole dans une analyse ascendante de la hiérarchie . Une définition assez large de la notion de port et d'interface d'une cellule permet alors la transformation d'un certain nombre de hiérarchies quelconques en hiérarchies strictes

7.2. L'utilisation des hiérarchies non cohérentes

Lorsque les recouvrements de sous-cellules sont autorisés, les hiérarchies produites sont incohérentes. Ces hiérarchies peuvent éventuellement être transformées en hiérarchies cohérentes [NEW 82].

Lorsque la hiérarchie à analyser n'est **pas cohérente**, pour une analyse comme le D.R.C. les interactions des données d'un même niveau doivent alors être étudiées dans la **profondeur de la hiérarchie** par une **analyse récursive en profondeur**.

Cette analyse en profondeur produit soit des **expansions partielles** du layout et l'examen des interactions par un algorithme conventionnel, soit une **analyse récursive** complète des interactions, dans la profondeur de la hiérarchie, jusqu'à atteindre les interactions d'éléments primaires. Ces expansions partielles peuvent être soit locales [SCO 85] [TAY 84], soit globales par rapport à la cellule analysée [WHI 81][LEU 86]. Elles peuvent éventuellement conserver certaines informations hiérarchiques [LEU 86].

Ainsi par exemple l'analyse de l'interaction d'une connexion avec les sous-cellules qu'elle traverse, peut amener soit à l'expansion du layout qui l'entoure [LEU 86], soit à l'analyse récursive des interactions de cette connexion dans les

modèles des sous-cellules en interaction [JOH82], [BER 86]. Un algorithme de D.R.C. utilisant cette dernière stratégie est ainsi proposé au *chapitre 6.*

Lorsque la définition des ports ou éléments terminaux d'une cellule est explicite; ces éléments terminaux constituent alors **l'interface topologique** (électrique) de la cellule et spécifient donc la manière dont la cellule doit être réutilisée. En dehors de ces terminaux de connexion, les interconnexions ne doivent pas interagir avec les sous-cellules et les traverser sans créer d'interaction ou d'erreur d'espacement à l'intérieur. Le DRC hiérarchique doit donc contrôler que les sous-symboles sont bien utilisés, soit en effectuant une extraction hiérarchique préalable de la cellule vérifiée et et utiliser alors les règles du DRC de base, soit en établissant des règles particulières de garde entre les *éléments primaires* du plus haut niveau de la cellule avec les divers niveaux / pseudo-niveaux utilisés dans les sous-cellules, *spécifiant ainsi complètement l'utilisation des symboles* [SCH 85], [SCH 86]. Cette deuxième approche est plus stricte que la première et constitue une contrainte supplémentaire pour les concepteurs, même si les règles choisies pour maintenir la légalité sont simples.

La première approche produit une analyse beaucoup plus complexe, car il faut utiliser les règles de base du DRC à travers la hiérarchie de conception et, compte tenu des équivalences électriques qu'il est nécessaire de reconnaître malgré la hiérarchie. Cette approche cependant est la seule qui puisse convenir lorsque, par hypothèse, il est impossible d'intervenir sur le processus de conception. En revanche, une telle approche est applicable à une classe de conceptions beaucoup plus large.

8. LA FAISABILITE DES DIVERSES METHODES

La *planche 2* compare les différentes stratégies d'analyse hiérarchique de layouts structurés en fonction du type de hiérarchie utilisé. Sur cette planche, il est également possible de situer dans cet ensemble ***notre stratégie de vérification*** du layout par le D.R.C. ***La stratégie générale d'analyse proposée*** pour l'analyse des layouts sera présentée *aux chapitres 5 et 6.*

Le *chapitre 4* propose et évalue une méthode de recherche géométrique par *quadtree.* Cette méthode permet de réaliser des analyses conventionnelles de layouts avec une bonne complexité théorique et un bon comportement expérimental *(cf. chap. 7 & annexe 5).* Les *vérifications incrémentales* de layout structurés utilisent aussi cette méthode de recherche géométrique pour réaliser la recherche des *éléments primaires* à vérifier, dans la *profondeur* de la hiérarchie. Le *sous-système de recherche géométrique* proposé permet ainsi la vérification des hiérarchies non cohérentes.

Le *chapitre 5* propose ainsi une méthode d'analyse hiérarchique de la *connectivité et des propriétés électriques* sur des layouts qui sont, pour la relation d'équivalence électrique, structurés selon une **hiérarchie stricte**. En conséquence une méthode d'analyse conventionnelle *(union-find)* de la connectivité électrique a pu être utilisée extensivement pour l'analyse hiérarchique. La complexité de l'algorithme de base, qui recherche de manière géométrique les éléments éventuellement en contact, est, pour *N éléments primaires,* entre O (N log N) et O

($N \log^2 N$) et en pratique en O ($N^{1.1}$).*(chapitre 4 & 5).* De plus le nombre d'éléments des sous-cellules à considérer dans l'analyse incrémentale d'une cellule est une faible proportion de leur modèle. La combinaison de ces deux propriétés devrait assurer une analyse incrémentale performante. En effet, d'une part le nombre *d'éléments primaires* - ports de la cellule - à réanalyser, lorsqu'une cellule de N éléments primaires est instanciée, est inférieur à O ($N^{1/2}$), d'autre part le temps de vérification de N éléments primaires est inférieur à O($N^{3/2}$), le produit de ces complexités est donc inférieur à 1, ce qui assure la faisabilité de la méthode avec une complexité des temps cumulés de l'analyse incrémentale au plus en O (N) *(cf. §6.2).*

Le *chapitre 6* propose une méthode de *vérification hiérarchique* des *règles de dessin* sur des layouts qui sont structurés selon une **hiérarchie stricte** mais **non cohérente.** En conséquence pour cette analyse du layout *le résumé* de la cellule est *la cellule elle-même.* L'extension du *D.R.C. conventionnel* par la technique des *résumés* - constitués alors par des *halos* - semblait impraticable, sans ajouter des hypothèses restrictives de recouvrements de sous-cellules; de plus la prise en compte de la garde maximale des règles de dessin d'un D.R.C. symbolique constituait un pire cas pour la technique des halos. Pour la vérification des règles de dessin de ces hiérarchies incohérentes, une méthode générale **d'analyse récursive en profondeur sans expansion** *(§7.2.)* des diverses interactions d'éléments fut donc proposée et développée.

Le *chapitre 7* présente les essais et résultats d'une première stratégie d'extension de *l'outil conventionnel de D.R.C.* par la méthode des *halos* - selon une hypothèse d'utilisation d'une **hiérarchie cohérente** (non recouvrement des sous-cellules). Le manque d'efficacité et de sécurité de cette méthode, pour l'analyse de conceptions peu cohérentes, a nécessité la recherche et la proposition de la **stratégie d'analyse récursive en profondeur** proposée *au chapitre 6.* Lorsque les hiérarchies analysées sont cohérentes, le nombre *d'éléments primaires* à examiner *en profondeur* lors d'une *vérification incrémentale* est en principe inférieur à celui qui est examiné par la méthode des *halos.* Les résultats théoriques observés pour l'analyse incrémentale des hiérarchies spécifiques, ont montré qu'avec une bonne structuration des données et de bons algorithmes de base, il suffit d'un gain assez faible dans le nombre *d'éléments primaires à vérifier* par rapport à une analyse conventionnelle pour obtenir globalement un temps de vérification théorique en O (N) et ainsi une amélioration de la complexité de la vérification. Des algorithmes efficaces sont proposées comme outils de base de l'analyse incrémentale et hiérarchique; ces algorithmes doivent ainsi assurer pour des hiérarchies presque cohérentes une complexité théorique proche de O (N) *(cf. §6.2.).* De plus *la méthode de recherche géométrique* proposée *(chap. 4)* et d'analyse en profondeur de la hiérarchie *(chap. 6)* doit permettre en plus la prise en compte de hiérarchies non restrictives. Même dans le cas d'un faible gain sur le nombre d'éléments à analyser, le **D.R.C. incrémental** doit permettre l'**analyse globale** du layout avec une bonne complexité, si on utilise pour effectuer l'analyse, des algorithmes de base qui ont un bon comportement *(cf. §6.2.).* De plus cette *analyse globale* est fragmentée et répartie tout au long du processus de conception, ce qui doit en permettre une utilisation pratique pour une conception beaucoup plus sûre.

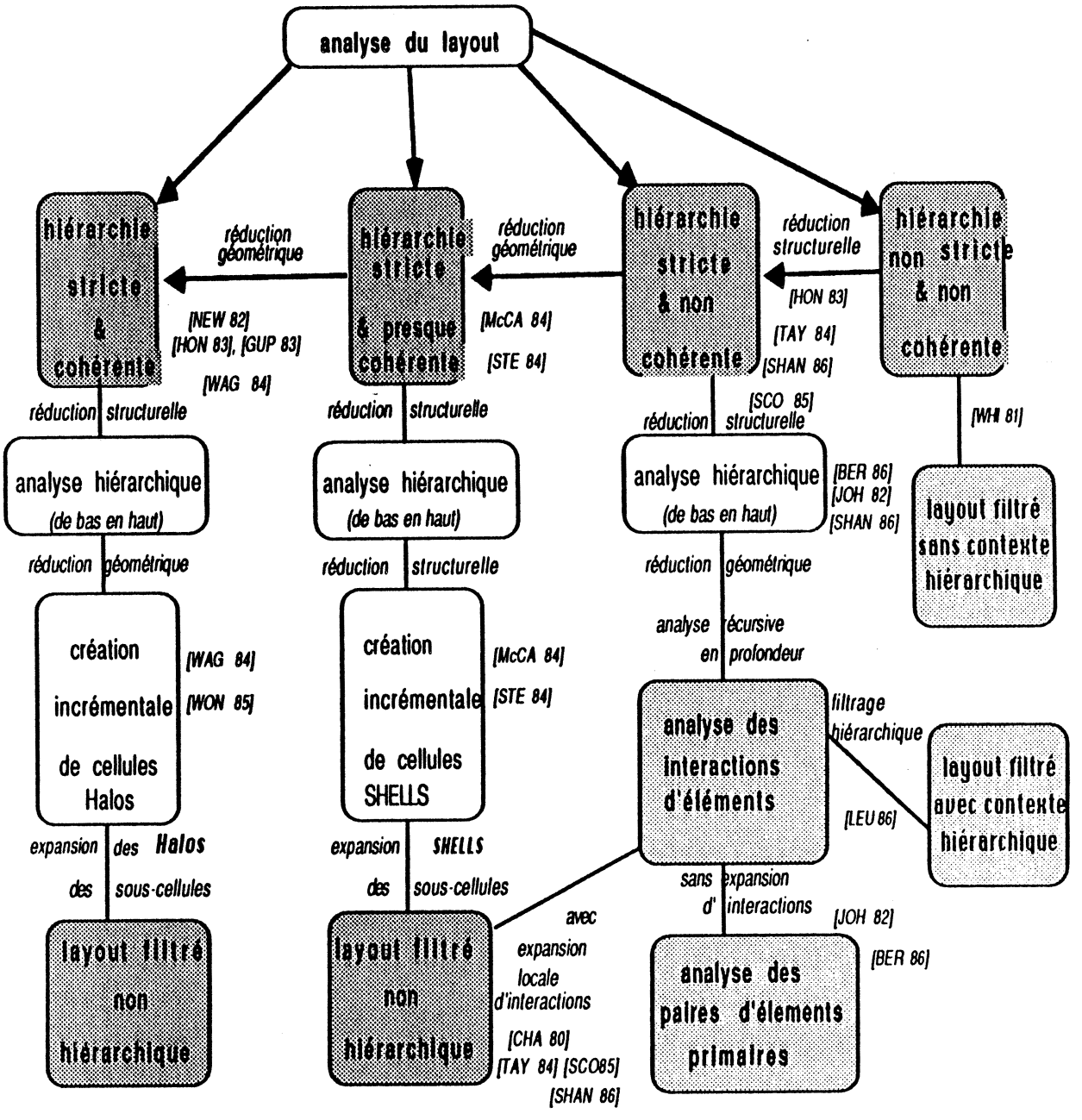


Planche 2 - L'analyse hiérarchique du layout et les diverses stratégies de réduction de la complexité - *Les méthodes qui analysent le layout de* ***manière ascendante*** *effectuent généralement l'analyse incrémentale des cellules tout au long du processus de conception. Lorsque la hiérarchie à analyser n'est* ***pas cohérente****, pour une analyse comme le D.R.C. les interactions des données d'un même niveau doivent alors être étudiées dans la profondeur de la hiérarchie par une* ***analyse en profondeur*** *. Cette analyse en profondeur produit alors soit des expansions partielles du layout et l'examen des interactions par un algorithme conventionnel., soit une analyse récursive complète des interactions jusqu'à atteindre les interactions d'éléments primaires. Ainsi par exemple l'analyse de l'interaction d'une connexion avec les sous-cellules qu'elle traverse, peut amener à l'expansion du layout qui l'entoure ou à l'analyse récursive des interactions de cette connexion dans les modèles des sous-cellules en interaction. Cette dernière stratégie a été retenue et sera présentée au chapitre 6 , qui propose une stratégie générale.*

Chapitre 3

LES METHODES

D'ANALYSE GEOMETRIQUE ET TOPOLOGIQUE

DES CONCEPTIONS VLSI

1. LE LAYOUT VERIFIE ET LA GEOMETRIE DES RECTANGLES

Les données géométriques qui représentent le layout d'un circuit (layout) sont constituées des motifs des différents niveaux de masques (layers). Les masques utilisés dans la fabrication des circuits intégrés sont fréquemment exprimés comme une collection de rectangles (ou polygones simples) avec des côtés parallèles à deux directions orthogonales (ou 45° diagonales). Chaque rectangle est soit une portion de métal, soit une région d'implantation d'ions, soit un contact .. [MEA 80].

Ces rectangles sont placés dans le plan selon certaines règles de conception qui spécifient leur espacement minimum ou recouvrement minimum. Les conditions d'espacement et de recouvrement proviennent des contraintes de la technologie nécessaires pour assurer les relations électriques des différents composants que le layout représente, tout en évitant les contacts non désirés, en dépit de possibles fluctuations de production. Par ailleurs, l'optimisation de la surface totale du circuit (contrainte économique) impose l'optimisation de l'espace laissé par les divers motifs (et la minimisation des interconnexions). Ce compromis nécessaire entre l'optimisation et la légalité du layout généré, est la source de nombreuses erreurs de conception.

Ainsi l'une des tâches principales de la conception des masques des circuits intégrés est la **vérification des règles de dessin.** Cette tâche est habituellement appelée **"D.R.C."** (Design Rule Checking). Par exemple, il peut être nécessaire de vérifier un espacement minimum λ entre le rectangle R et ses voisins (les autres rectangles). Pour vérifier cette situation, on peut élargir R au moyen d'un "halo" de largeur λ et vérifier si le rectangle R' ainsi obtenu a une intersection non vide avec l'un quelconque des rectangles avoisinants. Ainsi, la vérification des espacements est transformée en un problème d'**intersection géométrique**. Des variations de cette approche peuvent être utilisées pour vérifier les autres règles de conception, telles que les recouvrements, les inclusions.

Bien que les étapes ci-dessus concernent en un sens la "correction syntaxique" des masques, une autre tâche importante est la vérification que les masques sont fonctionnellement (c'est-à-dire topologiquement) corrects, c'est-à-dire réalisent bien les spécifications du circuit qu'ils décrivent. La première étape de cette tâche est l'interprétation de la géométrie des masques comme représentation d'un circuit électrique, ce qui correspond à l'identification des sous-ensembles de rectangles électriquement connectés (relation d'équivalence). Cette activité se nomme **l'extraction de circuit** (de nœuds et paramètres électriques). De manière plus abstraite, on peut imaginer un **graphe**, dont les sommets correspondent aux rectangles et dont les arêtes entre deux sommets correspondent aux équivalences réalisées par la relation "est en contact avec"; cette relation correspond aux diverses intersections de ces rectangles deux à deux lorsque ces rectangles appartiennent au même niveau de masques. Dans ce formalisme, l'extraction de circuit est résolue par la détermination des **composantes connexes** de ce graphe. Cependant, la détermination des composantes connexes ne résout pas tout à fait l'extraction de circuit. L'**extraction du contour réel de chaque composante connexe** est nécessaire à la résolution des divers paramètres électriques des nœuds électriques ainsi représentés.

Ainsi, la conception des circuits intégrés VLSI est une source importante de problémes géométriques mettant en jeu des rectangles comme les intersections de rectangles et les extractions de contour des composantes connexes. La résolution de ces problèmes géométriques doit être réalisée dans certains cas de manière dynamique (éditeurs, ..).

Des **algorithmes géométriques** déterminent les diverses propriétés d'une collection de rectangles. Une classification fondamentale peut ainsi être réalisée dans l'ensemble de ces algorithmes selon qu'ils opèrent dans un **environnement statique ou dynamique.** Dans le mode statique la collection des rectangles est entièrement disponibles avant que la tâche démarre. Le mode dynamique, au contraire, suppose la mise à jour dynamique de la collection et la détermination de ses diverses propriétés au fur et à mesure de l'introduction des données. On verra dans ce chapitre que les deux modes d'opération - statique et dynamique - impliquent des structures de données très différentes. Le mode statique a été largement étudié et les techniques algorithmiques correspondantes sont actuellement bien connues. Le mode dynamique, beaucoup plus complexe, est actuellement, en particulier à cause de l'explosion des VLSI, en pleine phase de développement et de recherche [PRE 85]; de plus, le comportement des algorithmes concernés par le mode dynamique est de prédiction plus difficile; les structures de données associées doivent cependant permettre de rapides mises à jour.

Les **algorithmes géométriques (computational geometry** [pre 85]) sont en général étudiés avec l'hypothèse que toutes les données résident en mémoire principale ou que les données couramment traitées proviennent d'un fichier à accès direct (mémoire secondaire). Par contre, les **algorithmes d'analyse et de vérification de layout** sont habituellement construits selon l'hypothèse que l'ensemble des données est stocké sur une mémoire secondaire à accès séquentiel et que seule une partie des données appartenant à une faible portion du plan de conception peut être résidente en mémoire principale. Ces différents modèles de traitement des données limitent les techniques de la géométrie algorithmique qui peuvent réellement être utilisées, surtout pour les "outils conventionnels" de vérification qui ne peuvent travailler que sur le layout expansé (sans hiérarchie).

Les algorithmes géométriques s'intéressent en général au comportement de "pire cas" des divers algorithmes. Or certains algorithmes qui ont un mauvais comportement de "pire cas" peuvent avoir un très bon comportement sur des "cas moyens" résultant d'une bonne distribution des données [BEN & HAK 80]. Cependant si on peut supposer que les motifs de layouts sans hiérarchie sont uniformément distribués [BEN & HON 80] il n'en est pas de même pour les descriptions hiérarchiques; ces descriptions hiérarchiques peuvent ainsi produire des comportements de pire cas aux "bons algorithmes" des outils conventionnels (analysant un layout sans hiérarchie).

2. INTRODUCTION AUX TECHNIQUES ALGORITHMIQUES DE RECHERCHE GEOMETRIQUE

On suppose que les recherches géométriques sont effectuées sur une collection d'objets d'un même plan tels que des points, des segments ou des rectangles et selon divers modèles de selection. Les algorithmes de recherche géométrique distinguent et résolvent séparément :

1. **Les problèmes de localisation** où la collection des objets représente une partition du plan en "régions" et où la recherche correspond, par exemple, à la localisation d'un point du plan.
2. **Les problèmes de selection des objets d'un domaine** (range searching) où la collection représente soit des points, segments, rectangles et où le domaine de recherche est une région du plan.

Les differents types d'intersection de 2 rectangles, selon l'algèbre ensembliste, sont ainsi généralement distingués par la géométrie numérique (computational geometry). Des algorithmes souvent différents reportent ainsi, soit toute les paires d'intersections, soit toutes les paires ordonnées d'inclusions.

La **géométrie algorithmique (computional geometry)** optimise les divers problèmes géométriques en recherchant pour chacun les algorithmes qui ont le meilleur comportement de pire cas. Ces études théoriques ne peuvent en effet prendre en compte les contraintes réelles de l'environnement d'utilisation; elles fournissent cependant une borne inférieure pour les divers coûts, espace mémoire, preprocessing, mise à jour, temps de recherche, ainsi qu'une bonne indication des meilleurs algorithmes. Cependant, il est remarqué dans [BEN 80 & HAK] que des algorithmes qui ont un mauvais comportement de pire cas, peuvent avoir cependant un excellent comportement pour des "cas moyens" et un certain type de distribution des données, ce qui est généralement le cas des conceptions VLSI non structurées.

L'hypothèse que les descriptions géométriques des circuits VLSI sont en général constituées de petits objets assez uniformément distribués (layout expansé), a guidé nos recherches vers des algorithmes non conventionnels (pour analyser des layouts hiérarchiques) de recherche géométrique d'intersection ou d'inclusion en mode dynamique. Bien que non optimaux en mode statique, ces algorithmes doivent cependant pouvoir être utilisés de manière satisfaisante.

Un layout de VLSI est normalement spécifié comme une collection hiérarchique de cellules, où chaque cellule contient des formes géométriques appartenant aux divers niveaux de masques et des références à des sous-cellules (définition de cellules instanciées).

Nous allons voir les divers mécanismes généralement utilisés pour faciliter les recherches géométriques pour des conceptions VLSI réelles , en prenant en compte l'ensemble des contraintes de l'environnement (espace mémoire et temps de réponse).

Nous supposerons que la collection d'objets est formée d'un ensemble de **N rectangles** de diverses couleurs (associées à chaque niveau de masque). La collection de rectangles est stockée comme une collection d'enregistrements; chacun

de ces enregistrements est identifié par un **4-tuple (x1,x2,y1,y2) formé des coordonnées inférieure droite et supérieure gauche du rectangle enregistré**. Les recherches cartésiennes du plan de conception constituent ainsi un cas particulier de la recherche par "clefs multiples" [KNU 73]; nous considérerons parfois ces 4-tuples comme des points du 4D-espace formé par le produit des domaines de valeurs des 4 attributs x1, x2, x3, x4.

La **recherche géométrique** est aussi identifiée par un **4-tuple (X1,X2,Y1,Y2)** qui spécifie un **domaine de recherche** et correspondent généralement à la recherche des objets ou rectangles couvrant ce domaine. La capacité des méthodes existantes à supporter l'entrée dynamique des données ainsi que l'intérêt de ces méthodes pour des layouts hiérarchiques sont discutés ci-après.

3. VLSI ET MECANISMES DE RECHERCHE GEOMETRIQUE EXISTANTS : LEUR DYNAMISME

3.1. Les listes chaînées

La plus simple technique pour représenter des rectangles est de les garder dans une simple liste. Cette technique fut utilisée dans le système Cæsar [OUS 81] : chaque cellule est représentée par une liste de rectangles pour chacun des niveaux de masques. Même à travers des opérations telles que les recherches du plus proche voisin, cette structure est assez efficace lorsque les grosses conceptions sont cassées hiérarchiquement en petites cellules [OUS 84]. Cependant des fonctions plus complexes telles que les extractions de circuits, les DRC et les compactions ne peuvent être implémentées efficacement avec des listes de rectangles.

3.2. Le "bucketting" ou "diviser pour régner"

La plus populaire des structures pour VLSI est basée sur le **"bucketting"** ou **méthode des casiers** (bin-based systems). L'aire du circuit est divisée en casiers au moyen d'une grille. Tous les rectangles ayant une intersection non vide avec un des "buckets" sont attachés ensemble et un 2D-tableau (directory) est utilisé pour localiser les listes des divers "casiers" ainsi remplis. La taille des "casiers" / "buckets" est un compromis entre le temps et l'espace. Lorsque les "casiers" deviennent plus gros, cela prend plus de temps pour inspecter les listes des casiers concernés par la recherche; lorsque les "casiers" deviennent plus petits, les rectangles peuvent recouvrir plusieurs casiers et ainsi occuper l'espace comme membres de plusieurs listes.

La méthode des casiers est meilleure lorsque les rectangles ont des tailles presque uniformes et sont aussi assez uniformément distribués ; cette méthode est inefficace (temps / espace) lorsque ces conditions ne sont pas réunies. Un cas pathologique est une cellule hiérarchique avec seulement quelques grosses cellules et beaucoup de petits rectangles qui les interconnectent. La méthode des casiers est donc difficile à utiliser directement sur des layouts hiérarchiques qui ne sont uniformes qu'en certaines régions de l'espace (sans hiérarchie) et qui sont presque vides là où les sous cellules référencent les symboles instanciés.

Diverses solutions ont été proposées comme solutions au problème de la non-uniformité. [NIE 84] propose un partitionnement du 4D-espace pouvant être associé à la collection de rectangles en **casiers de taille variable** et correspondant à une certaine occupation ; les divers casiers pouvant être divisés ou réunis lors d'insertions ou suppressions de rectangles.

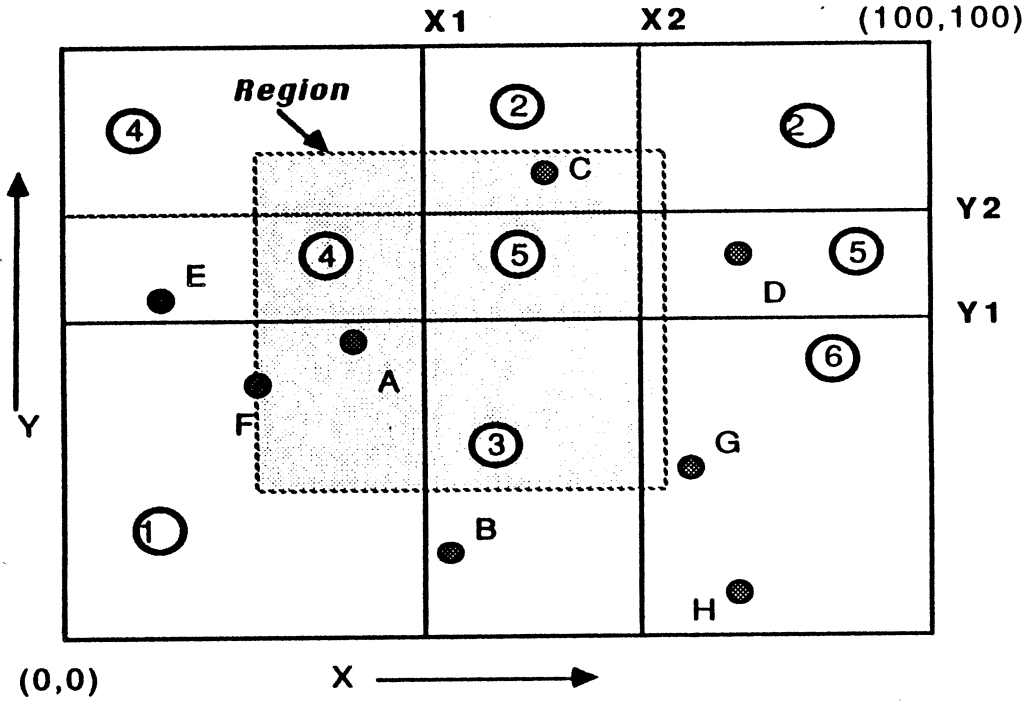


*Figure 1 - **Méthode des casiers adaptative** [NIE 84] - **Grid Directory** - La séquence d'entrée des données est A, B, C, D, E, F, G, H - La capacité des buckets est ici de 2 enregistrements - Ici chaque **point** ou **enregistrement** a k= 2 attributs différents, mais la méthode se généralise à **K attributs** ou **dimensions** - Le directory de la collection des enregistrements est formé de 9 casiers mais seulement de 6 buckets convexes, numérotés de 1 à 6. Le diectory formé par la grille ci-dessus peut être gardé sur un disque. Les échelles X et Y appelées échelles linéaires définissent la partition du domaine des valeurs de chaque attribut; elles doivent être résidentes en mémoire principale. Elles permettent l'accès aux casiers appropriés et améliorent la recherche des éléments d'un **domaine de sélection** - ou **région** de recherche - du **k-espace** des données.*

Divers **4D-trees** ont aussi été proposés dans le but d'accéder rapidement aux éléments d'un certain voisinage. Les **4D-trees adaptatifs** dits aussi **4D-tries** [BEN 79] réalisent un **"bucketting hiérarchique"** des rectangles du layout en des buckets d'espace variable, mais ayant presque le même nombre d'éléments. Les 4D-trees [BEN 75] réalisent une organisation spatiale de ces rectangles en les faisant correspondre aux nœuds d'arbres binaires particuliers, par partage récursif de la collection des rectangles parallèlement à chacun des axes (recherche du rectangle médian); 4D-trees et 4D-tries sont d'utilisation dynamique difficile (weight

balancing [WIL 78]) et ont pour les recherches géométriques, un mauvais comportement de pire cas, O ($N^{3/4}$) pour N rectangles. Comme ils ont un trés bon comportement en mode statique et lorsque la distribution des données est régulière, leur gestion dynamique est toujours un problème ouvert.

[KED 82] propose aussi un **"bucketting hiérarchique"** des rectangles du layout comme solution au problème de la non-uniformité : les casiers créés pour partager les rectangles du layout sont associés aux divers nœuds d'un quadtree *(chapitre 4)* ; cette méthode appartient à la classe des **quadtrees adaptatifs** [SAM 84] et en est une variante intéressante pour la gestion dynamique des données. Choisie comme méthode de base de recherche géométrique pour notre système de CAO, cette méthode a été améliorée de manière significative et sera discutée en détail au *chapitre 4.*

3.3. Méthodes d'accès direct

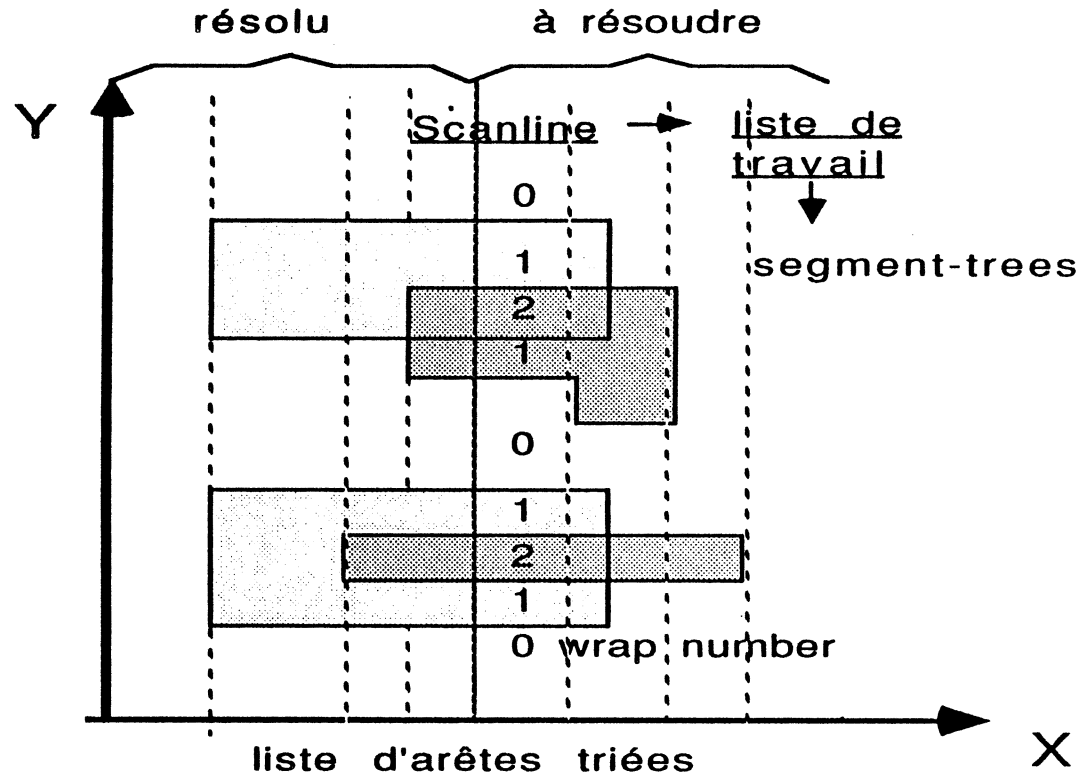
Les **méthodes d'accès direct** constituent un raffinement des méthodes d'organisation des données du layout par division. L'approche la plus grossière pour minimiser le temps de recherche, consiste dans le calcul préalable des réponses de toutes les recherches possibles (**locus approach**). Ainsi le temps de réponse pourrait être réduit à O (1) puisqu'un simple accès mémoire pourrait répondre à la recherche. Mais cette approche est très coûteuse en espace mémoire et irréalisable.

Les **range-trees** avec un coût espace mémoire de $N(\log N)^3$, un coût de recherche (query) de $(\log N)^4$ et un temps de preprocessing de $0(N\ (\log N)^3)$, illustrent l'irréalisme de ce type d'approche pour des applications VLSI [BEN 79 & FRI]. Il en est de même des **layered range-trees** au coût de recherche de $(\log N)^3$ [WIL 78][LUE 78] pour le même coût mémoire de $N(\log N)^3$.

3.4. Les approches incrémentales

La projection est une manière de réduire les problèmes du plan de conception en problèmes géométriques sur la droite réelle. Ainsi, le **balayage du plan par une droite (scan line)** de manière à construire de façon incrémentale de gauche à droite (ou de bas en haut) la solution générale d'un problème géométrique est une méthode statique très efficace pour résoudre les différents problèmes statiques de recherche géométrique d'intersection, d'inclusion de paires de rectangles, de calcul de contour et de surface des diverses composantes connexes [BAI 77], [BEN 80 & HAK], [NIE 82]. Ces méthodes de **scan line** utilisent des structures secondaires telles que les **segment-trees** [BEN 80 & WOO] et les **interval-trees** [EDE 80], [McC 81] pour la mise à jour de la liste des rectangles actifs (qui ont avec une intersection non vide avec le scan line).

Très utiles et souvent optimales pour l'étude de layouts complètement expansés (peu de données en mémoire principale), ces méthodes ne peuvent être utilisées dynamiquement. Par contre, elles peuvent constituer une solution efficace aux problèmes hiérarchiques [LEU 85], [ANN 84], [FOK 83].



- *entrée : figures & opérateur booléen;*
- *sortie : figures transformées (à gauche du scanline)*
- *support : liste d'arêtes triées en X,* ***segments-tries*** *(+ wrap number) en Y*

Figure 2 - ***Une méthode incrémentale par Scanline*** *- La géométrie des masques est représentée par des circuits fermés d'arêtes appelés figures. Les régions simplement connexes sont à l'intérieur d'une simple figure. Les figures sont orientées et l'intérieur d'un figure correspond à un côté d'une arête de cette figure. La méthode permet de réaliser des combinaisons booléennes entre layers. Les arêtes verticales sont triées dans l'orde lexicographique. Une ligne verticale balaie le layout de gauche à droite . Les différents pas du scanline sont les abcisses différentes des arêtes verticales. Le Scanline est divisé en plusieurs intervalles qui portent l'information de l'aire traversée. Ainsi le nombre de recouvrements d'un intervalle du scanline par une figure de l'entrée des masques est indiqué par le Wrap Number. Chaque arête horizontale marque le début et la fin d'un intervalle. Si le Wrap Number d'une figure s'accroît et exéde un seuil donné une nouvelle figure est générée ou une ancienne est étendue. Dans chaque cas une nouvelle arête est générée et stockée dans le layer de sortie. Cette arête produit le changement du W.N. Les figures de sorties sont fermées ou unies si le W.N. descend au dessous d'un certain seuil. L'orération UNION a un seuil de 1; l'opération ET a un seuil de 2 [BAR 80], [SCHI 85]. La complexité a comme borne inférieure celle du tri des arêtes en n log n.*

Une méthode dynamique construit la solution générale au fur et à mesure de l'introduction des rectangles. La méthode **Corner-stitching** est une méthode dynamique trés spécialisée qui élimine, par réécriture dynamique de chaque rectangle introduit, toutes les intersections de rectangles d'un même plan. Quelques plans "corner-stitched" forment ainsi le layout d'une conception. Les rectangles

appartenant à ces plans sont attachés entre eux par leurs coins inférieur gauche et supérieur droit; des **tuiles d'espacement** gèrent explicitement l'espace laissé entre les motifs. Une telle structure de donnée est particulièrement adaptée à la recherche dynamique des plus proches voisins d'un rectangle et à l'énumération dynamique des éléments d'un plan "corner-stitched" situés dans une région rectangulaire. Mais cette méthode a un mauvais comportement de pire cas pour la localisation d'un point du plan "corner-stiched", O(N) pour N rectangles; les passages nécessaires et fréquents entre les différents plans diminuent ainsi l'efficacité de cette structure.

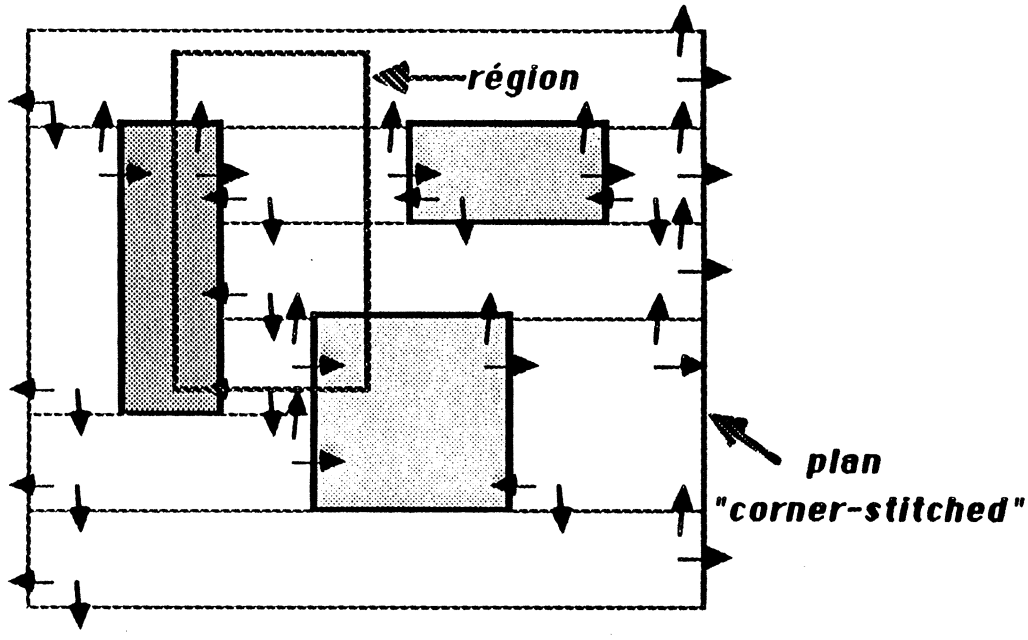


Figure 3 - ***Méthode Corner-stitching*** *[OUS 84] - Chaque point d'un plan Corner-stitched est contenu dans exactement une tuile. Ce plan corner-stitched est formé par un ensemble de tuiles attachées ente elles par leurs coins inférieur gauche et supérieur droit. Ce plan est formé de 3 tuiles solides et d'un certain nombre de tuiles d'espacement. Les tuiles solides sont de type différent selon le niveau de masque auquel elles appartiennent. L'enregistrement décrivant chaque tuile contient ainsi 4 pointeurs vers les enregistrements des autres tuiles. Cette structure produit une sorte de tri bi-dimensionnel et permet ainsi l'énumération des tuiles couvrant (au moins partiellement) un* ***domaine de recherche ou région*** *de selection.*

L'**éditeur Magic** utilise et maintient dynamiquement une telle structure de données, avec la possibilité cependant d'un certain recouvrement des sous cellules, soit entre elles, soit par des interconnexions [OUS 84 & HAM]. Cette méthode permet des extractions [SCO 85] et DRC incrémentaux [TAY 84] et même dynamiques.

3.5. Réalisme et dynamismes des diverses approches

Les approches de type **"diviser pour régner"** que nous avons abordées ont toutes des **coûts en espace mémoire proportionnels au nombre d'éléments**, bien qu'une certaine constante multiplicative (K) plus ou moins

importante les caractérise (K = 4 pour les 4D-trees). En général, elles ont pour la recherche des rectangles localisés sous un point donné un mauvais comportement de pire cas ($O(N^{3/4})$ pour les 4D-trees) [PRE 85], mais certaines de ces structures ont un très bon comportement pour des cas moyens [ROS 85], [BER 85] (en particulier pour les layouts sans hiérarchie).

Leur comportement en hiérarchie est souvent de prédiction difficile. Ces structures ne supportent pas facilement l'introduction dynamique des données, et utilisées comme telles, elles peuvent conduire à une dégradation progressive des performances. C'est le cas par exemple pour les K-D trees. La gestion dynamique de ces structures est souvent coûteuse et d'implémentation délicate [WIL 78].

Une étude assez importante sur la **recherche géométrique en mode dynamique** a été faite par [LEE 80 & WOO], [EDE 80]. La collection de rectangles est organisée par des structures primaires puis secondaires, telles que des **combinaisons de range-trees et segment-trees / interval- trees,** dans des structures appelées **SRI-pyramid** [EDE 82]. Des combinaisons différentes résolvent et optimisent séparément les differents types d'intersections de rectangles, ce qui rend ces structures de données très théoriques. De plus, le stockage de ces structures n'est pas en O (N) puisque les segment-trees ont à eux seuls un espace mémoire de O(NlogN) pour N segments et les range-trees de O(NlogN) pour N points d'un 2D-espace [PRE 85].

4. L'ANALYSE TOPOLOGIQUE ET SEMANTIQUE DU LAYOUT

La tâche d'un extracteur de circuit est de déterminer le circuit réalisé par la **description géométrique** que constitue **le layout** de ce circuit. Le niveau de détail dans la description de ce circuit varie largement d'un extracteur à l'autre selon l'usage que l'on veut faire du circuit ainsi calculé. Si l'on désire seulement vérifier le circuit extrait avec les spécifications du circuit (schémas), la description a seulement besoin d'inclure les transistors et leur connectique [WAG 84]. Quand le circuit extrait est utilisé pour des analyses de temps ou des simulations détaillées, beaucoup d'information doit être alors extraite du layout [McCO 84], [BAS 83]

Les interconnexions de transistors peuvent être interprétées comme des nœuds. Un **nœud** est une région équipotentielle qui possède une **résistance parasite** et **une capacité par rapport au substrat** que les vérificateurs de connectique peuvent ignorer. L'extracteur Crystal interprète un nœud electrique selon le réseau ci-dessous [SCO 85]. L'extraction des capacités internodales est aussi utile [TAR 83].

La description résultante du circuit est alors faite de trois sortes de composants : **les nœuds** décrits ci-dessus (ayant une résistance et une capacité), les **transistors** et les **capacités internodales**. Les transistors correspondent aux dispositifs électriques explicitement / implicitement placés par le concepteur. Chaque transistor a une grille, un terminal lié au substrat et deux terminaux de diffusion (ou plus). Les capacités internodales ont chacune deux terminaux et une valeur de capacité.

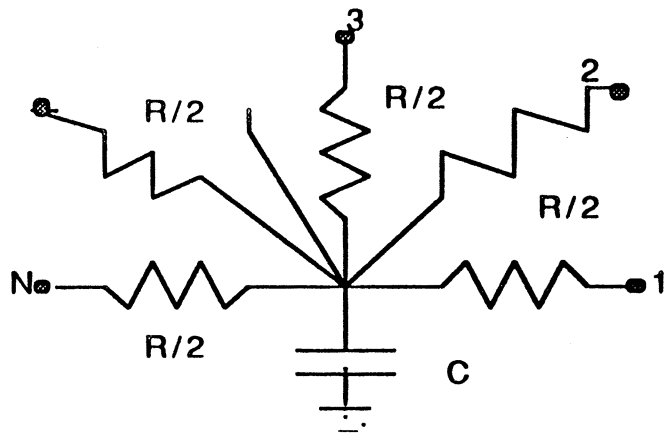


*Figure 4 - **Interprétation des résistances et capacités de substrat des noeuds électriques** - Chacun des points 1, 2 , ..,N ci-dessus est une connexion au noeud par un **terminal.***

4.1. Extracteur de base

L'entrée d'un extracteur est une description hiérarchique du layout; chaque cellule de layout peut contenir l'information de niveau masque et l'information des sous cellules. Le circuit extrait pour une cellule devrait, dans une hypothèse d'analyse hiérarchique, contenir aussi le circuit décrivant l'information associée aux masques et les références aux circuits de ses sous cellules, plus les ajustements et interconnexions entre ces sous cellules. L'information extraite du dessin des masques (symbolique ou physique) constitue l'extracteur de base qui ignore la hiérarchie. On peut imaginer que l'extracteur de base opère en trois passages à travers l'information géométrique du layout : le premier identifie tous les **nœuds**, le deuxième traite les **transistors** et le troisième calcule les **capacités internodales**.

4.2. Extraction des nœuds électriques. Diverses techniques

L'opération fondamentale d'un extracteur est de reconnaître tous les rectangles / polygones de la description géométrique qui appartiennent au même nœud électrique. Cette reconnaissance peut être réalisée par diverses méthodes.

4.2.1. Technique d'extraction "depth first"

L'algorithme d'extraction géométrique d'un nœud électrique peut consister en l'examen récursif des voisins de chaque rectangle reconnu comme appartenant au nœud courant et marquer les rectangles adjacents par le même nom / numéro. Une **table de règles de mise en contact** permet de savoir si deux rectangles adjacents sont électriquement en contact selon leur nature (couleur). Les étapes de cet algorithme sont simples. On examine chaque rectangle selon l'action ci-après :

***examiner -et- marquer** (nœud n, rectangle t)*
1. regarder si le rectangle t a déjà été marqué comme appartenant au nœud n
sinon
2. marquer le rectangle t comme appartenant au nœud n ;
3. pour chaque rectangle voisin de t électriquement connectés à t examiner -et-marquer (nœud n, rectangle voisin (t))
fin

***entrée** : un ensemble de rectangles à examiner et à marquer d'un nom de nœud*
***sortie** : un ensemble de nœuds électriques (composantes connexes)*

Cet algorithme a besoin d'un rectangle de départ. Un processus d'énumération géométrique ou numérique de tous les rectangles d'une cellule permet ainsi de trouver tous les nœuds. Si un rectangle a déjà été marqué par un nœud, il est sauté, sinon un nouveau nœud est créé et l'algorithme de recherche de nœud est appliqué à ce rectangle.

Cet algorithme est donc l'application de la recherche des **composantes connexes** dans un **graphe d'équivalence** par une **technique "depth first"** [AHO 74], [TAR 75]. Les listes d'adjacences sont recherchées géométriquement, ce qui évite l'extraction et la maintenance des listes d'adjacences de rectangles électriquement équivalents. Cette méthode est utilisée explicitement par Magic [OUS 84 & HAM], [SCO 85]. Elle est aussi utilisable à postériori pour retrouver dynamiquement un nœud électrique à partir d'un rectangle de départ et peut en conséquence permettre la visualisation interactive d'un nœud électrique quelle que soit la manière dont il a été préalablement extrait *(chapitre 5, actions Spanner et h-Spanner).*

On peut cependant reprocher à la méthode précédente de ne pouvoir traiter ainsi que des nœuds ou parties de nœud explicitement connexes. En effet lorque les relations electriques sont recherchée par intersection géométrique la relation d'équivalence électrique n'est pas vraiment distinguée de la relation de connexité géométrique. Il est difficile avec une telle méthode d'exprimer une relation électrique purement structurelle. Des informations complémentaires doivent cependant permettre certaines équivalences structurelles et la détection des conflits de noms.

4.2.2. L'algorithme UNION - FIND

Une composante électriquement connexe est une classe d'équivalence de la relation **"être en contact"**. Le problème de représentation des classes d'équivalences peut être vu par la définition de structures qui permettent d'effectuer des opérations de **fusion** (de deux composantes connexes). **Les arbres de Fischer-Galler** [FIS 72] [GAL 64] permettent de telles fusions et sont utilisés par l'algorithme **UNION - FIND** pour représenter les composantes connexes.

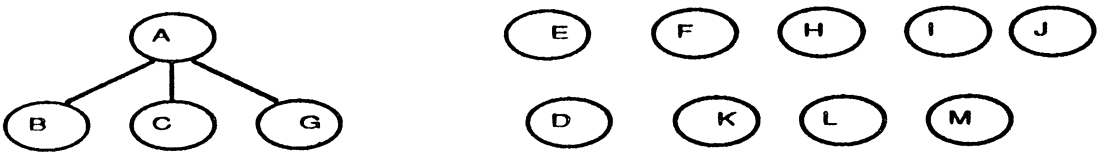
Nous pouvons désirer savoir si le rectangle X est électriquement connecté au rectangle Y sans avoir besoin de connaître véritablement le chemin qui connecte X à Y. Ce problème a été étudié pour calculer plus généralement des ensembles d'objets [AHO 74]. Les graphes d'équivalence correspondent aussi d'une certaine manière à des ensembles d'objets lorsque les arcs entre les sommets du graphe ont le sens "est dans le même ensemble que" (c'est-à-dire "est équivalent à"). Pour des ensembles, la question duale de "X et Y sont-ils équivalents ?" est "X et Y appartiennent-ils au même sous-ensemble ?" (partition).

Etant donné un ensemble d'arcs, il est toujours possible de construire une représentation du graphe d'équivalence par listes d'adjacences explicites ou implicites (recherche géométrique) comme cela a été vu au paragraphe précédent. La technique "depth first" assigne à chaque sommet (ou nœud) du graphe, c'est-à-dire à chaque rectangle du layout, le nom, ou index, de sa composante connexe, afin que les questions "est-ce que x est connecté à y ?" puissent avoir une réponse coûtant seulement deux accès de table et une comparaison.

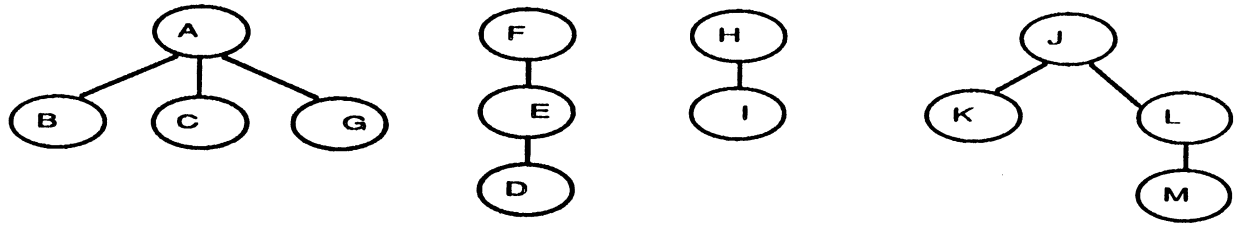
La **méthode UNION - FIND** que nous allons considérer maintenant, par rapport à la précédente technique ("depth first"), a l'avantage d'être **dynamique** : cette méthode peut supporter l'insertion et la suppression d'arcs de connexion, entrecoupées par des questions sur les équivalences déjà traitées. A cause de la correspondance avec le problème de sous-ensembles (la composition des rectangles en composantes connexes), l'addition d'une nouvelle équivalence (arc entre deux rectangles) est appelée **opération d'UNION** et les recherches d'équivalences sont appelés des **opérations FIND**.

Notre objectif est donc d'écrire une fonction qui puisse vérifier si deux rectangles X et Y sont dans la même composante connexe (même ensemble) et s'ils ne le sont pas de les y mettre (mettre un arc entre eux dans le graphe d'équivalence). Nous pouvons gagner en efficacité en utilisant une structure interne spécialement orientée pour produire de rapides opérations UNION et FIND. La structure interne que nous allons utiliser est une forêt d'**arbres de Fischer-Galler** [FIS 72], où **chaque arbre correspond à une composante connexe**. Nous avons besoin alors de trouver si deux rectangles appartiennent au même arbre et aussi de pouvoir combiner deux arbres en un seul, de manière efficace.

Initialement, tous les rectangles sont associés à un nœud et tous les nœuds sont dans un arbre différent (formé de ce seul nœud). Soient les rectangles A, B, C, D, E, ..., K, L, M. Ensuite on peut supposer l'insertion de l'équivalence AG, puis des équivalences AB et AC.

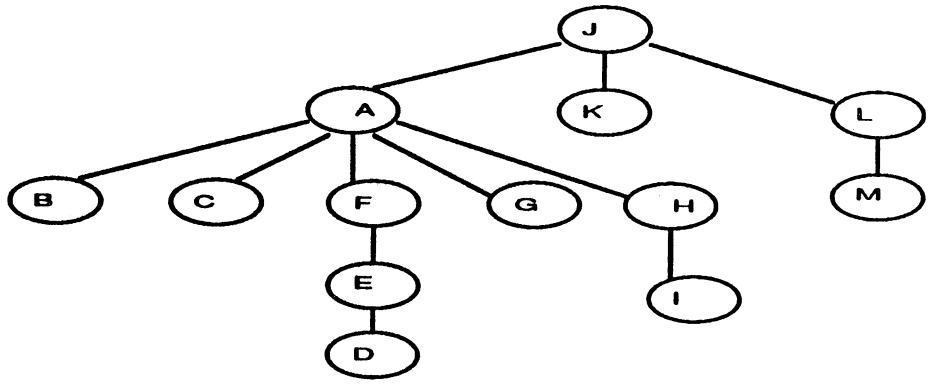


Les arcs LM, JL, JK construisent ainsi un arbre {J, K, L, M} .. , puis un arbre {F,E,D}, puis un arbre {H, I}.



Cette forêt indique alors à ce moment que les équivalences processées décrivent un graphe (réseau) avec quatre composantes connexes, ou de manière ensembliste, que l'ensemble des opérations d'union déjà processées ont construit quatre sous- ensembles (partition).

Maintenant, l'insertion à cette structure de l'équivalence FE ne change rien puisque F et E sont déjà équivalents, tandis que l'insertion de l'équivalence AF combine les deux premiers arbres ; GC ne change rien, mais GK et LG transforment l'ensemble de départ en un ensemble complètement connexe représenté par un seul arbre :



On peut représenter ces arbres par exemple par une table d'entiers DAD [1..N] qui garde pour chaque rectangle d'index I, l'index de son père (avec comme entrée 0 pour les rectangles qui sont à la racine d'un arbre). Pour chercher le père d'un rectangle "J" (index du rectangle assimilé à son <nom>), nous pouvons simplement faire J <- DAD [J] et trouver ainsi la racine de l'arbre auquel J appartient en répétant cette opération jusqu'à atteindre 0. Les opérations UNION et FIND sont alors très simples à implémenter :

```
action UNION_FIND (X, Y : integer; UNION : boolean; var FIND : boolean);
var I, J : integer;
begin
   I := X; while DAD [I] > 0 do I := DAD [I];
   J:= Y; while DAD [J] > 0 do J := DAD [J];
    if UNION and (I<>J) then DAD [J] := I;  FIND := (I<>J);
end;
```

Cette fonction retourne le flag FIND à VRAI si les deux composantes connexes de X et Y ont été trouvées différentes. Dans ce cas, et si le flag UNION est mis, elles

sont alors mises dans la même composante. La méthode utilisée est très simple : utiliser la table DAD pour atteindre la racine de l'arbre associé à chaque rectangle, ensuite regarder s'ils ont la même racine (représentant de la classe d'équivalence). Pour réunir les arbres de racines I et J nous mettons simplement un arc, c'est-à-dire nous opérons l'affectation DAD [J] <- I.

Cependant, l'algorithme décrit ci-dessus a un mauvais comportement de pire cas puisque les arbres ainsi formés pourraient dégénérer en listes et prendre alors un temps proportionnel à N^2 pour construire le graphe d'équivalence et produire aussi des temps de recherche d'équivalence proportionnels à N (N = nombre de rectangles dans le graphe).

Diverses méthodes ont été proposées pour prendre en compte ce problème lors de la fusion de deux arbres par une nouvelle équivalence. Quand un arbre de racine J doit être fusionné avec un arbre de racine I, un des nœuds reste racine, tandis que l'autre descend d'un niveau. Pour minimiser la distance de la racine à la plupart de ses descendants, il semble intéressant de prendre pour racine le nœud qui a le plus de descendants. Cette idée appelée **"weight balancing"** est facilement implémentée par la maintenance de la taille de chaque arbre (nombre de descendants de la racine) dans la table DAD, codé par un nombre négatif ou nul afin de pouvoir toujours détecter le nœud RACINE quand UNION - FIND remonte un arbre.

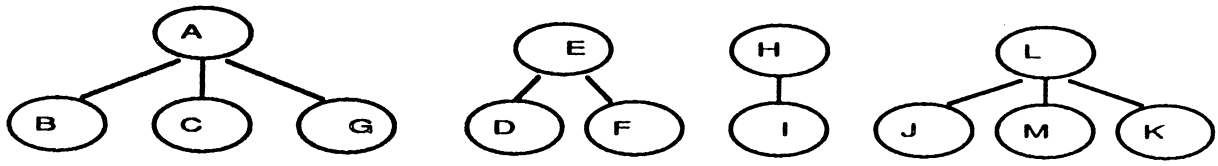
Idéalement, nous aimerions que chaque nœud pointe directement à la racine de son arbre (c'est-à-dire vers le représentant de la classe ou équivalent électrique). L'action UNION - FIND peut approcher cet idéal chaque fois qu'elle examine un nœud en compressant le chemin de ce nœud vers la racine de l'arbre ; ceci est facile à faire et peut rendre dynamiquement les UNION - FIND plus efficaces. Cette méthode peut être complétée en un second passage appelé **compression** ("path compression" de chaque rectangle) des arbres de Fischer-Galler.

La combinaison dynamique des actions "weight balancing" et "path compression" assurent l'efficacité tout au long du processus interactif. L'implémentation suivante montre que l'extra code nécessaire est un faible coût à payer pour éviter les cas de dégénérescence.

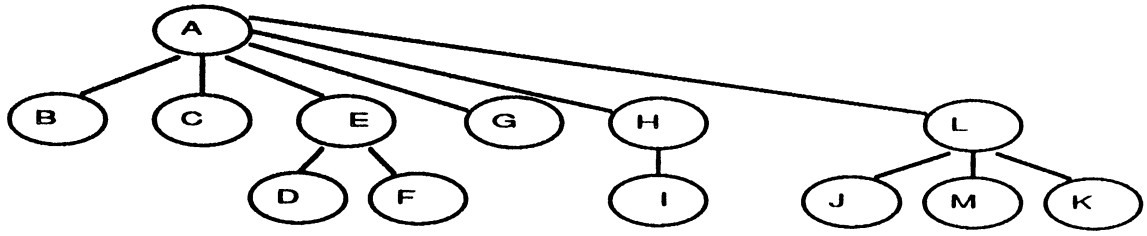
```
action FAST_UNION_FIND (X, Y : integer; UNION : boolean;
                        var FIND : boolean);
var I, J : integer;
begin
   I := X; while DAD [I] > 0 do I := DAD [I];
   J:= Y; while DAD [J] > 0 do J := DAD [J];
while DAD [X] > 0 do begin     (*path compression*)
    T := X; X := DAD [X]; DAD [T] := I; end;
while DAD [Y] > 0 do begin     (*path compression*)
    T := Y; Y:= DAD [Y]; DAD [T] := J; end;
if UNION and (I<>J) then
  if DAD [J] < DAD [I] then begin DAD [J] := DAD [J] +DAD [I]  -1; DAD [I] := J;
                  end else begin DAD [I] := DAD [J] +DAD [I]  -1; DAD [J] := I;
  end; (* path compression *)
  FIND := (I<>J);
```

end;

La table DAD est initialisée à 0 (dans une pré-initialisation des structures). Pour le même problème, grâce à la compression et à l'équilibrage dynamique, nous obtenons :



1e étape



2e étape

L'analyse des algorithmes UNION - FIND est de prédiction difficile sur des situations pratiques, parce que les performances dépendent non seulement des **paramètres N (nombre de rectangles)** et **E (nombre d'équivalences réalisées),** mais aussi du nombre d'opérations FIND (recherche simple d'équivalences) et ce qui est pire, de l'ordre dans lequel ces opérations UNION et FIND apparaissent. En particulier **n opérations FIND nécessitent au plus un temps de O(n * α(n)), où α(n) est une fonction qui croît beaucoup plus lentement que log n.**

L'espace utilisé par UNION - FIND est proportionnel à N (nombre de rectangles) quel que soit le nombre d'équivalences E. Ceci est un avantage sur "depth first" lorsque cette technique est implémentée par des listes d'adjacences (pas si un accès géométrique détermine les adjacences, mais alors il faut aussi prendre en compte l'espace occupé par la méthode d'accès géométrique).

[TAR 75] montra qu'**il n'existe pas d'algorithme pour ce type de problème qui puisse aller plus vite que E α(E), (E = nombre d'équivalences** ou UNION réalisées), **α(E) < 4**, même si E est très grand ... (α fonction pseudo-inverse de la fonction d'Akermann) [AHO 83]; α(E) croît beaucoup plus lentement que log E.

Divers extracteurs utilisent avec succès la technique UNION-FIND confirmant ainsi son efficacité théorique [JOH 82], [HEI 82].

4.2.3. L'extraction des paramètres électriques d'un nœud

La **capacité de surface** d'un nœud electrique par rapport au substrat est calculée par sommation des capacités de surface des divers matériaux utilisés. Son calcul est donc dérivé de celui de **l'aire** du nœud electrique; une pondération par unité de surface de la capacité par unité de surface des divers matériaux utilisés est à prendre en compte dans le calcul.

Le calcul de la **capacité de périmètre** d'un nœud electrique se calcule aussi par sommation des capacités de périmètre des divers matériaux utilisés. La valeur de la capacité par unité de longueur dépend des matériaux adjacents *(fig. 5)*. Son calcul est donc dérivé de celui du **périmètre** du nœud electrique; une pondération par unité de longueur des capacités de périmètre des divers matériaux en interaction est à prendre en compte dans le calcul.

On peut supposer aussi que la **résistance** du nœud electrique est calculable en première approximation à partir de **l'aire** et du **périmètre** du nœud electrique. On suppose alors qu'elle est équivalente à la résistance d'une région rectangulaire de même aire et même périmètre. Si cette approximation est suffisante pour la plupart des nœuds électriques et surtout pour de longs bus, elle est assez médiocre pour des nœuds de branchement.

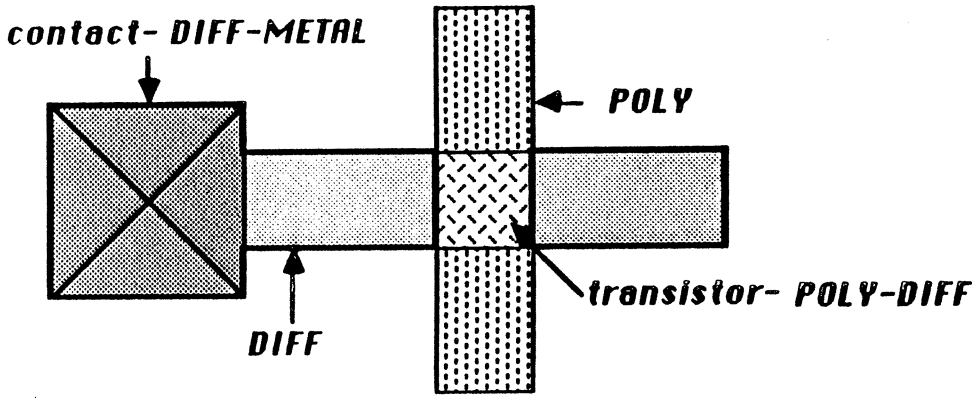


*Figure 5 - La **capacité de périmètre pour un rectangle / polygone simple** dépend des types / couleurs des autres éléments rencontrés sur son contour. Un fichier technologique fournit les valeurs de ces capacités de périmètre par unité de longueur en fonction des deux couleurs en interaction , assurant ainsi le paramètrage de la technologie. Par exemple un bord ayant d'un côté de la diffusion et de l'autre côté l'espace vide (considéré aussi comme une couleur) a une certaine capacité par unité de longueur commune, par contre un bord entre diffusion et contact-métal-diffusion n'en a pas, parce que ce type de contact contient de la diffusion dans sa fabrication; cette absence de capacité de périmètre est aussi fournie par le fichier technologique et une couleur est ainsi associée à chaque type d'élément multi-niveaux (contacts et transistors). Ainsi dans cette figure seulement les contours en trait épais ont une capacité de périmètre [SCO 85].*

Ainsi il est nécessaire d'avoir des algorithmes de calcul d'aire et de périmètre

d'un nœud electrique à partir des rectangles ou atomes qui le constituent.

Lorsque les rectangles du layout n'ont aucun recouvrement, l'aire d'un nœud(1) électrique est alors la somme des aires de chacun des rectangles qui le composent. Lorsque les rectangles peuvent se recouvrir au moins partiellement, il faut déduire les aires de recouvrement de l'aire totale des rectangles déjà sommés. Une technique de parcours "depth first" par recherche géométrique du nœud électrique à partir d'un de ses constituants peut être un mode de calcul de l'aire d'un nœud lorsque celui-ci est aussi géométriquement connexe.

Le **calcul des paramètres électriques** est difficile à maintenir de manière dynamique à cause des suppressions d'équivalences entre rectangles qui peuvent casser les nœuds électriques en deux. Ce calcul est réalisable après la phase d'extraction des différents nœuds électriques en parcourant l'ensemble des rectangles de la collection. Chaque rectangle de la collection est alors traité séquentiellement et l'intersection de ce rectangle courant avec les autres rectangles est étudiée par recherche géométrique et donne lieu à une mise à jour de l'aire du noeud electrique associé. Après le traitement de tous les rectangles la surface de chaque nœud électrique est connue.

Le périmètre d'un nœud électrique se calcule de la même façon. La recherche géométrique et les arbres de Fischer-Galler(1) permettent la sélection et la reconnaissance des rectangles équivalents et adjacents au rectangle traité. Les parties de périmètres du rectangle traité non couvertes par un rectangle de même nœud déjà traités assurent la mise à jour du périmètre de son nœud électrique.

4.3. Extraction des transistors

Dans une technologie MOS, un transistor est supposé avoir une grille qui recouvre le canal, une source et un ou plusieurs drains connexes au canal. Les **terminaux d'un transistor sont la grille, la source et les drains**. Tous les rectangles qui forment le canal d'un transistor peuvent être trouvés dans la phase de reconnaissance des divers nœuds électriques et avec le même algorithme. Le périmètre et la surface du transistor sont calculés comme pour un simple nœud électrique, ce qui permet le calcul de ses dimensions W et L [MEA 81].

Le transistor, dans la méthode UNION - FIND est donc représenté par un arbre de Fischer-Galler ; la méthode "depth first" permet de le parcourir exactement comme un nœud électrique, à partir d'un des rectangles qui le composent *(figure 7)*. Une fois que les transistors sont extraits, la reconnaissance des rectangles adjacents au transistor détermine les différents nœuds connectés à leurs sources et à leur grille.

Lorsque la description du layout est faite uniquement avec des rectangles appartenant à des niveaux physiques, ces rectangles doivent avoir été préalablement réécrits, lorsqu'ils interagissent pour créer des transistors et la couleur canal est prise en compte dans la réécriture *(figure 6)*. Le type de matériau adjacent à un transistor détermine le terminal du transistor auquel le nœud correspondant se connecte. Un rectangle de "polysilicon" adjacent à un transistor se connecte à sa grille, tandis qu'une "diffusion p" forme une source ou un drain de ce transistor.

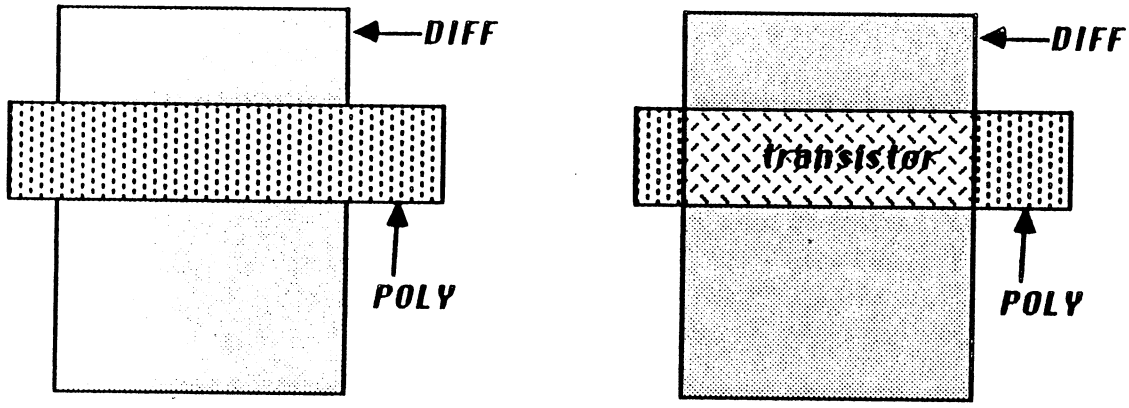


Figure 6 - ***Réécriture de certains rectangles*** *en fonction des couleurs des éléments en superposition; les règles de réécriture sont fournies par le fichier technologique en fonction des couleurs en superposition ; la superposition (DIFF, POLY) se réécrit ainsi dans les couleurs (transistor -DIFF-POLY, DIFF).*

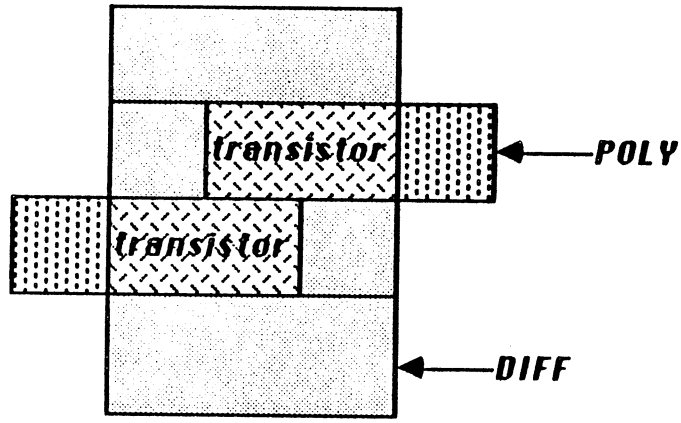


Figure 7 - Le même algorithme utilisé pour trouver les noeuds électriques peut être utilisé pour trouver toutes les parties connexes d'un transistor , ainsi les ***transistors irréguliers*** *peuvent être identifiés en utilisant la table de d'équivalences électriques (régles d'équivalences fournies par le fichier technologique des équivalences électriques).*

4.4. Extraction des capacités entre nœuds

Des capacités de couplage internodales surviennent quand des fils de différents niveaux de masques et appartenant à des nœuds électriques différents se recouvrent, ou quand des fils parallèles de nœuds électriques différents courent bord à bord *(figure 8).*

---

(1) : lorsque l'algorithme Union-find a été utilisé pour reconnaître les nœuds électriques - 2 rectangles équivalents ont alors le même représentant à la racine de l'arbre de F.G. associé . Dans le cas de la technique Depth-first, après reconnaissance des nœuds electriques, deux rectangles équivalents ont le même nom. Il est donc facile de reconnaître dans les 2 méthodes l'équivalence de 2 rectangles quelconques une fois réalisée la reconnaissance des divers nœuds electriques.

Le **calcul des capacités de couplage de recouvrement** est très simple. Pou **chaque rectangle t** du plan de conception il faut :

1. ***rechercher géométriquement les recouvrements avec les autres rectangles;***
2. ***comparer les nœuds de chaque rectangle ainsi trouvés avec le nœud de t. S'ils sont différents, calculer une capacité de couplage entre ces deux nœuds (fonction des niveaux respectifs) des rectangles en interaction et de la surface réelle de recouvrement;***
3. ***pour être correctement calculée, la surface réelle de recouvrement doit être ajustée par la déduction des participations possibles des rectangles du même nœud électrique que t et déjà traités (recouvrements de rectangles).***

Ainsi, pour chaque interaction de rectangles t et t', déduire donc éventuellement la partie de capacité déjà comptée entre des rectangles t" et t', (t" rectangle équipotentiel à t). Le calcul de l'intersection t , t', t" résout complètement la correction. Comme pour les calculs de périmètre, surface et diverses capacités nodales, après la phase de reconnaissance des divers nœuds électriques, **chaque rectangle peut être traité pour le calcul des capacités inter-nodales de son nœ électrique** (lorsque la technique UNION-FIND est utilisée).

Les **capacités de couplage entre côtés parallèles** sont calculées e recherchant d'abord de manière **géométrique** les éléments des quatre **"halos"** (zones d'interaction) **autour de chaque rectangle t**. Un ajustement du calcul de l'interaction de t avec un autre rectangle doit être aussi réalisé par la déduction des capacités internodales déjà comptées pour des rectangles t" équivalents à t et le recouvrant.

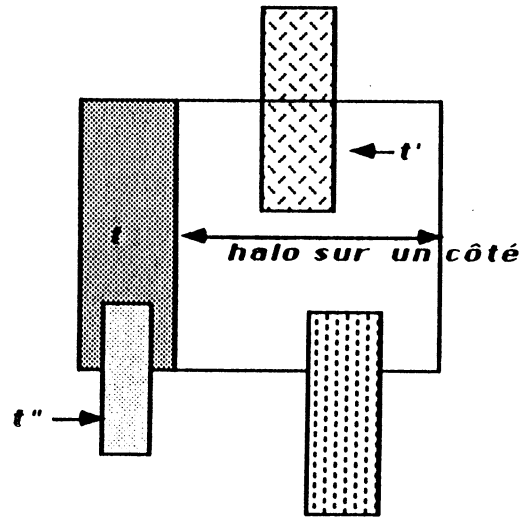


Figure 8 - ***Extraction des capacités de couplage*** - *Comme les capacités entre côtés parallèles diminuent rapidement, la recherche des éléments en interaction avec un côté de (t) est limitée à un halo de quelques lambda . Un ajustement du calcul est réalisé par la reconnaissance (t' < t) et l'évaluation de la capacité internodale déjà sommée (t"< t) pour des éléments (t") équivalents (t).*

Chapitre 4

LE SOUS-SYSTEME DE RECHERCHE GEOMETRIQUE PROPOSE

1. QUADTREES ET STRUCTURES EQUIVALENTES. LEUR PRINCIPE

Les structures de données hiérarchiques sont largement utilisées pour représenter les données dans des domaines aussi divers que l'informatique graphique, le traitement des images, les systèmes de bases de données et informations géographiques. Ces structures de données sont essentiellement basées sur le principe de décomposition récursive **"diviser pour régner"** [AHO 74]; l'une d'elles est appelée **quadtree**. Le terme *quadtree* a un sens générique et diverses représentations des données sont ainsi nommées par les différents domaines qui les utilisent. Ces divers quadtrees représentent alors différents types de données et utilisent des processus de décomposition récursive aussi différents; la décomposition peut être régulière ou gouvernée par les données; la résolution (degré de décomposition) est soit fixe, soit gouverné par les données.

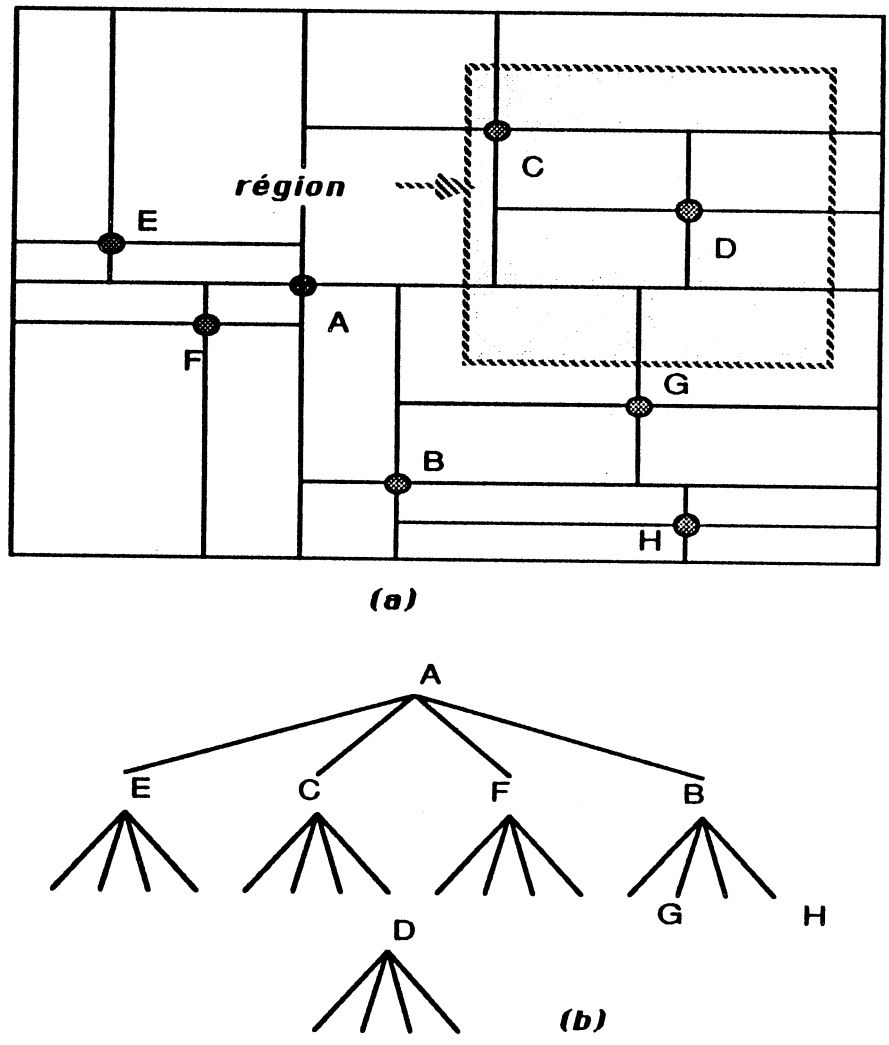


*Figure 1- **Quadtree de points** - Ce quadtree de point (b) est construit par la séquence d'entrée des points : A, B, C, D, E, F, G , H en (a). Un **point** représente ici un **enregistrement** qui a **2 attributs** de sélection, mais ce principe de partition se généralise à **K attributs** ou **dimensions,** avec 2^K branches par nœud. Cette partition des données améliore la sélection des données d'un **domaine** de recherche ou **région** de selection.*

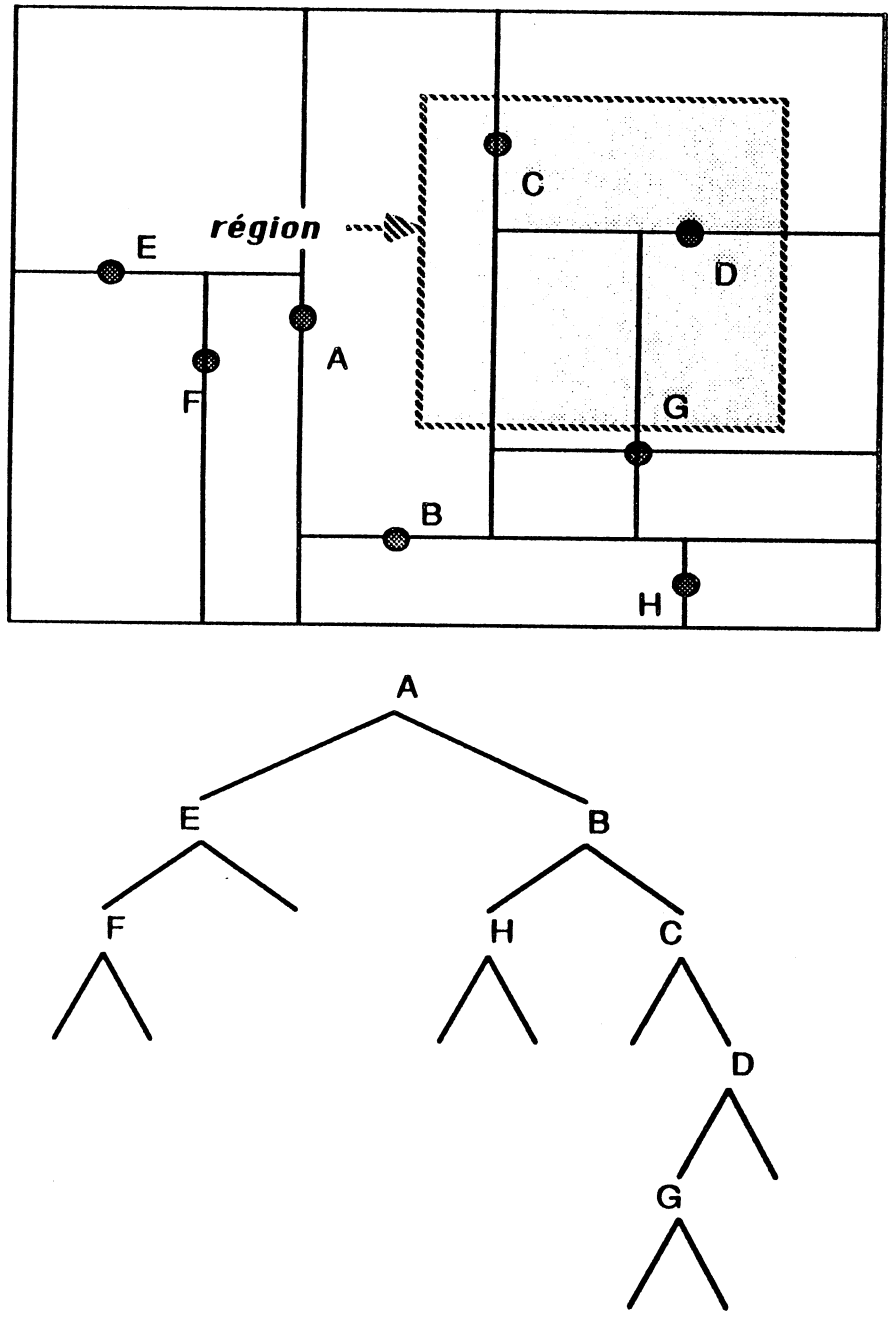


Figure 2 - ***Un k-d tree et les enregistrements qu'il représente*** *- Le 2-d tree , en (b), est construit par la* ***séquence d'entrée*** *des 2-tuples ou points : A, B, C, D, E, F, G en (a). Un* ***point*** *représente ici un enregistrement qui a 2 attributs de sélection, mais ce principe de partition se généralise à K* ***attributs*** *ou* ***dimensions,*** *avec toujours 2 branches par nœud et la selection d'un attribut de parttiion à chaque nœud, ce qui produit un K-D tree. En mode statique, la recherche des points ou enregistrements médians de chaque partition permet de construire des k-D trees équilibrés. En mode dynamique la maintenance de K-D trees équilibrés est difficile [WIL 78]. Cette partition des données améliore la sélection des données d'un* ***domaine*** *de recherche ou* ***région*** *de selection.*

Originellement **quadtree d'image**, le quadtree fut introduit pour représenter des trames d'images [SAM 80 & ROS], [ROS 82]. Ce quadtree représente alors un ensemble de blocs adjacents et multiples (puissance de 4) d'un bloc minimal (résolution de l'image). Ces *quadtrees d'images (region quadtrees)* furent généralisés en **quadtrees de points** (point quadtrees) par [FIN 74 & BEN],

généralisant ainsi les arbres de recherche binaire [KNU 73] pour K attributs ou dimensions. Lorsque K = 2, chaque K-tuple appartenant à l'espace des données peut alors être représenté par un point du plan euclidien. Ainsi les opérations fondamentales réalisées sur un ensemble de données dynamiques [KNU 73], sont-elles réalisables dans le K-espace associé aux données. Les **recherches exactes** *(point queries)* qui déterminent si un k-tuple donné est dans la base de données, ainsi que **l'extraction des k-tuples d'un domaine** *(range query)*, ou **"query de région"**, sont des opérations fondamentales réalisables sur le K-espace associé aux données.

Toutes ces structures hiérarchiques sont très dépendantes de l'ordre d'entrée des données, c'est-à-dire des divers tuples du K-espace associé. Diverses **méthodes** vont donc tenter d'améliorer ces *quadtrees de points.* J.Nievergelt groupe ces méthodes en deux catégories : celles **qui organisent directement les données** et celles **qui organisent** plutôt **l'espace support des données** (régionnement / bucketting), distinguant ainsi les **"trees"** *(premier type)* des **"tries"** *(deuxième type)* appelés généralement **structures ou arbres adaptatifs** *(adaptative trees)* [NIE 84].

1.1. Quadtrees de point et K-D trees

Le **quadtree de points** [FIN 74 & BEN] généralise les arbres de recherche binaire *(figure 1)* ; chaque nœud représente un k-tuple et possède 2^K fils (branches) pour K attributs de sélection (dimension). L'équilibre de cette structure de données est difficile à maintenir tout au long des insertions et suppressions de points (k-tuples) [OVE 82]. Le **K-D tree** [BEN 75] est une amélioration des quadtrees de points destinée à éviter l'important facteur de branchement (2^K branches) ; la recherche binaire est faite dimension par dimension et à chaque point de subdivision (nœud de l'arbre) une nouvelle direction de subdivision est choisie [FRI 77 & BEN] *(figure 2).* Mais la maintenance de K-D trees équilibrés qui supportent les insertions et suppressions d'enregistrements est un problème largement ouvert [BEN 79 & FRI], [WIL 78], [TRO 81].

1.2. Quadtrees adaptatifs et K-D tries

1.2.1. Le K-D trie

Le **K-D trie** est un **K-D tree adaptatif** qui est un régionnement de l'espace des données, plus qu'un arrangement des données en arbre *(figure 3)* . Cependant, cette structure de données construite par une succession de partitions binaires selon une des dimensions, nécessite la connaissance préalable de l'ensemble des données. Les k-tuples (ou points) sont alors stockés aux nœuds terminaux et une correspondance peut être facilement établie entre ces nœuds et les pages d'un fichier, assurant ainsi une répartition géographique de la base de données [MAT 84]. Cependant, ce type de structure de données reste réservé aux **bases de données qui évoluent lentement** et supportent de nombreuses interrogations géographiques (point search, point enclosure, region search) et peu de mises à jour.

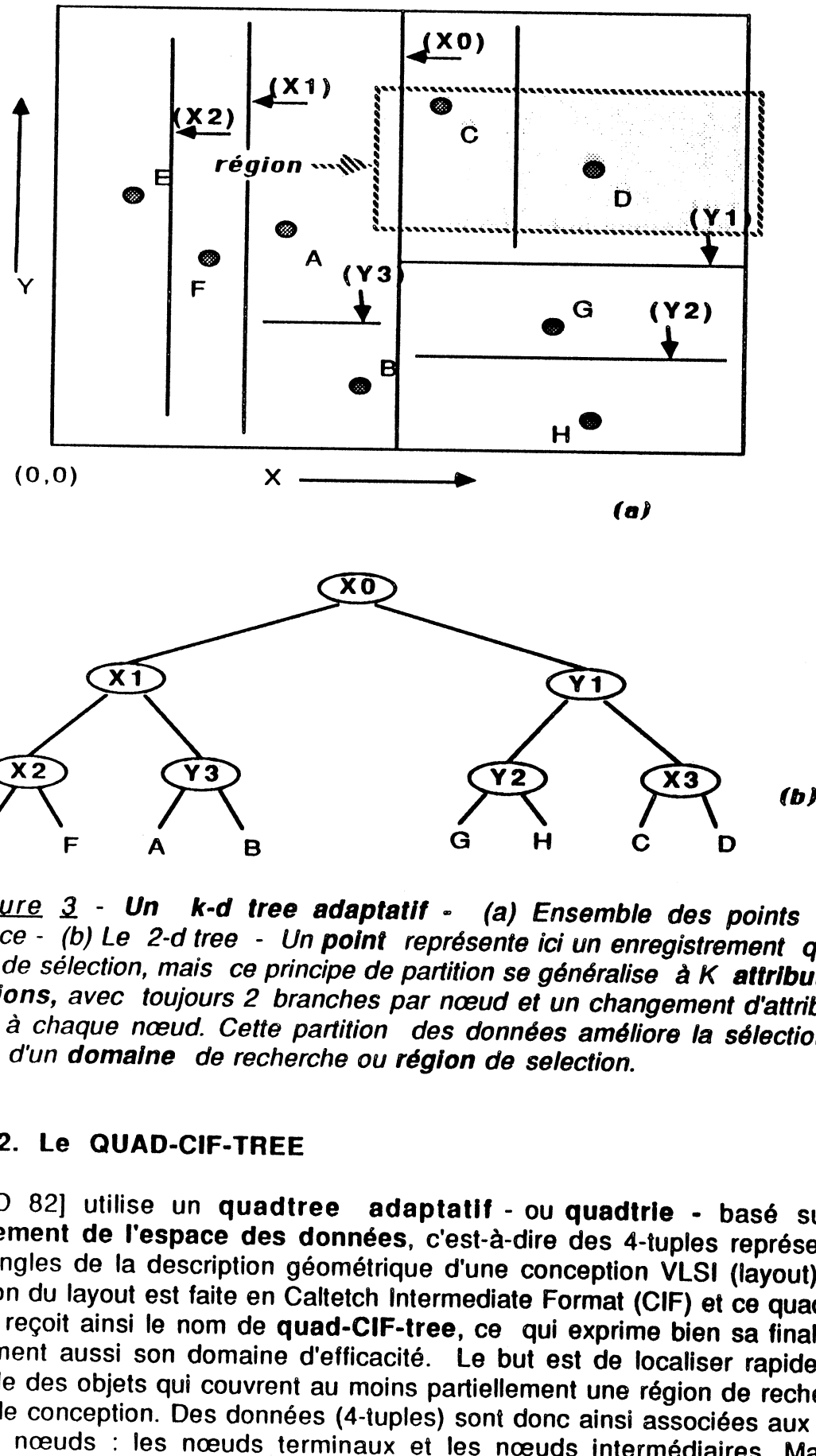
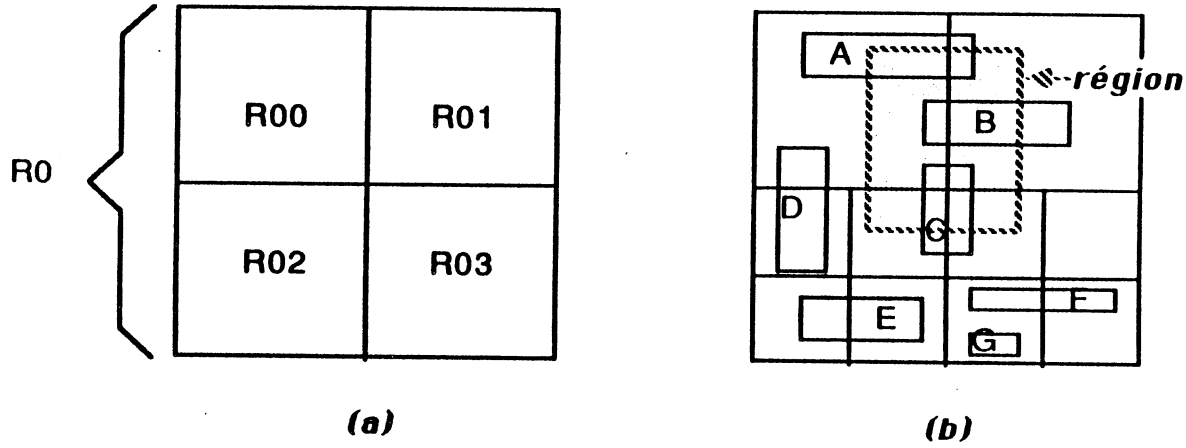


Figure 3 - ***Un k-d tree adaptatif*** - *(a) Ensemble des points d'un 2D-espace - (b) Le 2-d tree - Un* ***point*** *représente ici un enregistrement qui a 2 attributs de sélection, mais ce principe de partition se généralise à K* ***attributs*** *ou* ***dimensions,*** *avec toujours 2 branches par nœud et un changement d'attribut de partition à chaque nœud. Cette partition des données améliore la sélection des données d'un* ***domaine*** *de recherche ou* ***région*** *de selection.*

1.2.2. Le QUAD-CIF-TREE

[KED 82] utilise un **quadtree adaptatif** - ou **quadtrie** - basé sur le **régionnement de l'espace des données**, c'est-à-dire des 4-tuples représentant les rectangles de la description géométrique d'une conception VLSI (layout). La description du layout est faite en Caltetch Intermediate Format (CIF) et ce quad-tree adaptatif reçoit ainsi le nom de **quad-CIF-tree**, ce qui exprime bien sa finalité et implicitement aussi son domaine d'efficacité. Le but est de localiser rapidement l'ensemble des objets qui couvrent au moins partiellement une région de recherche du plan de conception. Des données (4-tuples) sont donc ainsi associées aux deux types de nœuds : les nœuds terminaux et les nœuds intermédiaires. Mais le processus de division des **quad-CIF-trees** est limité par le blocage des 4-tuples (rectangles) dans les nœuds intermédiaires lorsque les rectangles associés couvrent

au moins partiellement les médianes (bissectrices) de partition des objets de ces nœuds. L'organisation des données de ces nœuds par des structures de données secondaires constitue alors un moyen d'améliorer la partition générale des données dans le but de répondre efficacement aux recherches géométriques.



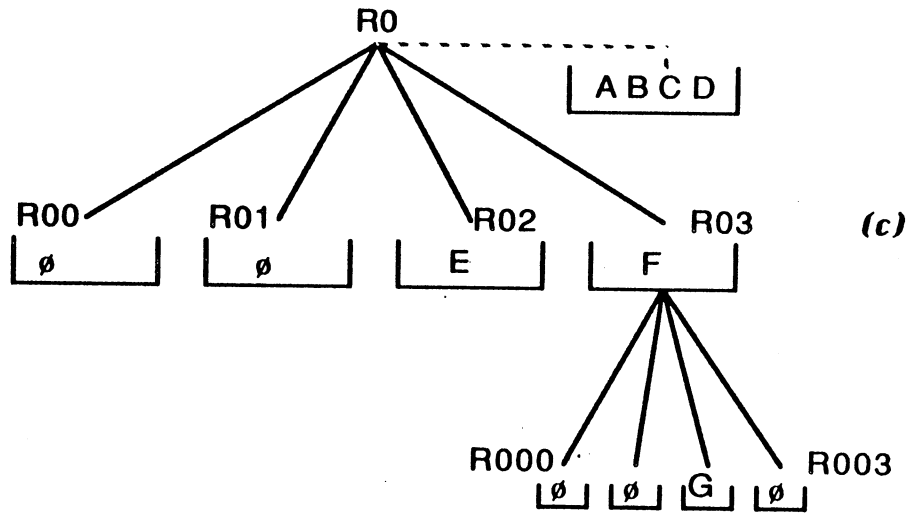


*Figure 4 - **Le Quadtree adaptatif de rectangles** - Le quadtree adaptatif de **rectangles** partage une **collection de rectangles** par subdivisions quaternaires successives jusqu'à la partition minimale - **Principe du partage** : Le quadrant R0 est partagé en 4 sous-quadrants par ses axes médians. Si l'un des rectangles touche l'une des médianes du partage d'un quadrant alors il reste dans le **"panier"** associé à ce quadrant sinon il descend dans l'un des 4 sous-quadrants et est reconsidéré lors du partage de ce quadrant. Un **rectangle** représente ici un enregistrement qui a 4 attributs de sélection ou **dimensions**. Cette partition des données améliore la sélection des données d'un **domaine** de recherche ou **région** de selection.*

[KED 82] a proposé de réitérer le processus de partage récursif sur chaque médiane de partition en considérant les 2-tuples (segments) associés aux rectangles projetés et le même seuil de partition (résolution). [BROW 86] propose de stocker de manière multiple chaque rectangle attaché aux bissectrices d'un nœud dans les quadtrees fils de ce nœud et couvert (partiellement) par le rectangle concerné. [BER 85] propose de différencier et paramétrer indépendamment la résolution quaternaire

et la résolution binaire des originels quad-CIF-trees et d'associer le rectangle bloqué dans un nœud au bissecteur sur lequel il a la plus faible projection (indépendamment des bissecteurs couverts), dans le but d'accélérer les recherches géométriques. Les **mixed-quadtrees adaptatifs** ainsi réalisés (mixe-quadtries) bien qu'apparentés aux quadtrees adaptatifs, en diffèrent profondément ; ils permettent par exemple de réaliser une partition des données avec une **faible résolution quaternaire** (large seuil de partition) et une **résolution binaire plus fine** (faible seuil de partition) et un très bon compromis espace - temps pour la résolution des recherches géométriques demandées par les diverses fonctions de la CAO des VLSI *(chapitre 6).*

2. LE MIXED-QUADTRIE PROPOSE COMME ACCES GEOMETRIQUE

Le mixed-quadtrie appartient à la classe des quadtrees adaptatifs et réalise un régionnement hiérarchique d'une collection de rectangles dans le but de résoudre efficacement la recherche des rectangles couvrant au moins partiellement une région donnée. La description géométrique d'un circuit VLSI peut être composée d'objets beaucoup plus complexes que des rectangles; ces objets peuvent alors être décomposés en objets plus simples ou atomes de géométrie tels que des rectangles ou des polygones simples [CHA 79], [OHT 82], ou être simplement recouverts par de tels atomes [CHAI 81], [YAM 84], [FRA 84]. Ces atomes de géométrie sont alors cherchés de manière géométrique dans la collection par leur **enveloppe rectangulaire** (plus petit rectangle englobant).

2.1. Quadtree adaptatif et quad-CIF-tree

2.1.1. Principe de base

La meilleure manière de décrire un quadtree adaptatif est de regarder comment il est construit *(figure 5).* Supposons que nous partions d'une région rectangulaire contenant la collection des rectangles (ou des polygones simples). Cette région rectangulaire est associée à la racine du quadtree et est divisible par ses médianes (ou bissectrices), en quatre sous-régions égales (quadrants). Chacun des rectangles (polygones) de la collection couvre un ou plusieurs quadrants de la subdivision. Les rectangles (polygones) complètement inclus dans un de ces quadrants sont amenés dans le nœud correspondant (bucket). Les rectangles couvrant (ayant une intersection non vide avec) soit la **médiane horizontale (X-médiane)**, soit la **médiane verticale (Y-médiane)** sont alors en principe mis dans une liste associée à chacune d'elles et stockés ainsi dans le nœud courant. Ainsi, dans un nœud non terminal, tous les rectangles sont amenés, soit dans un nœud fils, soit mis dans une des deux listes associées aux axes médians du nœud courant. Chacun des nœuds fils est à son tour divisé en quatre quadrants et traité de la même manière à moins que la résolution du régionnement soit atteinte. La **résolution de la partition** est définie par la valeur du **seuil de partition quaternaire** qui pourrait être, soit une profondeur maximale (**résolution relative**), soit un **nombre maximum d'objets**, soit une **dimension minimale de subdivision** des rectangles associés aux nœuds (**résolution absolue**).

Originellement, les structures adaptatives utilisent le nombre d'objets stockés dans un nœud comme seuil de partition, mais lorsque la distribution des objets est assez uniforme, à un choix de dimension minimale correspond alors un nombre

d'objets et réciproquement, de sorte que pour une telle distribution des données, les quad-CIF-trees sont très proches des originels **quadtrees adaptatifs** ; lorsque la partition minimale correspond à la largeur des objets, très peu d'objets appartiennent aux nœuds terminaux et les quad-CIF-trees ressemblent alors à des **arbres parfaits (perfect quadtrees)**. Or, les layouts complètement expansés sont constitués de petits objets assez uniformément répartis [BEN 80 & HAK], ce qui assure pour ces layouts l'équivalence des différents modes de résolution. Par contre, les **layouts hiérarchiques** nécessiteront l'implémentation d'un quadtree par modèle et donc le **choix d'un paramétrage absolu de partition** de tous ces quadtrees.

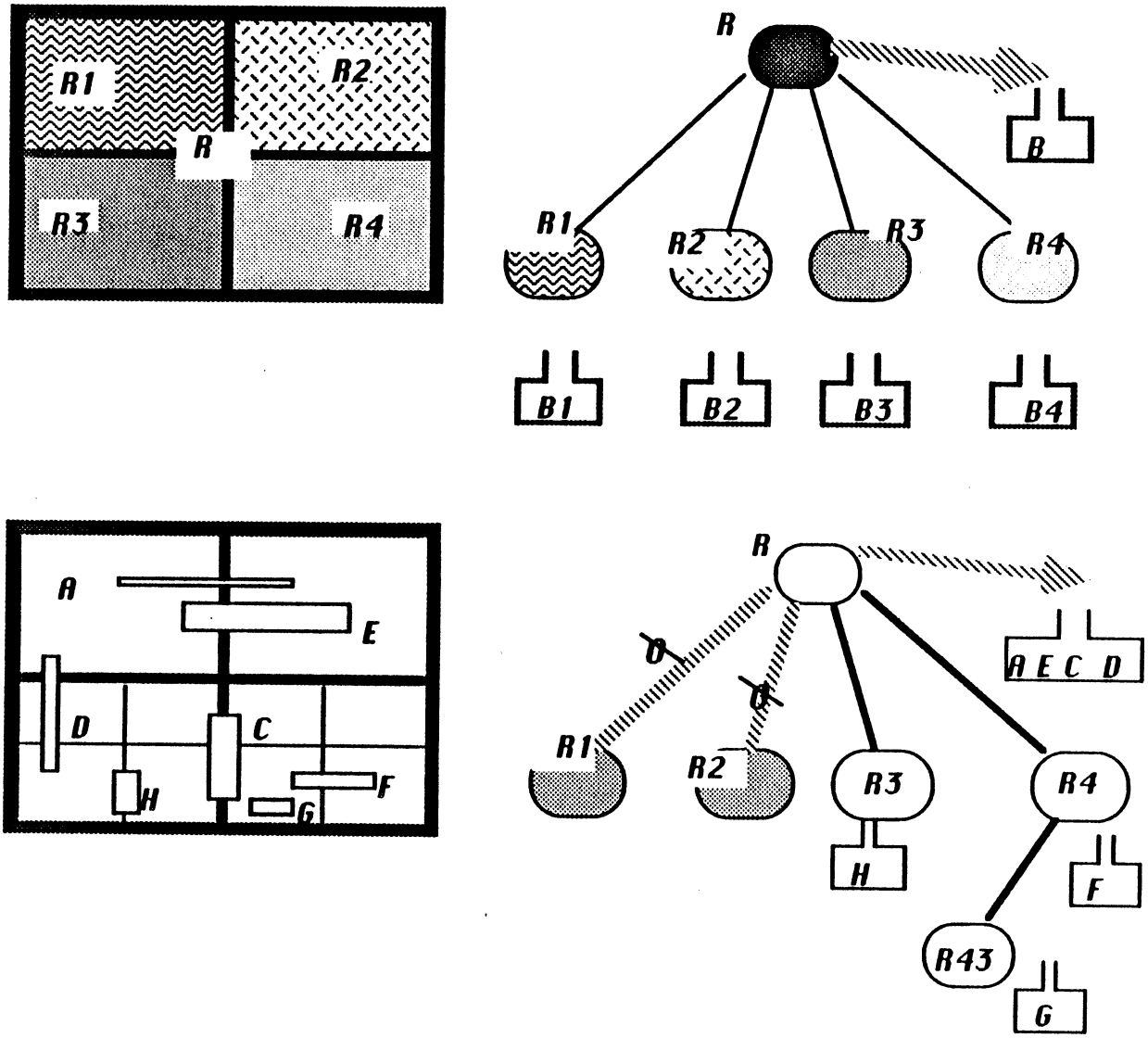
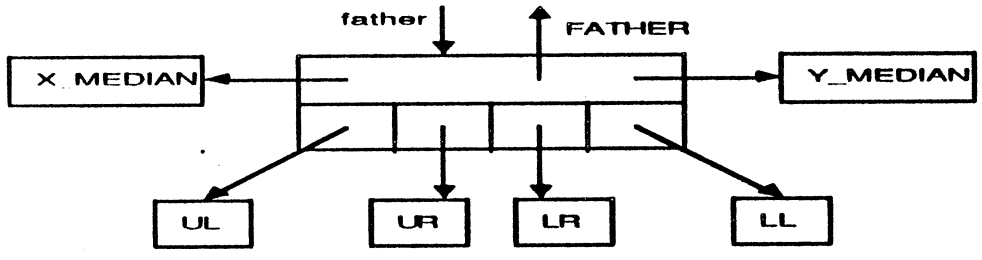


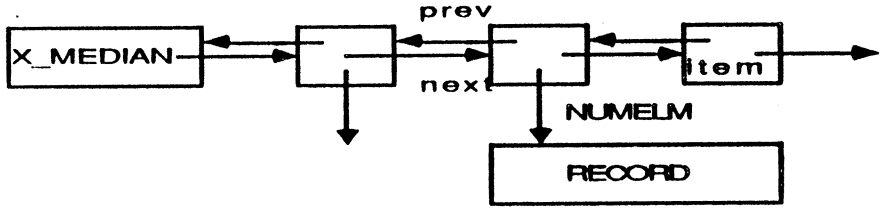
Figure 5 : ***Construction du quadtree d'une collection d'objets*** - *La collection d'objets est partagée en paquets selon le principe suivant : le rectangle R est divisé en 4 sous-rectangles (quadrants) par ses axes médians ; si l'élément A est inclus dans un des 4 sous-rectangles alors il tombe dans la collection d'éléments qui lui est associé sinon il reste attaché au rectangle R. Ce processus de division continue ainsi récursivement jusqu'à ce que les sous-rectangles / quadrants atteignent une taille minimum ou seuil de partition ou qu'il n'y ait plus d'objet à partager; la profondeur maximale possible du quadtree correspond au nombre maximum possible de réitérations du partage (résolution quaternaire); elle est théorique et ne correspond pas à la profondeur locale et réelle .*

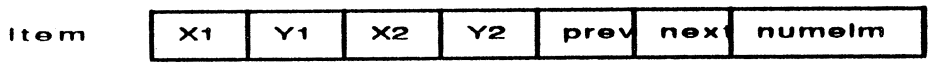
Les quad-CIF-trees supportent mieux les insertions et suppressions d'objets que les originels quadtrees adaptatifs et ces opérations sont moins coûteuses; de plus, le nombre d'objets couvrant les médianes de partition est a priori inconnu et dépend de la taille du rectangle associé au nœud; la gestion de la résolution de la partition par le paramètrage de la partition minimale est d'implémentation fort simple. Une extension de ce principe de partition et de résolution va de plus nous permettre d'implémenter des **quadtries complètement dynamiques** prenant en compte **l'élargissement de la collection** et du rectangle qui l'englobe, ce qui constitue une amélioration importante de cette structure de données *(cf. §2.1.2.)*.

Il y a deux types de nœuds quaternaires : les nœuds feuilles et les nœuds non feuilles (terminaux et non terminaux). Ces deux types de nœuds ont la même structure; ils contiennent deux pointeurs vers les listes associées aux axes médians, un pointeur associé à chacun des fils, un pointeur associé au père, afin de faciliter les suppressions



Ainsi 8 mots sont utilisés pour chaque nœud quaternaire et 7 mots pour un rectangle appartenant aux listes X-médian ou Y-médian.





Chaque objet de la collection appartient à une et une seule des listes X-médian et Y-médian associées à chaque nœud. **Les coordonnées supérieures droites et inférieures gauches du plus petit rectangle englobant, constituent la clef d'accès géométrique de l'objet dans le quadtree** (une troncature particulière évite les ambiguïtés causées par les transformations). Chaque objet est référencé dans le quadtree par son adresse, que l'utilisateur peut définir comme une adresse mémoire primaire ou secondaire (index, etc. ..), ce qui garantit **l'indépendance de la méthode d'accès et des applications qui l'utilisent.**

La collection des objets rangés dans un quadtree est à priori supposée non structurée, contrairement aux algorithmes proposés par [KED 82], en particulier pour la recherche des objets d'une région.

Cette hypothèse n'empêche pas de ranger dans un quadtree des objets structurés, mais elle garantit l'indépendance de l'application par rapport à la

méthode d'accès. Ainsi, par exemple, une collection d'objets rangés dans un quadtree peut être une cellule utilisée dans un circuit VLSI; **pour connaître les éléments appartenant à une région** de cette cellule, on utilise alors l'action **SELECT-QD** qui sélectionne les objets de la cellule appartenant à cette région; **pour les objets sélectionnés et structurés, il sera nécessaire de descendre hiérarchiquement dans leur description pour connaître les objets primaires appartenant à la région de sélection** (avec une transformation de la région et un changement de quadtree). Ainsi, l'algorithme de recherche géométrique *"travaille en largeur"* dans la hiérarchie de conception, et n'a pas besoin de connaître le type d'objet qu'il manipule ni la hiérarchie de conception.

Les pointeurs des différents noeuds et éléments ont été indifférenciés dans l'implémentation de cette structure de données de manière à pouvoir les redéfinir facilement par simple redéfinition d'un type d' adresse. Ces quadtrees peuvent ainsi être gérés facilement soit par allocation dynamique, soit dans des tableaux, soit comme une relation de la base de données (les nœuds et éléments étant des tuples), soit dans un mélange de ces modes. Actuellement deux de ces modes ont été effectivement implémentés pour des applications différentes; l'implémentation en tableaux est utilisée pour les ERC et DRC conventionnels [LEC 84], [BER 85]; l'implémentation par allocation dynamique en mémoire est utilisée dans le nouvel éditeur multifenêtres qui interface la base de données des circuits (éditeur COSMIC du projet CVS en cours de développement [JUL 86]).

Intérêt de cette technique de partition des données pour les conceptions structurées

Puisque la plupart des **rectangles** qui composent les masques de CI tendent à être **petits et uniformément distribués** [BEN 80 & HAK] la plupart des rectangles seront aux feuilles du quadtree. Ainsi la **profondeur espérée de l'arbre est O(logN)**. Pour un **quadtree adaptatif de type perfect-tree** il y a un élément par feuille, donc à peu prés **N feuilles** et donc pour le quadtree à peu prés N nœuds terminaux et N nœuds non terminaux. Les **perfect quadtrees constituent une borne supérieure de coût espace mémoire** puisque les **quadtrees adaptatifs à plus large seuil de partition sont** bien sûr **moins coûteux**.

La technique de partition des données par ce type de quadtree est intéressante en ce sens qu'elle prend avantage de la **localité de l'information** : les quadtrees se développeront uniquement là où il y a beaucoup de petits rectangles. Or notre objectif est de pouvoir utiliser les quadtrees sur des hiérarchies de rectangles (layout structurés de VLSI). Dans une **cellule structurée de VLSI il y a beaucoup de petits rectangles là où il n'y a pas de hiérarchie** - et la distribution de ces rectangles est alors celle des layouts non structurés - **et peu de gros rectangles peu superposés là où il y a des sous-cellules** (instances de symboles). Ces cellules structurées **ne doivent donc pas produire pour ces quadtrees adaptatifs des comportements de pire cas si les layouts sans hiérarchie ne le faisaient pas**.

Si les layouts de CI (structurés ou non) ne produisent pas de pire cas aux quadtrees adaptatifs il est cependant facile de le faire avec une collection de long objets couvrant tous les médianes du rectangle englobant la collection ou avec une collection d'objets localement tous en superposition. Dans un layout de VLSI le

facteur de superposition est toujours très faible car les concepteurs n'empilent pas les motifs de manière quelconque, les superpositions accroissent d'ailleurs considérablement la complexité de l'analyse de ces descriptions géométriques.

Cependant beaucoup de rectangles peuvent rester dans les nœuds supérieurs du quadtree, ce sont les rectangles qui couvrent (au moins partiellement) une des médianes de partition. Ainsi **si une fenêtre de selection couvre partiellement au moins l'une des médianes de division de l'un des nœuds supérieurs, tous les objets qui sont restés bloqués dans le nœud doivent être contrôlés** (même les objets assez loin de la fenêtre) *(§2.2.)*

2.1.2. La gestion dynamique des quadtrees

L'algorithme de mise à jour des quadtrees permet **l'agrandissement de la collection d'objets en nombre et en surface occupée.** Le principe de l'algorithme de mise à jour est donné ci-dessous.

action RANGER_ITEM (var RACINE , var CADRE, ITEM);
{entrée, sortie: RAC et CADRE : racines et rectangle associé du quadtree
entrée : ITEM : l'élément à ranger dans le quadtree
cette action range un élément dans le quadtree}

1. si RACINE = nil alors CREER_NOEUD (RACINE);
2. si bbox (ITEM) non inclus dans CADRE
   alors AGRANDIR_QD (RACINE, CADRE);
3. placer_item (RACINE, CADRE);

L'étape 1 alloue une racine si le quadtree n'existe pas
L'étape 2 **agrandit le cadre enveloppe du quadtree** s'il est trop petit pour recevoir l'élément et alloue successivement les différents nœuds du chemin d'agrandissement *(figure 6)*.
L'étape 3 **place alors l'objet dans le quadtree** selon le processus décrit dans la *figure 5 (§1.1.1)*.
Sur la *figure 6* les secteurs angulaires de sommet un sommet du rectangle RECT et contenant ce rectangle définissent les zone1, zone2, zone3, zone4. De même on peut définir quatre quadrants multiples du rectangle RECT, soit RECT1, RECT2, RECT3, RECT4.
Sont actives toutes les zones qui ont une intersection non vide avec le rectangle ITEM *(E1 ou E2)*. On ne peut élargir le quadtree que dans une zone active. Comme les diverses zones i ont des intersections non vides la priorité de recherche d'intersection de zone i, i: 1..4, avec ITEM sera l'ordre des i croissants.

Le **CADRE du quadtree est ainsi agrandi soit par l'un des quadrants RECT i , soit dans l'une des zones i par un multiple de 4.** Le cas où l'ITEM est inclus dans un des quadrants ainsi générés constitue le but atteint, sinon le processus récurse. Si la solution obtenue n'est pas optimale cela n'affecte pas cependant les performances des recherches géométriques. Des collections entières de rectangles génèrent ainsi de nombreux agrandissements dynamiques. Les éditeurs de cellules utilisent avec profit les agrandissements dynamiques de surface sans détérioration sensible des performances. L'algorithme d'agrandissement est donné dans *l'annexe 1.§.3.1.*

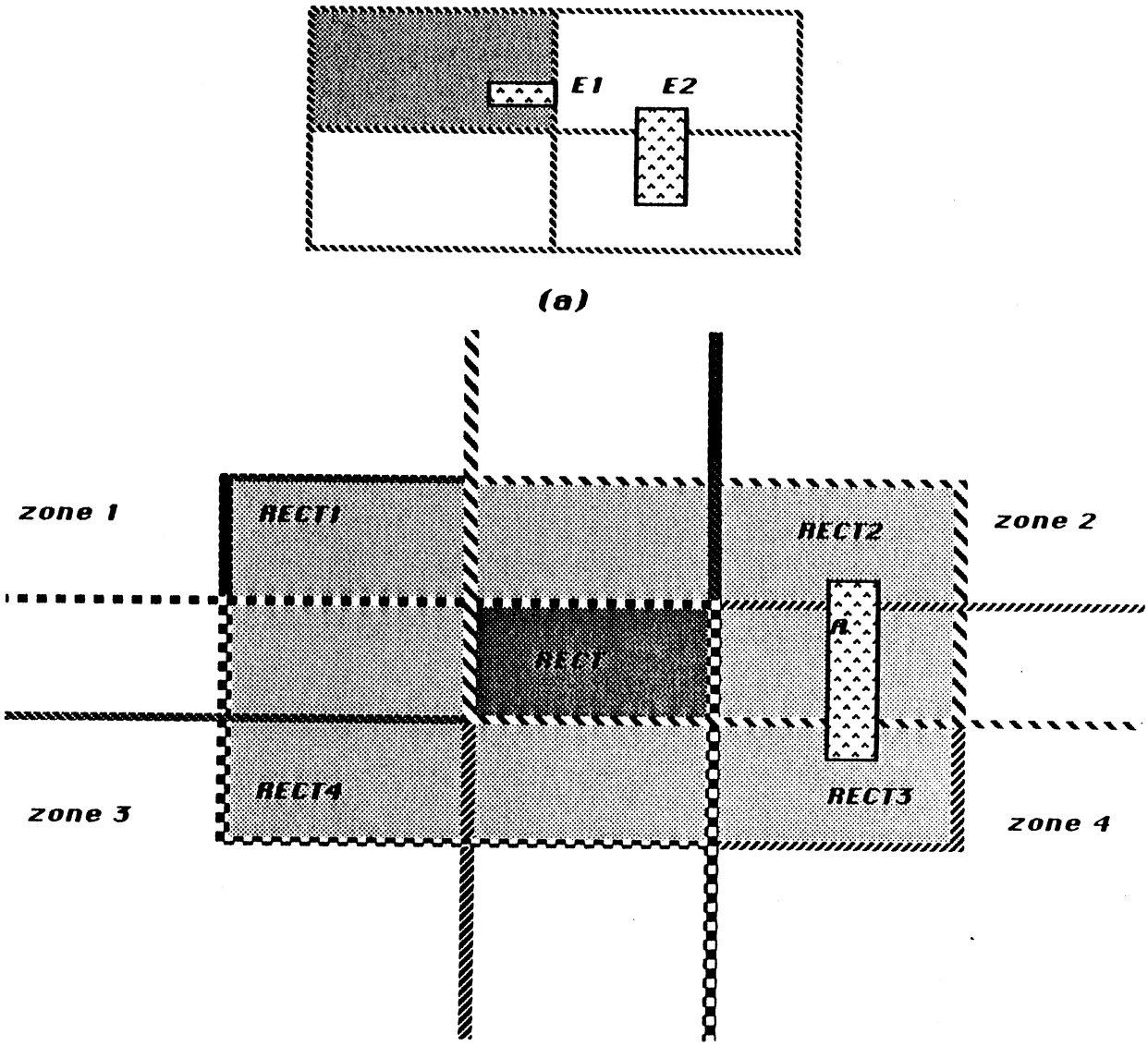


Figure 6 : **La gestion dynamique des données et l'agrandissement des quadtrees** - **(b)** - *L'algorithme de rangement d'un objet dans un quadtree -1. alloue eventuellement une racine - 2.* ***agrandit le cadre enveloppe du quadtree*** *s'il est trop petit pour recevoir l'élément et alloue successivement les différents nœuds du chemin d'agrandissement - 3* ***place alors l'objet dans le quadtree selon le processus décrit dans la figure 5*** *(§1.1.1). Les secteurs angulaires de sommet un sommet du rectangle RECT et contenant ce rectangle définissent les zone1, zone2, zone3, zone4. De même on peut définir quatre quadrants multiples du rectangle RECT, soit RECT1, RECT2, RECT3, RECT4. Sont actives toutes les zones qui ont une intersection non vide avec le rectangle ITEM (E1 ou E2). On ne peut élargir le quadtree que dans une zone active. Comme les diverses zones i ont des intersections non vides la priorité de recherche d'intersection de zone i, i: 1..4, avec ITEM sera l'ordre des i croissants.Le* ***CADRE du quadtree est ainsi agrandi soit par l'un des quadrants RECT I , soit dans l'une des zones I par un muliple de 4****. Le cas où l'ITEM est inclus dans un des quadrants ainsi générés constitue le but atteint, sinon le processus récurse. Si la solution obtenue n'est pas optimale cela n'affecte pas cependant les performances des recherches géométriques. Des collections entières de rectangles génèrent ainsi de nombreux agrandissements dynamiques. Les éditeurs de cellules utilisent avec profit les agrandissements dynamiques de surface sans détérioration sensible des performances. L'algorithme d'agrandissement est donné dans l'annexe 1.§.3.1.- (a) - Le quadrant associé à un noeud du quadtree est un rectangle ouvert, c'est-à-dire que sa frontière est considée comme appartenant aux quadrants des nœuds ascendants; ainsi dés qu'un élément touche la frontière de ce rectangle comme* ***E1*** *le quadtree s'agrandit.*

Le processus d'**agrandissement** est appelé **dès qu'un élément touche le bord** c rectangle associé à la racine du quadtree. Ainsi chaque **rectangle** associé à un nœud est **ouvert et son contour appartient à ses ancêtres**. Ce choi d'implémentation évite les ambiguitës d'appartenance d'un segment appartenant aux médianes de partage ou d'un élément touchant seulement ces médianes.

2.1.3. Algorithmes généraux des quadtrees

Algorithme d'insertion d'un élément dans un quadtree

Il suffit de savoir **insérer un élément dans un sous-quadtree** :

action INSERER_SUBQD (RACINE, RECT, ITEM, TRESH)
{cette action insère un élément dans un quadtree ou un sous-quadtree
entrée : RACINE et RECT : racine et rectangle associés;
ITEM : élément inséré; TRESH : seuil de partition des rectangles}

1. si le seuil de partition n'est pas atteint
1.1. si l'élément appartient à un des 4 sous-rectangles associés au noeud courant ; créer le nœud fils correspondant au rectangle englobant l'élément s'il n'existe pas; récurser avec le nœud fils correspondant et son rectangle associé.
1.2. sinon mettre l'élément dans le nœud courant RACINE.
2. sinon mettre l'élément dans le nœud courant RACINE.

Il existe comme action duale une action de suppression d'un élément d'un quadtree; afin de pouvoir aussi supprimer tous les noeuds associés au chemin d'insertion lorsqu'ils sont vides il a fallu rajouter aux nœuds un pointeur sur leur père.

Algorithme de suppression d'un élément dans un quadtree

action SUPR_ITEM_SUBQD (var RACINE, RECT, ITEM, TRESH)
{cette action supprime un élément dans un quadtree ou un sous-quadtree
entrée : RACINE et RECT : racine et rectangle associés;
ITEM : élément inséré; TRESH : seuil de partition des rectangles}
{sortie : racine du sous quadtree : à nil si le sous quadtree est supprimé}

1. si le seuil de partition n'est pas atteint : chercher si l'élément appartient à un des 4 sous-rectangles associés au noeud courant; récurser avec le nœud fils correspondant et son rectangle associé.
2. sinon chercher l'élément dans le nœud courant RACINE;
3. supprimer la RACINE si elle est vide et récurser la suppression de noeuds sur ses ancêtres...;

L'annexe 1 fournit plus de détails d'implémentation de ces algorithmes.

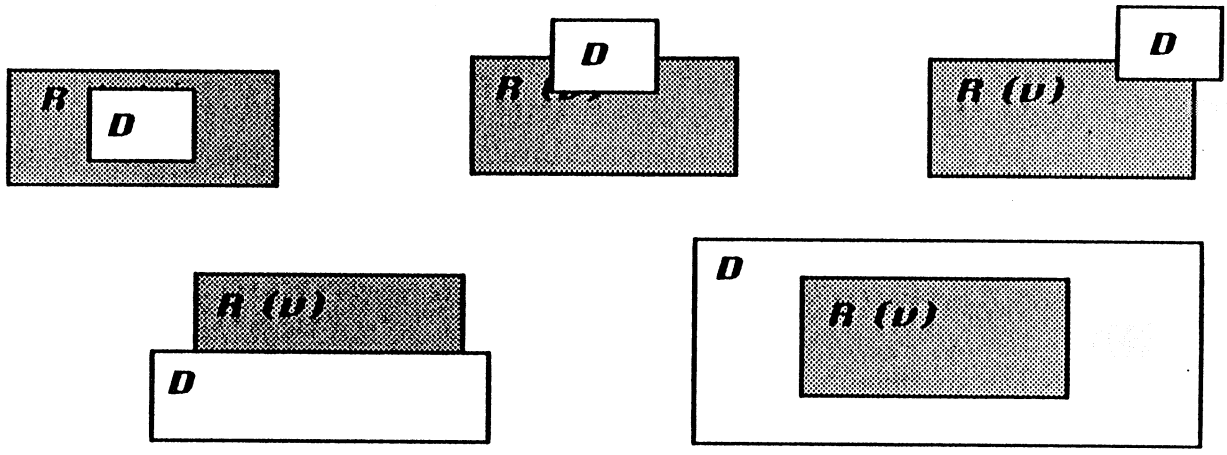


Figure 7 : **Différents types d'intersections du domaine de recherche D avec le domaine R (v) associé a un ensemble de données** (noeud d'un quadtree).

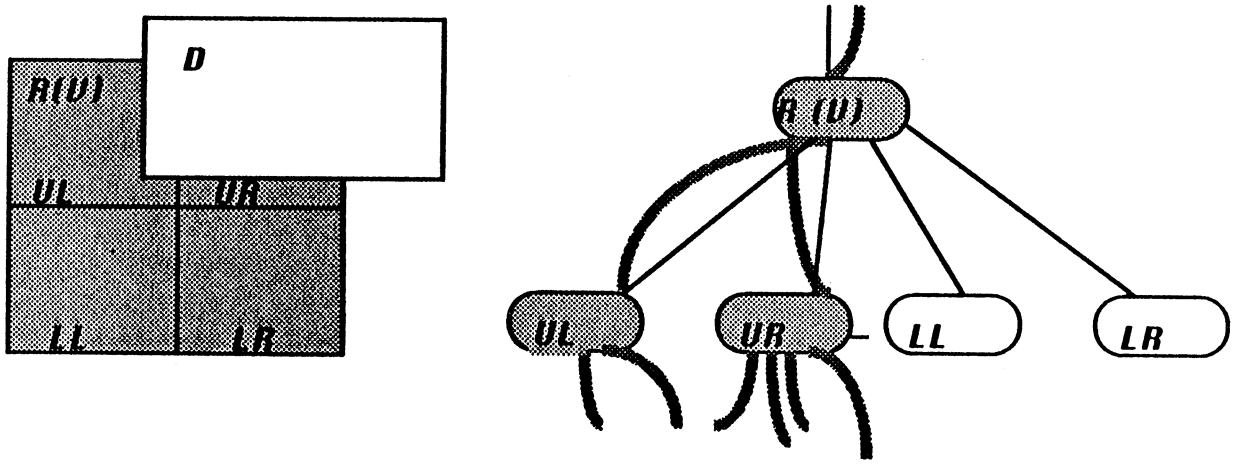


Figure 8 : **Recherche géométrique dans un quadtree de domaine de recherche D sur l'ensemble R(v) des données associées a un noeud.** Les données associées aux noeuds UL et UR doivent être inspectées.

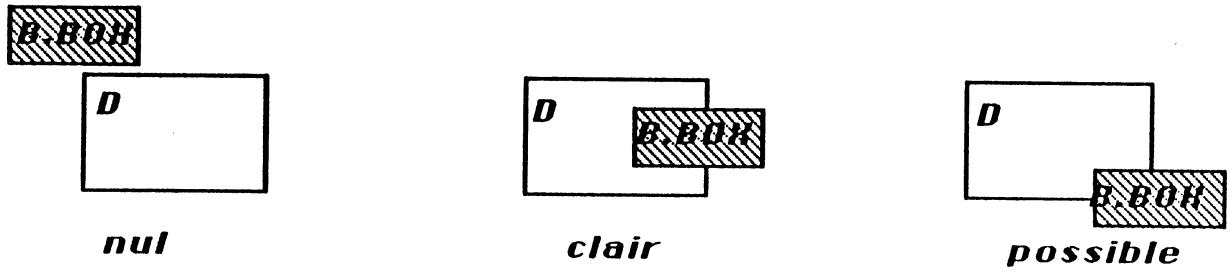


Figure 9 : **Intersection d'un domaine de recherche D et d'un objet E , d'enveloppe BBOX, rangé dans un noeud du quadtree** : 3 résultats possibles pour l'intersection : **nulle, claire , possible** d'après les différents types d'intersections du domaine D avec le rectangle enveloppe B.BOX de l'objet E.

Algorithme de recherche géométrique dans un quadtree

action VISITER_SUBQD (RACINE, RECT, REGION, var L_OBJECT)
{cette action visite un quadtree ou un sous-quadtree
entrée : RACINE et RECT : racine et rectangle associés;
sortie : L_OBJECT : liste des objets trouvés couvrant au moins partiellement la REGION de recherche.}

- si le sous-arbre n'est pas vide (RACINE <> nil)
1. tester si RECT est inclus dans REGION et dans ce cas énumérer dans L_OBJECT tous les éléments du sous-quadtree
2 - SINON si l'intersection de RECT et de REGION n'est pas vide :
2.1. visiter le nœud courant et **sortir les éléments de ce nœud couvrant au moins partiellement la REGION;**
2.2. visiter aussi les quadtrees fils couverts au moins partiellement par REGION en récursion.

La *figure 7* illustre tous les types d'intersection possibles du domaine de recherche D, équivalent au paramètre REGION de l'algorithme ci-dessus, avec un des rectangles rangé dans le quadtree. Le rectangle ***R (v)*** **couvre** *(au moins partiellement)* la région D c'est-à-dire qu'ils ont une intersection non vide dans les 4 cas et les différents types d'intersection (inclusion ou intersections..) ne sont à priori pas distingués bien qu'ils pourraient l'être. Cependant comme il est remarqué en *figure 9,* si l'élément n'est pas rectangulaire il est des cas où sa propre intersection avec la région de recherche correspond bien à l'intersection de son **rectangle enveloppe** *(B.BOX);* **l'intersection** est alors **claire**; sinon elle est **possible** ou **nulle**; les 2 types d'intersections *claire* et ***possible*** sont donc précisés par l'algorithme de *recherche géométrique* qui devient ainsi un filtre pour des éléments non rectangulaires. Pour de nombreuses applications ce filtre est suffisant et l'interprétation des éléments retournés relève des applications appelantes.

La *figure 10* illustre l'action **VISITER_QD** et ainsi le **principe de la recherche géométrique** dans un quadtree.

2.2. Amélioration de la partition quaternaire : les mixed-quadtries

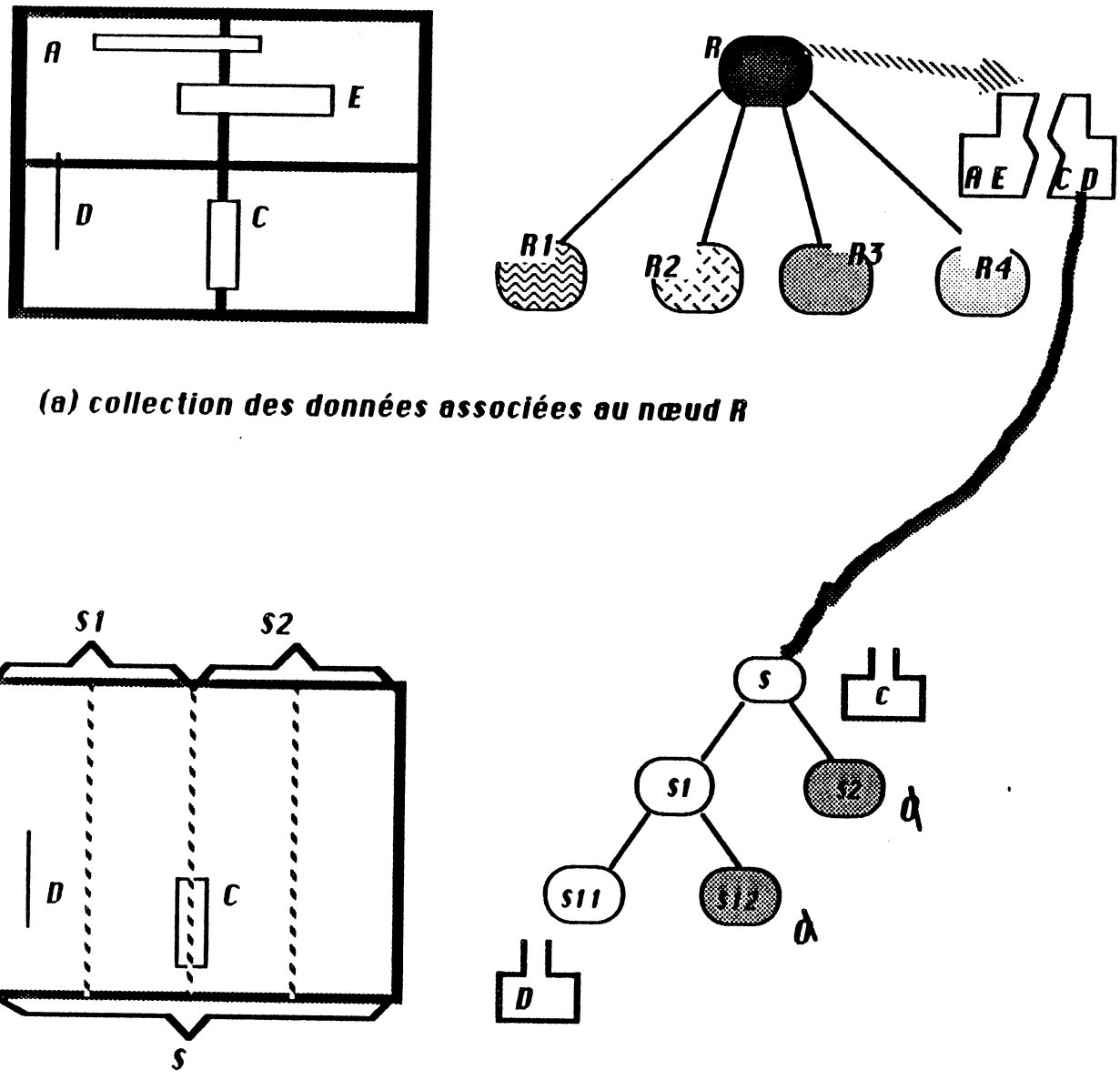
Afin de diminuer le nombre d'éléments ainsi examinés dans ces nœuds quaternaires l'implémentation de structures secondaires s'impose. La collection des rectangles associés à un nœud est ainsi partagée entre les deux listes (X-médian et Y-médian) et **chaque rectangle est associé à la médiane selon laquelle il a la plus faible projection, indépendamment de la médiane qu'il couvre** (partiellement). De plus, **les listes X-médian et Y-médian sont organisées comme des arbres binaires adaptatifs** par régionnement des projections des objets qui les composent et selon le principe de régionnement des quadtries appliqué à une dimension *(figure 10);* ce principe de régionnement d'intervalles est d'ailleurs proche de celui des interval-trees [EDE 80] ; la structure quadtree choisie comme structure primaire de régionnement des données permet de plus de prendre en compte la dynamique d'une **base de données à évolution rapide** tout au long du processus de conception d'une cellule VLSI.

La **distinction des paramètres de résolution binaire et quaternaire**, c'est-à-dire des seuils de partage des sous-collections d'objets (buckets) permet de trouver pour chaque application le **meilleur compromis** *espace mémoire - temps de recherche* en adaptant éventuellement ces paramètres à la nature des recherches géométriques. Chaque problème utilise une certaine aire moyenne pour les régions de sélection qui lui est propre; les aires correspondent rarement à de simples *"pics"* (point searchs). Il s'agit de trouver une bonne structure pour les régions de sélection d'aire non négligeable aussi bien que pour les *"pics"*. Ainsi conçus, les **mixed-quadtrees** semblent réaliser convenablement ce but. De la classe des quad-trees adaptatifs et dérivés des quad-cif-trees, les mixed-quadtrees vont produire des recherches géométriques beaucoup plus efficaces pour une occupation moindre de l'espace mémoire *(cf. §2.3. & 4).*

L'**insertion d'un élément dans un nœud quaternaire** est ainsi réalisée par **l'insertion de l'élément dans l'un des 2 arbres binaires associés au nœud quaternaire**. Le choix de l'arbre binaire correspond à celui associé à la médiane sur lequel l'élément a la plus faible projection de manière à éviter le blocage de l'élément dans les nœuds supérieurs et à produire ainsi une meilleure séparation et localisation des données. Cette méthode doit réduire le taux d'échec d'une recherche géométrique (rapport entre le nombre d'éléments examinés et le nombre d'éléments trouvés).

L'**insertion d'un élément dans un arbre binaire** est réalisée de la même façon que dans un arbre quaternaire par partage récursif de la médiane correspondante.

action INSERER_SUBTREE (RACINE, **SEG,** ITEM, TRESH)
{cette action insère un élément dans un sous-arbre binaire adaptatif
entrée : RACINE et **SEG** : racine et **segment** associés; ITEM : élément inséré; TRESH : seuils de partition quaternaire et binaire}
1. si le seuil de **partition binaire** n'est pas atteint :
1.1. si l'élément appartient à un des **2 sous-segments** associés au noeud courant ; créer le nœud fils correspondant au sous-segment s'il n'existe pas; récurser avec le nœud fils correspondant et son segment associé.
1.2. sinon mettre l'élément dans le nœud courant RACINE.
2. sinon mettre l'élément dans le nœud courant RACINE.



(a) collection des données associées au nœud R

(b) partage du sous-ensemble X-median des données associées au nœud R

Figure 10 : **Amélioration de la partition quaternaire par la gestion de structures secondaires.** *Une segmentation particulière des données va améliorer les temps de recherche géométrique :*

*1. Les éléments d'un noeud quaternaire sont d'abord partagés entre 2 sous-ensembles , **X-median et Y-median.** Les éléments qui ont leur plus faible projection sur l'axe des X appartiennent au sous-ensemble X-median, les autres appartiennent au sous-ensemble Y-median.*

2. Les éléments des sous-ensembles X-median et Y-median subissent une partition secondaire binaire.

-(a) - La collection des éléments A, E, C, D associée au nœud quaternaire R. le bucket associé au nœud R est cassé en deux sous-ensembles X-médian et y_médian.

(b) - Le sous-ensemble X-médian subit alors une partition binaire et les éléments de cette liste sont rangés dans un arbre binaire. La résolution du partage binaire des données est indépendante de la résolution quaternaire.

Ainsi l'action **VISITER un noeud quaternaire** lors d'une recherche géométrique revient à **visiter ses 2 arbres binaires associes**.

L'action **visiter un arbre binaire** est réalisée par l'examen des rectangles rangés dans le nœud racine et sa récursion éventuelle aux sous-arbres associés à la partition binaire de ce nœud (si leur segment associé est couvert au moins partiellement par la projection de la REGION de recherche). *L'annexe 1 donne les algorithmes complets de ces recherches géométriques.*

2.3. Quadtrees de points et de rectangles, résultats théoriques

Des analyses empiriques montrent que le temps moyen d'insertion des **quadtrees de points** est 0 (log N) pour un quadtree de N points [FIN 74 & BEN]. La construction du quadtree est de 0(N log N). La longueur totale du chemin (TPL : Total Path Length) est proportionnelle à NlogN. Si le quadtree de points dégénère cependant les pires cas d'insertion $0(N^2)$ et de recherche peuvent être observés. La recherche sous un point est proportionnelle au rapport : TPL / nombre de nœuds.

Les **quadtrees adaptifs de rectangles** héritent des coûts des quadtrees de points pour l'accès aux listes de rectangles, classes de la partition des données. Mais un inconvénient majeur des simples quadtries de rectangles est la nécessité pour les listes X-médian et Y-médian de stocker tous les rectangles non entièrement contenus dans un quadrant et de devoir complètement examiner ces listes lors de la recherche géométrique des éléments d'une région quand la région de sélection couvre le rectangle associé au nœud stockant ces listes.

Les **mixed-quadtrees** constituent en principe une solution dynamique au problème des listes X-médian et Y-médian et ainsi une amélioration des simples quadtries. Par l'indépendance des résolutions ou seuils de partage quaternaire et binaire, on cherche à obtenir une structure de données dynamique qui ait les avantages des quadtrees pour les *larges recherches géométriques* et des recherches binaires pour les **pics** et les *régions d'aire presque nulle* sur des ***collections de petits objets uniformément distribués*** (comme les atomes de géométrie des conceptions VLSI).

Evaluation théorique des quadtrees adaptatifs sur cette collection pour les recherches géométriques

Un nœud d'un quadtree est visité par l'action VISITER_SUBQD si et seulement si son rectangle englobant a, avec la REGION de recherche, une intersection non vide. Ainsi pour déterminer combien de nœuds sont visités durant une recherche il suffit de déterminer combien de nœuds sur chaque niveau (i) ont un rectangle englobant / quadrant qui couvre (au moins partiellement) la REGION de recherche et ensuite de sommer ces nombres pour tous les niveaux (i).

Nous allons maintenant utiliser cette technique pour calculer combien de nœuds quaternaires sont visités lors de la recherche des éléments de REGION. Soit V (i, x, y) le nombre de rectangles / quadrants de niveau (i) qui recouvrent la REGION rectangulaire (x * y); ceci correspond exactement au nombre de nœuds de niveau (i) visités par VISITER_SUBQD.

Tous les rectangles englobants ou quadrants des nœuds de niveau (i) forment un tableau régulier avec 4^i rectangles (2^i sur chacun des axes). Le nombre estimé de rectangles couverts (au moins partiellement) par REGION sur l'axe des X est entre x 2^i et x 2^i+1. Le nombre de rectangles couverts par REGION est donc approché par le produit $(x2^i+1)(y2^i+1)$; cette quantité que nous notons V (i, x, y) fournit une bonne estimation (et une borne supérieure) pour le nombre de nœuds visités sur le niveau (i) par VISITER_SUBQD quand on fait une recherche géométrique dans un quadtree complet.

Dans le but de déterminer le nombre total de nœuds examinés durant une recherche géométrique nous avons seulement besoin de sommer V (i, x, y) sur tous les niveaux i, $0 \le i \le m-1$. Soit E (m, x, y) le nombre d'éléments visités quand on cherche dans un arbre de m niveaux. Soit V_LIST (i) le nombre d'éléments coupant un des axes médians d'un nœud de ce niveau (i) pour $0 \le i \le m-2$ et et le nombre d'éléments des nœuds terminaux pour i = m-1. On prend pour unité (1*1) le rectangle de plus haut niveau. Soit n le nombre d'éléments. Les lettres grecques sont des constantes.

On a

$$E(m, x, y) = \sum_{0 \le i \le m-1} V(i, x, y) * V\_LIST(i) \quad \text{avec}$$

$$V(i, x, y) = (x2^i+1)(y2^i+1) = xy\,4^i + (x+y) * 2^i + 1$$

$V\_LIST(i) = 2^{-i} * \alpha\sqrt{n}$ pour $0 \le i \le m-2$, α est une constante, $\alpha\sqrt{n}$ nombre d'éléments du nœud racine

$V\_LIST(i) = \beta$ pour $i = m-1$; β est une constante et le seuil de partage quaternaire

$$E(m, x, y) = xy * \sum_{0 \le i \le m-1} 4^i * V\_LIST(i) + \sum_{0 \le i \le m-1} (x+y) * 2^i * V\_LIST(i) + \sum_{0 \le i \le m-1} V\_LIST(i)$$

$$= n*xy + \sum_{0 \le i \le m-2} (x+y) * \alpha\sqrt{n} + (x+y) * \beta * 2^m + \sum_{0 \le i \le m-1} V\_LIST(i)$$

$$= n * xy + (x+y) * \alpha\sqrt{n} * (m-1) + (x+y) * \beta * 2^m + \sum_{0 \le i \le m-1} V\_LIST(i)$$

$$= n * xy + (x+y) * \alpha\sqrt{n} * (m-1) + (x+y) * \beta * 2^m + (1-2^{-m+1}) * 2\alpha\sqrt{n} + \beta$$

$$= n * xy + (x+y) * \alpha\sqrt{n} * ((\log_4(\tau n / \beta) - 1) + (x+y) * \beta * 2^m; + (1-2^{-m+1}) * 2\alpha\sqrt{n} + \beta$$

$$= n * xy + (x+y) * (\lambda\sqrt{n}\log n + \mu) + \gamma\sqrt{n} + \beta$$

$\lambda, \beta, \gamma, \mu$ constantes

Ainsi pour des **recherches sous des points** : $E(m, 0, 0) = \gamma\sqrt{n} + \beta$

Dans le cas général d'une **région de recherche d'aire non nulle le coût ajouté par la recherche (overhead)** est de :

$$E(m, x, y) - xy * n = (x+y) * (\lambda \sqrt{n} \log n + \mu) + \gamma \sqrt{n} + \beta$$

Remarque : Ce coût de recherche par quadtrie est meilleur que pour une liste triée (liste inverse) selon x ou une recherche binaire où $E(n, x, x) = (x+e)\, n + \lambda \log n$ Le coût ajouté par la recherche binaire est $E(n, x, x) - x^2 * n = x * (1+e-x)\, n + \lambda \log n$; e est la largeur d'un rectangle de la collection.

Evaluation théorique des mixed-quadtrees sur cette collection pour les recherches géométriques dans le cas d'une recherche sous un point (pic)

Dans ce cas les listes des nœuds quaternaires sont remplacées par des arbres binaires adaptatifs et la recherche dans un nœud quaternaire devient alors :

$V\_LIST(i) = \log_2 (\alpha\sqrt{n} / 2^i) + \delta$; δ constante et seuil de partage binaire

$= 0.5 \log_2 n - i + \eta$; η constante avec $0 \le i \le m-2$ si on suppose $2 \le m$

sinon $V\_LIST(0) = 0.5 \log_2 n + \eta$

$V\_LIST(i) = \kappa$ pour $i = m-1$;

$$E(m, 0, 0) = \sum_{0 \le i \le m-1} V\_LIST(i)$$

$$= 0.5 * (m-1) \log_2 n - (m-1)\, m / 2 + \eta\, (m-1) + \kappa - \eta$$

$$= (m-1) [0.5 \log_2 n - 0.5\, m + \eta] + \kappa - \eta \qquad \textit{(a)}$$

Ainsi calculé en supposant que $m = \log_4 (\tau n / \beta) = 0.5 \log_2 n + \mu$ on a :

$E(m, 0, 0) = O(\log^2 n)$ ce qui est meilleur que $O(\sqrt{n})$ obtenu pour les pures quadtrees...

Si on **suppose** maintenant que **le nombre de niveaux quaternaires m est choisi constant**, le calcul s'évalue différemment ; on a alors :

$V\_LIST(i) = \log_2 (\tau n / \beta)$ pour $i = m-1$;

$$E(m, 0, 0) = \sum_{0 \le i \le m-1} V\_LIST(i)$$

$$= 0.5 * (m-1) \log_2 n - (m-1)\, m / 2 + \eta\, (m-1) + \log_2 (\tau n / \beta) - \eta$$

$$= (m-1) [0.5 \log_2 n - 0.5\, m + \eta] + \log_2 (\tau n / \beta) - \eta$$

$$= (m - 1) [0.5 \log_2 n - 0.5 m + \eta] + \log_2 n + \mu \qquad \textit{(b)}$$

et **E (m, 0, 0) est alors en O (log n)**

Ainsi si pour les cellules VLSI sur lesquelles on **construit un mixed-quadtree on choisit un large seuil de partage quaternaire (faible résolution m)**, aux petites et moyennes valeurs de N on réalise une recherche sous des points plus proches du calcul *(b)* où m est constant que du calcul *(a)*.

2.4. La recherche des plus proches voisins et la gestion de l'espace vide

Quelques algorithmes de CAO de VLSI ont besoin d'énumérer rapidement les plus proches voisins d'un rectangle ou atome de géométrie. Des structures de données très spécialisées sont alors nécessaires pour résoudre le problème de la recherche du plus proche voisin [PRE 85]; mais ce problème n'est pas toujours crucial. Ce problème est cependant cependant important pour la plupart des algorithmes de compactage du layout. Certains de ces algorithmes de compactage doivent synthétiser l'espace vide *(chap. 6 §5.2.)*. Il peut être alors intéressant de conserver explicitement avec la collection des rectangles du layout l'espace vide qu'ils laissent dans la cellule.

Un rectangle d'espacement peut ainsi être préalablement associé au rectangle englobant une cellule; le premier élément placé dans cette cellule produit alors une réécriture de rectangle d'espacement, et ensuite chaque élément placé dans cette cellule *(figure 11)* produit une **réécriture** des divers rectangles d'espacement *qu'il couvre*, rectangles que l'on recherche de manière géométrique (ce qui accroît le coût de mise à jour du quadtree associé). La **synthèse des îlots d'espacement** entre les divers dispositifs placés dans la cellule est alors réalisable par l'algorithme général d'extraction des nœuds d'un layout (**depth-first ou union -find**); la relation de connexité géométrique **"appartenir au même îlot"** est une relation d'équivalence dans l'ensemble des rectangles d'espacement dont les **classes d'équivalences** constituent les divers **îlots** de la cellule.

Mais la représentation explicite dans la structure de données des rectangles / tuiles d'espacement accroît le coût mémoire de cette structure. De plus, la plupart des algorithmes dépendent du nombre total de rectangles, ce qui risque de les rendre inefficaces. Cependant dans une cellule formée de **N rectangles solides ou atomes de géométrie** sans recouvrement il n'y aura **pas plus de 3N + 1 rectangles espaces.** Ce résultat est démontré par récurrence par [OUS 84]. Pour des rectangles qui ont des superpositions, moins de rectangles représentent la même géomètrie et ce résultat est peut-être à majorer légèrement , pour N rectangles avec recouvrements.

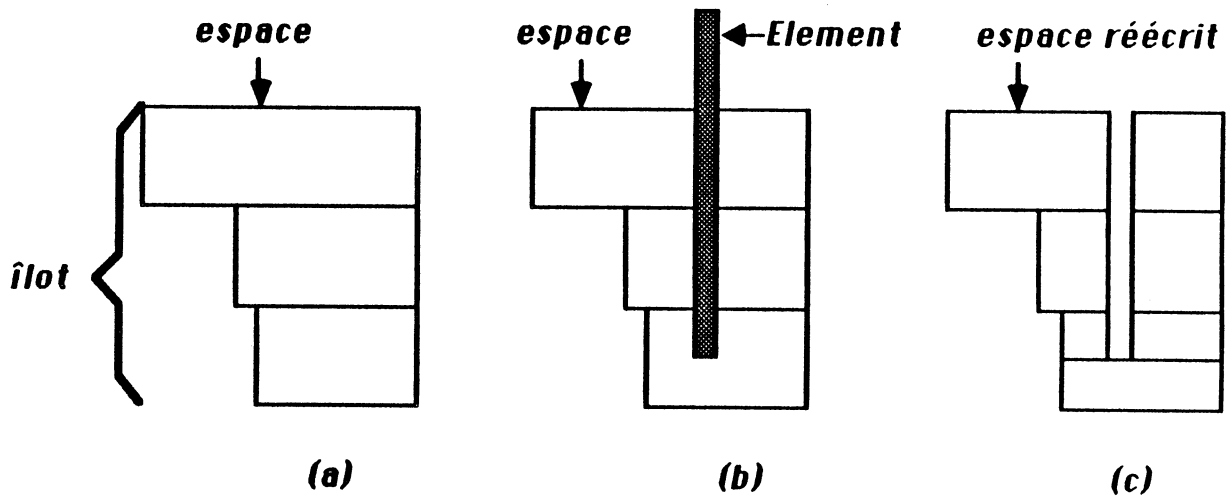


Figure 11 - *La réécriture incrémentale des* ***rectangles "espace"*** *lorsqu'un élément est inséré dans une cellule permet de maintenir l'espace laissé vide ; le processus de réécriture commence alors avec un seul* ***rectangle "espace"*** *dans le cas où la cellule est rectangulaire. Les* ***rectangles "espace"*** *sont maintenus dans le quadtree de la cellule et les différents îlots sont reconnus par l'algorithme (union-find) d'extraction des noeuds de la géométrie des masques de la cellule. Les suppressions d'éléments sont simples puisqu'il suffit de remplacer l'élément par un rectangle d'espacement et d'unifier (avec union-find) les îlots adjacents obtenus par recherche géométrique. L'évaluation de cette méthode de synthèse de l'espace pour des cas spécifiques (2D-compacteurs) est un problème ouvert.*

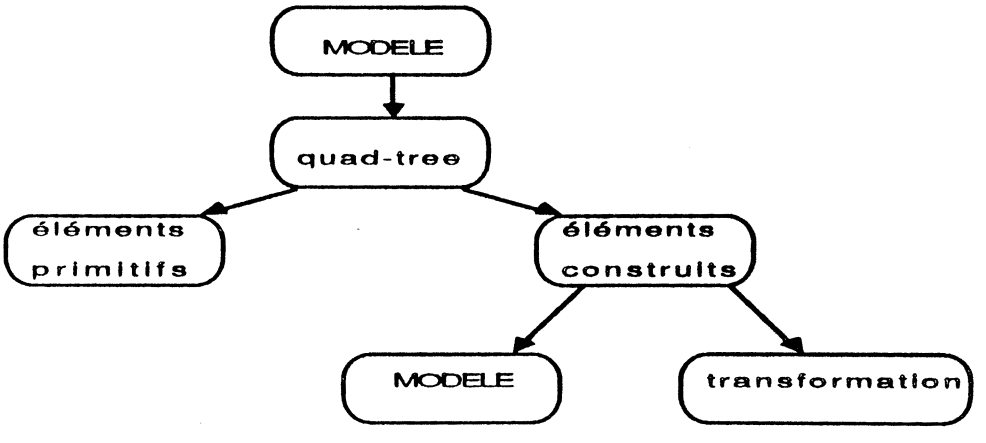
2.5. Intérêt de la méthode pour les bases de données géographiques

Les recherches géométriques constituent aussi l'un des problèmes des bases de données géographiques. L'extraction de toutes les régions couvrant au moins partiellement un domaine de recherche est souvent demandée. Certaines de ces bases de données emploient des méthodes de clefs analytiques. Ces clefs analytiques permettent de localiser rapidement les données qui doivent être extraites de la base. Elles sont généralement basées sur une **décomposition quaternaire** du plan [MOR 66], [WEB 79], [COM 81], [KLI 79], [JON 81], [GAR 82]. Le problème du recouvrement au moins partiel du domaine de recherche est résolu par [ABE 83], [ABE 84] à l'aide d'une décomposition quaternaire identique à celle du quadtree adaptatif de rectangles; la clef analytique d'un enregistrement indique alors sa classe d'appartenance dans la partition quaternaire des données *(Annexe 3).* La recherche géométrique est alors réalisée selon le même principe de base; elle exige l'extraction de la base de donnée de tous les enregistrements des classes à examiner. Si on considère les résultats théoriques et expérimentaux obtenus pour les simples décompositions quaternaires, on peut proposer aux géographes un raffinement du calcul de ces clefs analytiques par une **décomposition binaire secondaire** analogue à celle des **mixed-quadtrees**. La clef ainsi calculée assurerait une **meilleure localisation** des régions dans les quadrants, ce qui produirait un taux d'echec analogue à celui des mixed-quadtrees.

2.6. Quadtrees et hiérarchie des descriptions géométriques

Comme cela a été vu *en §2.1.1* les layout hiérarchiques de VLSI ne produisent pas de comportement de pire cas aux algorithmes de recherche géométrique par quadtree adaptatif de rectangles. Un quadtree ou mixed-quadtree est ainsi associé à la description géométrique de chaque modèle de cellule.

L'interface suivante permet aux applications hiérarchiques de créer des **hiérarchies de quadtrees**. Le **quadtree associé à chaque modèle fait implicitement référence aux quadtrees des modèles de ses sous-cellules**. En effet les *éléments construits* (sous-cellules instances de symboles) sont placés dans le quadtree exactement comme des *éléments primaires* et font ainsi géométriquement référence aux modèles qu'ils instancient et en conséquence à leur quadtree associé.



Les quadtrees de cette forêt peuvent être créés ou supprimés par l'appel des actions ou procédures de l'interface générale d'utilisation des quadtrees par les applications. *Un sommaire de cette interface est donné ci-après (& annexe 1.§.2.).*

3. SPECIFICATIONS DE L'INTERFACE PROPOSEE

Les **quadtrees créés** sont repérés par un **numéro d'identification** (type entier positif); ils sont accessibles dans l'index de l'ensemble des quadtrees par **hash-coding sur leur numéro d'identification**. Les différents **mixed-quadtrees adaptatifs de cette forêt** peuvent être tous ouverts soit avec les même seuils de partition soit avec des seuils de partition différents. Ainsi pour des applications telles que les ERC et DRC les seuil de partition dépendent de la dimension moyenne des régions de recherche c'est-à-dire de la dimension moyenne des éléments rangés dans l'arbre et éventuellement des régles de garde. Il est donc conseillé pour ce type d'application de choisir un mode de partage absolu et des seuils de partition globaux à toute la forêt. D'autres applications comme les éditeurs gèrent des fonctions très différentes dans un même process et il peut être intéressant de choisir différents modes et seuils de partitions (recherches géométriques multimodales). L'interface proposée offre ces divers possibilités.

Actions proposées par l'interface :

0. action BEGIN_QD;
entrée : QD_TRESH : mode et seuils de partition quaternaire et binaire ou résolution de la partition par défaut des quadtrees;

Cette action initialise l'ensemble des quadtrees c'est-à-dire l'index des quadtrees et le mode de résolution par défaut de la partition des données.

1. action CREATE_QD;
entrée : CELL_ID : numéro d'identication du quadtree
CELL_BBOX : cadre enveloppe du quadtree; il sera modifiable tout au long du process; il est cependant conseillé de le choisir englobant la liste des objets connus (optimisation)
QD_TRESH : mode et seuils de partition quaternaire et binaire ou résolution de la partition.

Cette action vérifie qu'il n'y a pas déjà de quadtree ouvert à ce numéro en mettant à jour l'index et sinon produit un message d'erreur.

2. action DELETE_QD;
entrée : CELL_ID : numéro d'identification;

Cette action supprime un quadtree de l'index et de l'ensemble des quadtrees.

3. action ADD_QD;
entrée : CELL_ID : numéro d'identification
ITEM_BBOX : coordonnées du rectangle enveloppe de l'élément
ITEM_REF : adresse de l'enregistrement réél de l'élément.
Cette action ajoute un élément au quadtree;

4. action SUPR_QD;
entrée : CELL_ID : numéro d'identification
ITEM_BBOX : coordonnées du rectangle enveloppe de l'élément
ITEM_REF : adresse de l'enregistrement réél de l'élément.

Cette action supprime un élément du quadtree;

5. action SELECT_QD;
entrée : CELL_ID : numéro d'identification
REGION : region de recherche
sortie : L_OBJECT : liste des éléments trouvés

Cette action de **recherche géométrique** recherche les éléments couvrant (au moins partiellement) la region de recherche géométrique **(range-searching)**.

6. action ACCESS_QD;
entrée : CELL_ID : numéro d'identification
ITEM_BBOX : coordonnées du rectangle enveloppe de l'élément
sortie : L_OBJECT: liste des éléments trouvés de coordonnées identiques à ITEM_BBOX

Cette action est une **recherche exacte** des éléments de même coordonnées que ITEM_BBOX; cette recherche pourrait aussi être réalisée par SELECT_QD mais elle serait alors beaucoup moins efficace (SELECT_QD est un *range-searching* tandis que ACCESS_QD est un *exact query* ou *exact match* [KNU 73]) .

Remarque : Les ITEM_BBOX sont les coordonnées inférieures gauches et supérieures droites du plus petit rectangle englobant les éléments rangés dans un quadtree. Toutes les coordonnées (item, cadre, region) peuvent être arrondies, tronquées ou transformées en entier par les applications (principalement celles qui effectuent des transformations); l'essentiel est de **ranger et de rechercher toujours les éléments avec les même coordonnées associées** qui constituent la **clef d'accés géomètrique** des éléments.

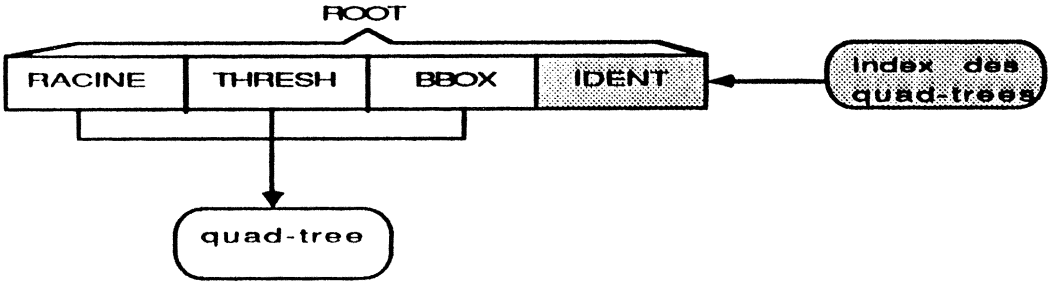


Figure 12 - ***Gestion de la forêt de quadtrees (mixed-quadtrees).*** *L'interface multi-quadtrees permet de gérer différentes hiérarchies de quadtrees. Les quadtrees ont un numéro d'identification et sont accessibles par hash-coding sur ce numéro; une <root> est associée dans cette interface à chaque quadtree créé ; elle maintient l'adresse <RACINE> du nœud racine du quadtree, sa résolution ou seuil de partition <TRESH> et le rectangle associé à la racine du quadtree <BBOX>.*

Recherche géométrique des éléments primitifs dans la profondeur d'une hiérarchie de rectangles.

action **H_SELECT;**
{entrée : CELL_ID : numéro d'identification de la cellule
REGION : region de recherche
sortie : L_OBJECT : liste des éléments trouvés}

Cette action de **recherche géométrique** recherche les **éléments primitifs** couvrant (au moins partiellement) la région de recherche géométrique *(range-searching).* Cette action n'est pas offerte par l'interface utilisateur et constitue seulement un **exemple d'utilisation hiérarchique** des actions de cette interface par les programmes d'application. *(cf. figure 13 et annexe 1.§3.4.).*

1. Selectionner par action SELECT_QD les éléments primitifs ou construits couvrant (au moins partiellement) la région de recherche.
2. Afficher les éléments primitifs et aller chercher les sous-éléments des éléments construits dans les modèles associés aux éléments construits (sous-cellules) en effectuant l'action H_WINDOW sur leur modèle avec comme région de selection la REGION et comme transformation absolue la transformation du symbole (son inverse sera en fait appliquée à REGION pour la repérer par rapport au repére attaché au modèle).

action **H_WINDOW;**
{ entrée : TOP_SYMBOL : symbole / modèle de plus haut niveau;
SYMBOL : le symbole / modéle instancié en un élément construit;
T_A : transformation absolue du symbole dans TOP_SYMBOL;
REGION : région de recherche repérée par rapport à TOP_SYMBOL
{ cette action va chercher les sous-éléments du symbole instancié par T_A et couvrant (au moins partiellement) REGION (repérage absolu); elle affiche ses éléments primitifs et récurse sur ses éléments construits }

1. Inverser T_A -> reverse_T_A et calculer R_REGION <- reverse_T_A (REGION)
2. Selectionner les éléments de SYMBOL avec cette R_REGION ainsi repérée par rapport au repére attaché au symbole
3. Afficher les éléments simples (action DISPLAY) et récurser sur les éléments construits aprés mise à jour de la transformation absolue (par sa composition avec la transformation relative de l'élément construit).

action DISPLAY;
entrée : TOP_SYMBOL : symbole de plus haut niveau
: ELEMENT : élément primitif à afficher repéré par rapport à son père hiérarchique
entrée : T_A : transformation absolue de l'élément
{ affichage de l'élément dans TOP_SYMBOL après la transformation T_A}

Cette application des recherches géométriques à des hiérarchies de symboles montre la nécessité de passer toujours avec un **élément primaire ou construit** son **contexte hiérarchique** (plus ou moins détaillé) mais de conserver toujours le

repérage relatif de l'élément par rapport à son *père hiérarchique*, dans la hiérarchie de conception. En effet les **algorithmes hiérarchiques n'instancient jamais réellement** les éléments pour les traiter; ils utilisent seulement des repérages absolus (par rapport au niveau le plus haut) et relatifs (par rapport aux repère attachés aux éléments construits); ils opèrent ainsi des **instanciations virtuelles**.

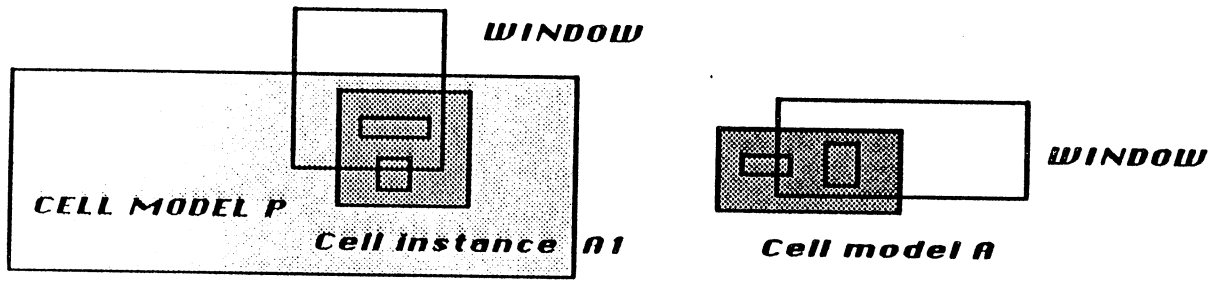


Figure 13 : ***Hiérarchie de cellules et recherches géométriques.****. Une simple recherche géométrique sur un modèle de cellule est réalisé par une recherche géométrique sur son quadtree associé; une recherche géométrique dans toute la profondeur de la hiérarchie de conception (sélection hiérarchique) est réalisée en cherchant aussi récursivement dans les sous cellules couvrant la région (window) de sélection; la région de sélection doit être alors repérée par rapport au repère attaché à leur symbole associé (par la transformation inverse de la transformation appliquée au symbole). Les applications hiérarchiques gèrent elles-mêmes description et recherches géométriques hiérarchiques. Cette méthode assure l'indépendance du système de recherche et des diverses applications.*

4. EVALUATION EXPERIMENTALE DES DIVERS QUADTREES

Le comportement des quadtrees a été étudié sur les description symboliques et aussi sur les descriptions logiques des circuits réels pour de petites et moyennes cellules composées d'éléments primaires (rectangles / segments / polygones) et d'éléments structurés représentés par leur placement (vue externe des sous-cellules ou instances de modèles) [BER 85]. Ces cellules provenaient du circuit *Microprocesseur de Traitement du Signal* réalisé au Centre National d'Etudes des Télécommunications de Grenoble.

4.1. L'évaluation de la méthode sur des layouts de VLSI

Différents seuils de partage quaternaire et binaire furent choisis en mode relatif c'est-à-dire que les résolutions quaternaires et binaires (Q-depth et B-depth) paramétraient la décomposition des données. Les différents mixed-quadtrees ainsi crées peuvent ainsi correspondre à des structures de données très différentes. Les cas suivants apparaissent :

1. Les **résolutions quaternaires et binaires sont nulles** : le mixed-quadtree est alors une simple **liste d'éléments.**
2. La **résolution binaire est nulle** : le quadtree est un simple **quadtree adaptatif.**
3. La **résolution quaternaire est nulle** : le quadtree est un **arbre binaire adaptatif.**
4. Aucune des deux résolutions n'est nulle : le quadtree est alors un mixed-quadtree adaptatif (mixed-quadtrie). **Un mixed-quadtree de résolution quaternaire Q-depth et de résolution binaire B-depth est repésenté par la notation (Q-depth, B-depth).**

Ces seuils différents de résolution quaternaire et binaire nous ont permis d'étudier le comportement de ces divers structures et de les comparer entre elles *(figures 15 et 16).* Les algorithmes sont écrits en Pascal et ont été implémentés sur un VAX 11 / 785. Les quadtrees ainsi implémentés et évalués utilisent un seul niveau de mémoire, la mémoire interne. Les **résolutions maximales** de partage des données **correspondent** toujours pour les différents exemples traités à la réalisation de **perfect trees**; les *perfect trees* sont des arbres adaptatifs qui n'ont qu'un élément (en moyenne) par nœud terminal.

Un **espace mémoire de O (K)** est en moyenne utilisé par **chaque élément rangé dans le quadtree avec K constante multiplicative** pour la mémoire nécessaire à une collection de N éléments. Cette constante multiplicative dépend des résolutions. Dans les cas qui ont été étudiés et pour tous les types de **mixed-quadtrees** créés, cette **constante multiplicative de l'espace des données n'a jamais dépassé 2.5** et un **facteur multiplicatif de 1.5** correspond généralement à de **bonnes performances** des arbres ainsi construits. L'évolution comparative de l'espace mémoire ajouté a été étudié en facteur des résolutions quaternaires et binaires *(figure 18);* elle montre l'intérêt des partitions binaires moins coûteuses que les partitions quaternaires et aussi efficaces pour des recherches géométriques de faible surface *(figure 14).* **Les mixed-quadtrees adaptatifs** réalisent ainsi une structure **plus efficace que les simples quadtrees adaptatifs** pour différentes aires de recherche à égalité de mémoire ajoutée.

Ainsi les mixed-quadtrees sont des structures moins coûteuses en mémoire que les 4-D trees (qui on un facteur multiplicatif 4); le coût ajouté par les mixed-quadtrees aux simples listes d'éléments paraît peu supérieur au coût ajouté par les **multiple storage quadtrees** (avec K= 1.25) [BRO 86]; les *multiple storage quadtrees* doivent stocker les éléments de manière multiple dans les différents sous-arbres.

Les mixed-quadtrees paraissent aussi intéressants que les 4-D trees [ROS 85] pour la recherche des éléments couvrant (au moins partiellement) une petite région avec un ratio du nombre d'éléments examinés au nombre d'éléments trouvés variant en moyenne de 2 pour les moyennes fenêtres à 8 pour les petites fenêtres (point search) pour des cellules d'environ 80 transistors (700 rectangles) *(figure 14, qd (4,6))*.

De faibles seuils de partage (grande résolution Q-depth ou B-depth) sont meilleurs pour de petites fenêtres de recherche et de larges seuils de partage sont meilleurs pour de grandes fenêtres *(figure 17)*. Les **quadtrees adaptatifs** sont meilleurs pour les grandes fenêtres et les arbre binaires adaptatifs sont meilleurs pour les petites fenêtres et les recherches sous des points. Ainsi les **mixed-quadtrees réalisés avec un large seuil de partition quaternaire et un faible seuil de partition binaire réalisent un bon compromis** entre ces deux structures *(fig.10 & fig.11)*. De l'ordre de 2 ou 3 secondes pour environ 700 éléments les temps de construction des quadtrees et mixed-quadtrees sont raisonnables *(figure 14)*. Ils sont souvent négligables devant la durée totale du process qui les a créés *(annexe 5.§2.)*.

Une question essentielle est souvent posée par les utilisateurs potentiels d'une telle méthode de recherche : **Quelle est la taille minimale des cellules pour laquelle l'utilisation de quadtrees est intéressante ?** La réponse dépend beaucoup de la nature des applications susceptibles de bénéficier de cette méthode. Pour **une application de DRC conventionnel**, sur des cellules sans hiérarchie, l'expérience a montré, et le calcul a confirmé, qu'**à partir de 10 transistors** (et donc moins d'une centaine de rectangles) la méthode de recherche géométrique devenait intéressante et **qu'au delà les gains devenaient très vite importants**. Pour des applications telles que **les ERC conventionnels** la méthode géométrique est profitable pour de très petites cellules de **seulement quelques transistors;** cette application bénéficie alors d'une recherche rapide des quelques éléments situés sous une très petite fenêtre de recherche par rapport à la surface globale de la cellule et les gains réalisés sont alors importants *(annexe 5.§.2.)*.

Les diverses applications qui utilisent l'interface d'utilisation des mixed-quadtrees trouvent généralement leurs **meilleurs seuils de partage ou résolutions quaternaire et binaire** par **optimisation du temps global d'un process** sur une cellule type ou sur un ensemble de cellules (conceptions) très diverses.

Les **applications hiérarchiques** utilisent la hiérarchie des mixed-quadtrees induite par la hiérarchie de conception pour des **recherches géométriques** simples ou hiérarchiques (range searching); mais elles les utilisent aussi comme des index géométriques qui permettent l'**accès géométrique** aux éléments des instances de cellule par **recherche géométrique dans une région** (range

searching) sur leur quadtree associé ou par **recherche géométrique exacte** (exact match / query) [KNU 73]. Cette double fonction de recherche et d'accès géométriques assurent l'intérêt et l'efficacité globale de cette méthode pour des applications hiérarchiques. Si **les divers quadtrees de la forêt hiérarchique** ainsi **induite par une conception VLSI** paraissent individuellement peu équilibrés, ils sont en fait équilibrés *de manière cachée* dans la profondeur de la hiérarchie de conception puisque la distribution des rectangles / atomes de géométrie est de manière cachée assez uniforme [BEN 80]. Il est ainsi facile de les optimiser globalement par un **paramètrage absolu et global des seuils de partition quaternaire et binaire.**

Cet accès géométrique a été conçu et **testé sur de petites et moyennes conceptions VLSI sans hiérarchie et sur des conceptions hiérarchiques.** Il n'a donc pas été évalué pour des conceptions complètement instanciées c'est-à-dire pour N grand (nombre de rectangles / d'atomes). Cependant cette méthode doit être aussi efficace pour de grandes valeurs de N. Afin de s'assurer globalement de l'efficacité de cette méthode sur de plus grandes cellules, **N cellules sans hiérarchie** d'environ 80 transistors ont été **juxtaposées et expansées**; le **DRC conventionnel** sur la cellule sans hiérarchie ainsi obtenue s'est réalisé en un **temps de $N^{1.1}$ fois le temps obtenu sur une seule cellule.** Ce résultat laisse **espérer un bon comportement de la méthode aux grandes valeurs de N et pour le même type de distribution des données.**

ratio A0 / A	*éléments de A*	*éléments examinés / éléments trouvés*		
en % d'éléments	*nombre*	*(0,0)*	*(4,0)*	*(4,6)*
100 %	691	1	1	1
42 %	294	2.3	1.65	1.26
10 %	71	9.7	3.34	2.15
3.47 %	24	28.82	6.37	2.37
1.01 %	7	99.00	18.45	3.72
0.29 %	2	345.82	64.10	9.51
0.14 %	2	345.82	64.10	8.00
0.00 %	1	691.	128.00	16.00
	0			
BUILD		1 s	2 s	3 s

Figure 14 - ***Ratio du nombre d'éléments examinés par le nombre d'éléments trouvés*** *dans des régions d'aire A ; A0 est l'enveloppe de la cellule; (p,q) : quadtree de résolution quaternaire p et de résolution binaire q.*
BUILD : temps de construction du quadtree (p,q).
Remarque : Le graphique correspond aux moyennes de 30 mesures réalisées sur des fenêtres placées aléatoirement; les petites fenêtres qui tombaient dans un espace complétement vide de la cellule et correspondaient à un temps de recherche presque nul ont été éliminées de la moyenne et regénérées (effets de bords).

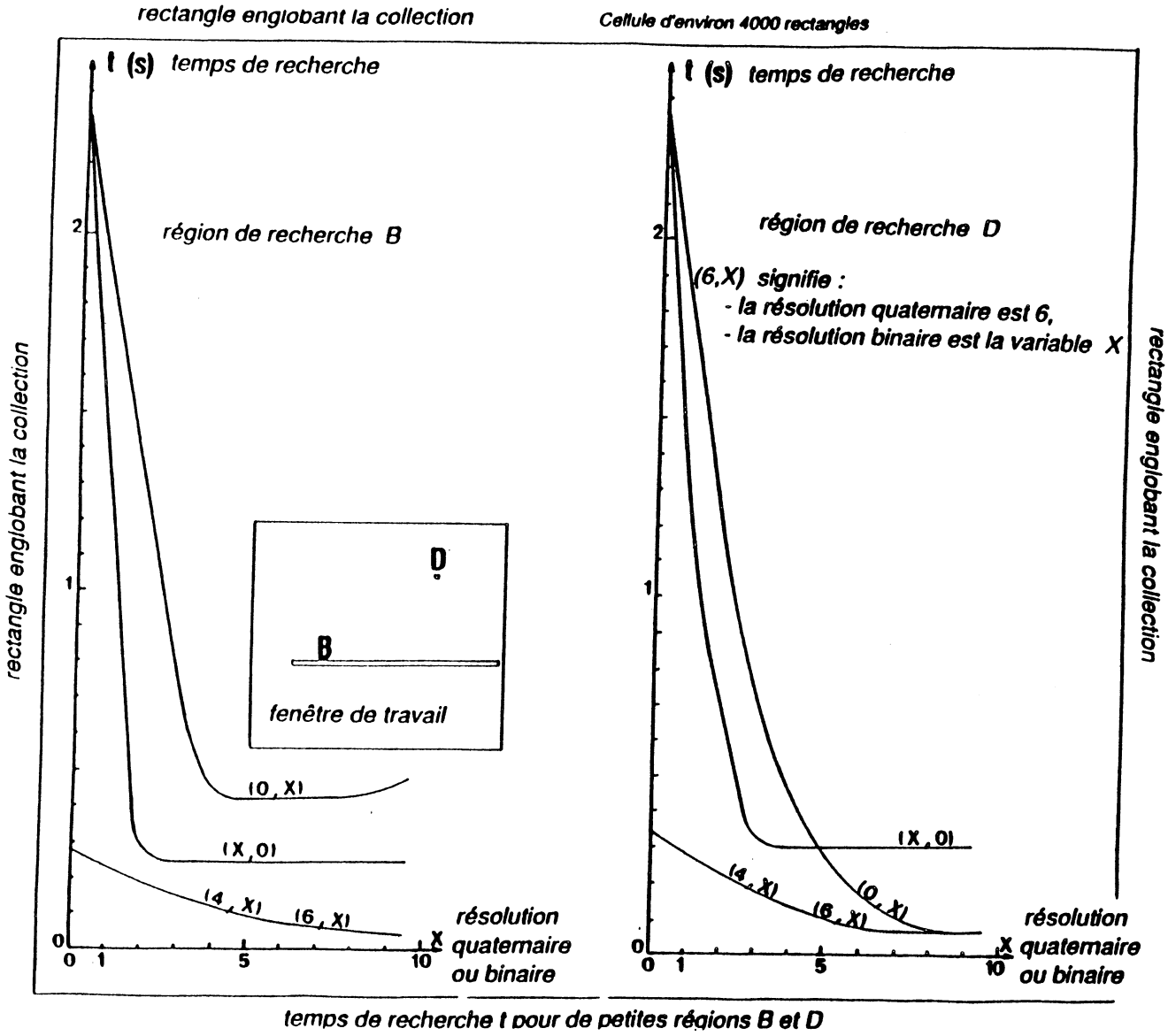


Figure 15 - ***Performance des recherches géométriques*** *sur une conception correspondant à environ 4000 rectangles, pour de petites régions de sélections relativement à la taille du rectangle englobant la cellule et* ***en fonction des résolutions quaternaire et binaire des quadtrees*** *,à gauche pour la fenêtre de dimension B, à droite pour la fenêtre de dimension D;* ***pour (6, X) lire la résolution quaternaire est 6, la résolution binaire est X.***

Remarque : Le graphique correspond à la moyenne de 30 mesures, estimée à moins de 5 % dans l'intervalle de confiance à 0.95.

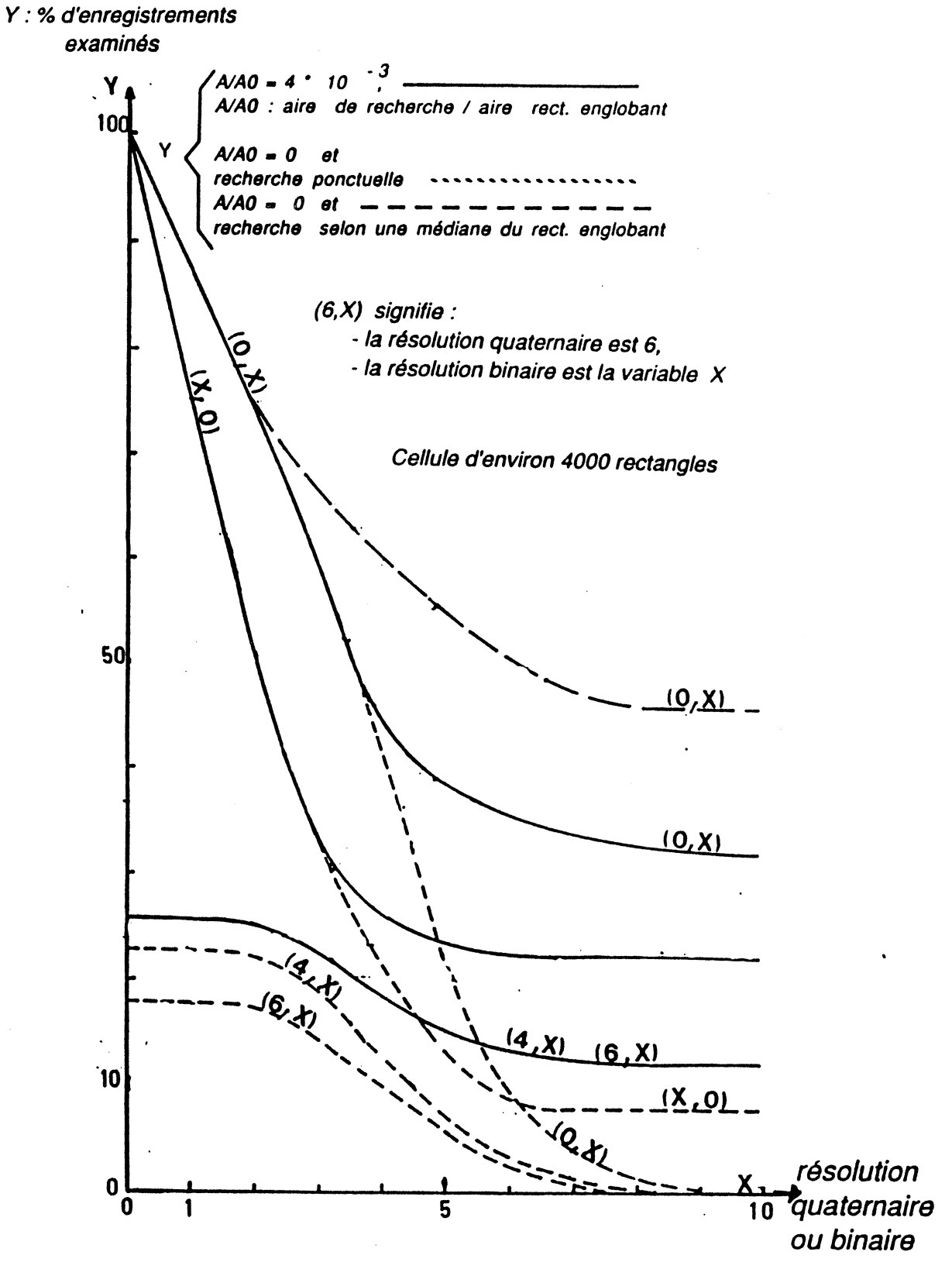


Figure 16 : **Pourcentage d'éléments examinés par diverses recherches géométriques** sur une cellule RAM (4000 rectangles).

Y = (nombre d'éléments examinés/ nombre d'éléments trouvés) * 100;
Les fenêtres de sélection d'aire A sont données en fonction de leur dimension relativement au rectangle enveloppe de la cellule $A = A_0 * 2^{-p} * 2^{-q}$, **les régions d'aire non nulle sont en trait plein ; les régions associées à des segments sont en pointillé long (p=10, q=0); les régions associées à des pics (point searchs) sont en pointillés courts (p=10, q=10)**
pour (6, X) lire la résolution quaternaire est 6, la résolution binaire est X.

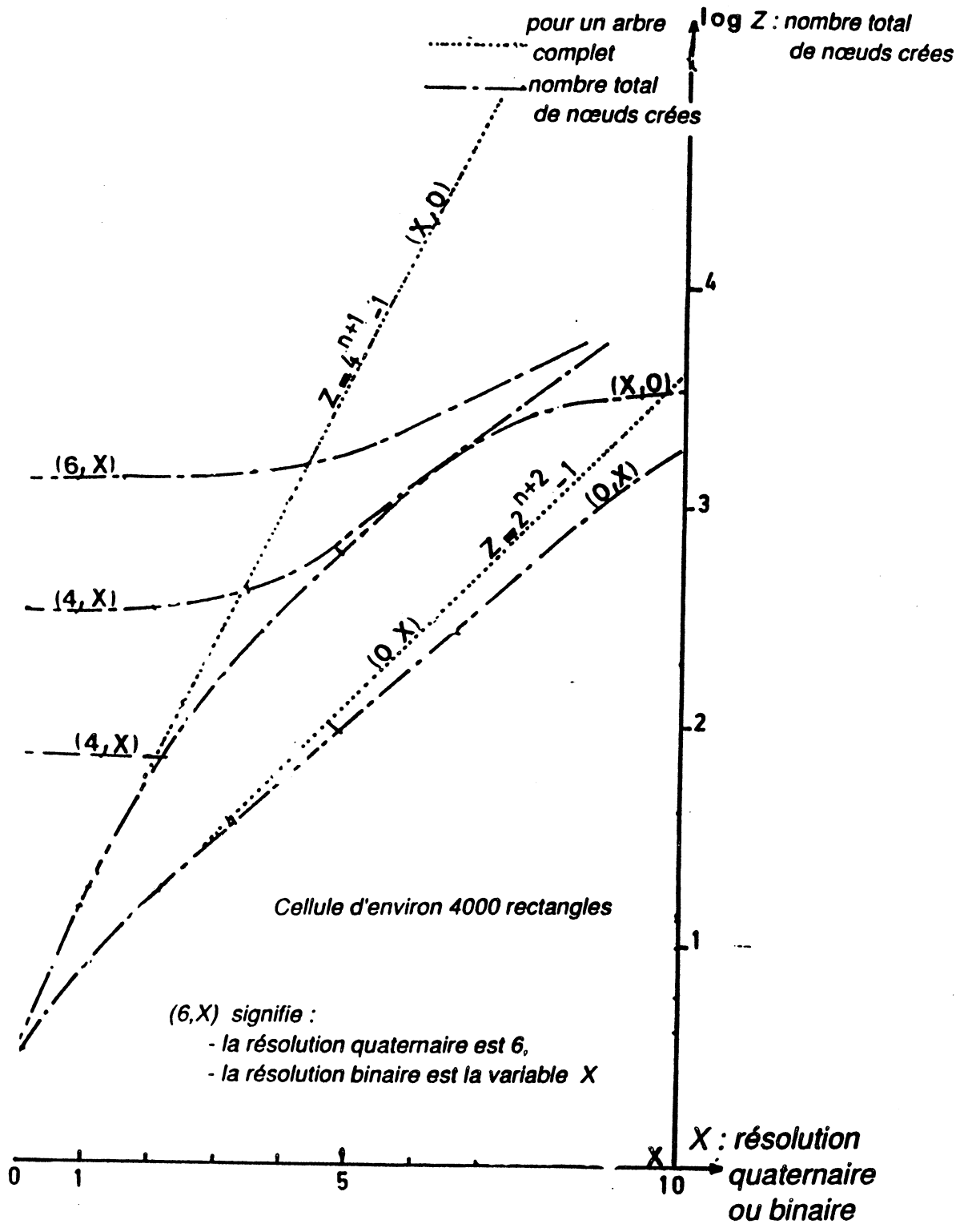


Figure 17 - **Espace mémoire utilisé en fonction des valeurs des résolutions quaternaire et binaire des divers quadtrees réalisés** sur une cellule RAM (4000 rectangles). **Z = noeud créés (binaires et quaternaires); (en log_{10} Y)**
pour (6, X) lire la résolution quaternaire est 6, la résolution binaire est X

Les lignes pointées en (a) représentent la mémoire théorique d'un arbre complet de même paramètres. **L'espace mémoire Z est en pointillés longs.**

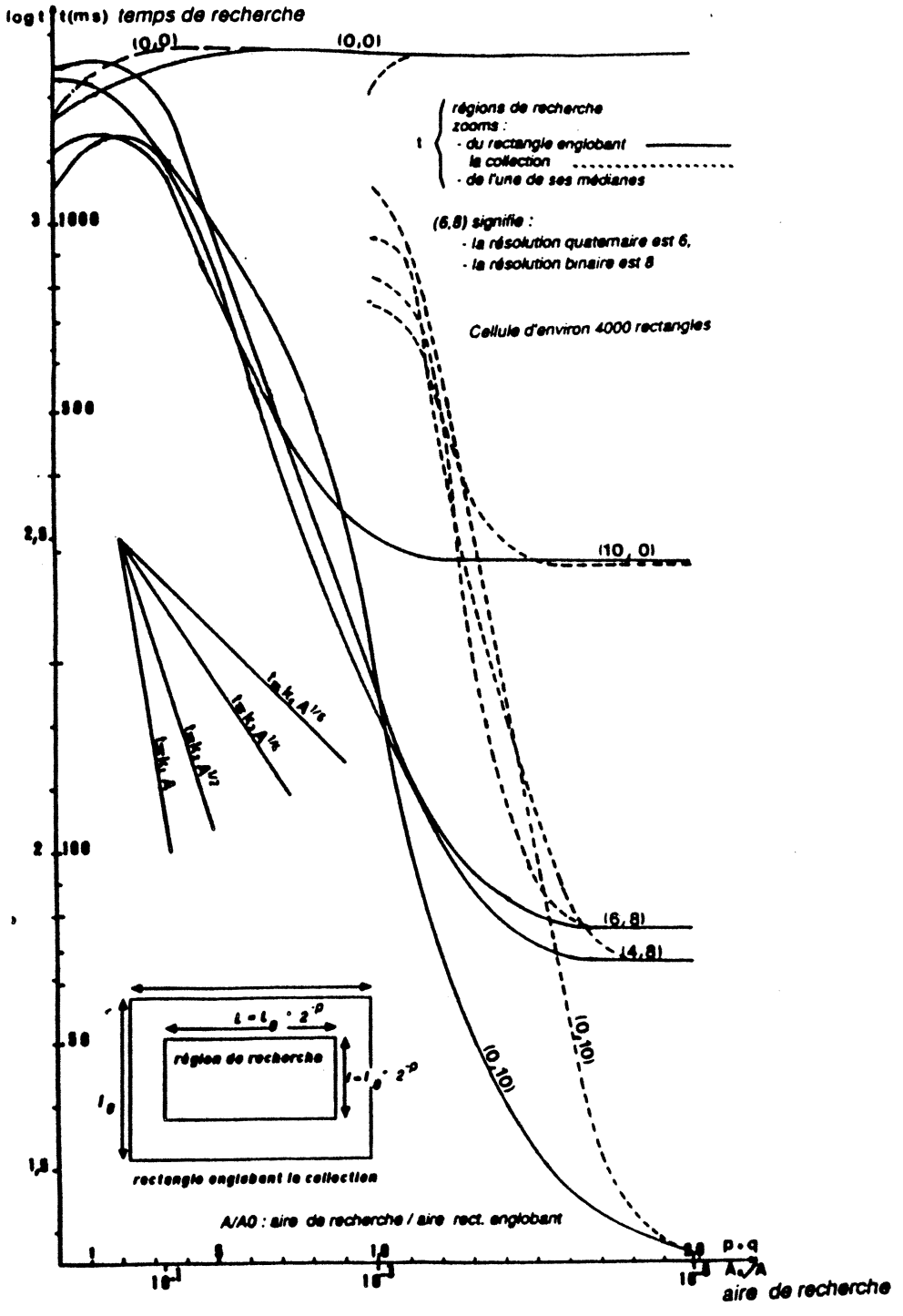


Figure 18 : ***Temps de recherche t, en fonction des aires des régions de recherche pour divers quadtrees*** *créés sur la cellule RAM ('4000 rectangles);*

A0 : aire du rectangle enveloppe de la cellule; $A_0 = L * \ell$

A : aire de la région de sélection donnée comme suit $L=L_0 * 2^{-p}$, $\ell=\ell_0 * 2^{-q}$ *avec p , q : (0..10);*

ainsi p+q est proportionnel à ***log (A/A0)*** *marqué sur l'axe horizontal.*

Les ***recherches géométriques d'aire presque nulle (segments) sont en pointillé long*** *, les recherches géométriques d'aire non nulle sont des* ***"zooms" du rectangle enveloppe de la cellule , représentés en trait plein.***

(4,8) signifie que la résolution quaternaire est 4 et la résolution binaire est 8.

Remarque : Le graphique correspond aux moyennes de 50 mesures par fenêtre; ces moyennes sont estimées à moins de 5 % dans l'intervalle de confiance à 0.95.

Le premier extrémum de chaque courbe est un effet de bord. Le deuxième extrémum correspond à la saturation du quadtree.

4.2. L'adéquation des résultats expérimentaux au modèle théorique

Les quadtrees adaptatifs de rectangles ont, pour des layouts aux distributions uniformes de rectangles, une profondeur globale en O (log N); pour un même seuil de partition, lorsque le nombre d'éléments est 4 fois plus grand, la profondeur de l'arbre augmente de 1. Cette profondeur globale donnerait ainsi pour des quadtrees complets, *figure 17,* la ligne pointée (X,0) pour les quadtrees purs, la ligne pointée (0,X) pour des partitions essentiellement binaires ou quadtrees à large seuil de partition quaternaire. Mais sur une cellule de 4000 rectangles, les divers quadtrees sont loin d'être complets; ils sont d'autant plus incomplets que leur seuil de partition est plus petit. On remarque ainsi l'infléchissement des courbes (X, 0) et (0, X) en pointillé pointé, à mesure que la la profondeur augmente.

Les quadtrees à large seuil quaternaire et faible seuil binaire de partition sont globalement plus économiques *(figure 17) (fig. 15, 16, 18)* et plus performants. *L'annexe 5* présente *en §2.1* l'amélioration des outils conventionnels d'analyse du layout, par l'utilisation du sous-système de recherche géométrique. Les mixed-quadtrees qui améliorent le mieux les outils conventionnels d'analyse du layout sont de 1.5 à 1.8 fois plus volumineux que de simples listes d'éléments; pour N cellules juxtaposées et expansées, le rapport de leur volume est un ratio pratiquement indépendant de N, avec une même partition des données. Ainsi la mémoire utilisée par un mixed-quadtree est en O (N), ce qui correspond à son évaluation théorique, *en §2.1.1.*

L'optimum des performances des outils conventionnels d'analyse est obtenu par une bonne adéquation des seuils de partage aux aires de recherche, indépendamment du nombre d'éléments à traiter. Ainsi la partition des données qui optimise un traitement tel qu'un D.R.C. ou un E.R.C. est indépendante de N *(annexe 5).*

Les évaluations expérimentales ont confirmé la supériorité des *mixed-quadtrees.* Les *quadtrees purs* sont intéressants pour les grandes aires de selection mais leur efficacité devient constante à partir d'une certaine aire à cause du blocage des éléments dans les nœuds quaternaires. Les *figures 14* et *16* comparent d'ailleurs leurs *taux d'échec* respectifs - rapport du nombre de nœuds examinés au nombre de nœuds trouvés - en fonction de différentes aires de recherche. Les partitions binaires secondaires diminuent sensiblement ce taux d'echec et réalisent ainsi une recherche très locale; très peu d'éléments sont examinés et ces éléments sont toujours assez proches de la région de recherche.

Pour N cellules juxtaposées et expansées, les D.R.C. ont été réalisés avec des temps de l'ordre de $N^{1.1}$ *(annexe 5§2.1.).* Des temps de l'ordre de $N^{1.6}$ étaient observés lorsque l'on utilisait des simples quadtrees à la place des mixed-quadtrees. De plus les gains de temps observés sont optimaux et très supérieurs dans le cas de l'utilisation des *mixed-quadtrees. (annexe 5§figure 6.).* Ces gains de temps sont encore plus importants pour l'algorithme d'extraction des nœuds électriques pour lequel les recherches géométriques ont des aires presque nulles *(annexe 5.§figure 8).* La *figure 18* a d'ailleurs permis une prédiction de ces résultats; si on compare les temps de recherche pour une même aire, les mixed-quadtrees réalisent l'optimum. Les *mixed-quadtrees* à large seuil de partition quaternaire et faible seuil de partition binaire réalisent l'optimum des temps de

recherches géométriques ponctuelles; cet optimum correspond à un taux d'échec minimum. Les évaluations expérimentales de l'efficacité des mixed-quadtrees semblent avoir une bonne adéquation avec l'évaluation théorique faite *en §2.3.*, puisque les recherches ponctuelles ont théoriquement une complexité théorique entre O (log N) et O ($\log^2$ N) pour une collection de N éléments uniformément distribués.

Pour les layouts de VLSI structurés, les performances sont localement les mêmes la où il n'y a pas de hiérarchie. Là où il y a de la hiérarchie les quadtrees des sous-cellules sont alors utilisés pour une recherche en profondeur et, de manière cachée, la distribution des données de base est la même que celle des layouts non hiérarchiques. La hiérarchie des quadtrees ainsi associée à la hiérarchie des données optimise globalement une même analyse du layout structuré avec des paramètres globaux de partition quaternaire et binaire bien choisis. L'optimisation des performances des D.R.C. et E.R.C. conventionnels montre qu'un même paramètrage des mixed-quadtrees assure, pour chacune de ces analyses, une bonne gestion des ressources de calcul et un temps d'analyse pratiquement optimal *(annexe 5, §2.1, figure 6).* On espère ainsi pouvoir utiliser efficacement ces hiérarchies de quadtrees, pour l'analyse hiérarchique et incrémentale des layouts structurés *(chap. 5 et 6).*

Chapitre 5

LE SOUS-SYSTEME DE RECHERCHE D'EQUIVALENCE PROPOSE

1. L'EXTRACTEUR DE BASE DE LA CONNECTIVITE électrique

Les interconnexions de transistors sont interprétées comme des nœuds. Un **nœud** est une région équipotentielle qui possède une **résistance parasite** et **une capacité par rapport au substrat** que les vérificateurs de connectique peuvent ignorer. Un nœud électrique est interprété selon le réseau ci-dessous *(cf. chap 3)*.

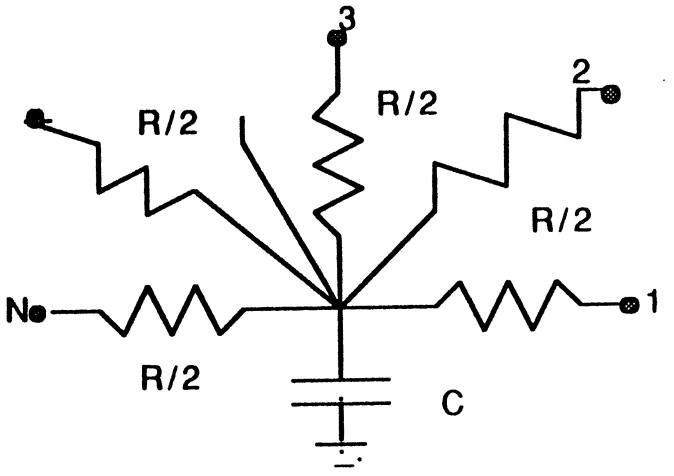


*Figure 1- **Interprétation des résistances et capacités de substrat des noeuds électriques** - Chacun des points 1, 2 , ..,N ci-dessus est une connexion au noeud par un **terminal.***

La description résultante du circuit est alors faite de trois sortes de composants : **les nœuds** décrits ci-dessus (ayant une résistance et une capacité), les **transistors** et les **capacités internodales**. Les transistors correspondent aux dispositifs électriques explicitement ou implicitement placés par le concepteur. Chaque transistor a une grille, un terminal lié au substrat et deux terminaux de diffusion (ou plus). Les capacités internodales ont chacune deux terminaux et une valeur de capacité.

L'information extraite du dessin des masques - symbolique ou physique - constitue l'extracteur de base qui ignore la hiérarchie. L'**extracteur de base** opère en trois passages à travers l'information géométrique du layout : le premier identifie tous les **nœuds**, le deuxième traite les **transistors** et le troisième calcule les **capacités entre nœuds**.

1.1. Construction des composantes connexes et accès géométriques

La reconnaissance des rectangles du layout ou atomes de géométrie qui appartiennent au même nœud électrique est réalisée par la **technique UNION-FIND** *(chap. 3.§4.2.2.)*. Pour générer les équivalences et réaliser les composantes connexes on procède d'abord à l'examen séquentiel de chaque rectangle t et des contacts qu'il peut avoir avec les autres rectangles du layout. Une **table de règles de mise en contact** permet de savoir si deux rectangles ainsi juxtaposés ou superposés sont électriquement en contact selon leur nature; dans ce cas l'action FAST-UNION-FIND est appelée afin de générer cette équivalence dans l'ensemble des composantes connexes de la cellule *(chap. 3.§4.2.3.)*. Si les deux atomes équivalents appartiennent à des composantes connexes différentes alors ces

composantes connexes sont fusionnées selon la méthode **Union-Find** *(chap. 3.§4.2.2.)*. Les étapes de cet algorithme consistent en :

1. examiner chaque rectangle t et extraire par recherche géométrique tous les rectangles voisins de t et qui ont avec lui un contact géométrique

2. examiner chacun de ces voisins si les règles de mise en contact électrique avec le rectangle t sont satisfaites. Dans ce cas générer une équivalence électrique entre les rectangles en contact par appel de l'action FAST-UNION-FIND avec le booleen Union à Vrai.

On ajoute aux équivalences ainsi générées les **équivalences purement structurelles** fournies par la base de donnée. L'action UNION-FIND est encore invoquée; les équivalences structurelles sont ainsi réalisées dans l'ensemble des composantes connexes par l'appel de l'action FAST-UNION-FIND pour chacune des paires de rectangles concernés.

1.2. Compression des composantes connexes

Aprés la génération des équivalences électriques, les composantes connexes de la cellule sont complétement "comprimées" afin que chaque rectangle pointe directement sur le représentant du nœud électrique que constitue la racine de l'arbre de *Fischer-Galler* associé à ce nœud *(chap. 3.§2.2.)*. La compression des arbres de *Fischer-Galler* est ainsi réalisée globalement par énumération et traitement - compression de chemin - de chaque rectangle de la cellule.

1.3. Accès à un nœud électrique

Dans un ensemble de rectangles formant un ensemble de nœuds électriques, l'un de ces nœuds peut être sélectionné soit par son nom, soit par l'un des rectangles qui la composent.

Après la phase d'extraction des divers nœuds electiques un **nœud électrique est recherché par son nom** c'est-à-dire par le **nom du rectangle qui représente sa composante connexe** par l'action **Selection-par-nom.**

L'action **Selection-par-nom** effectue une **recherche séquentielle** dans l'ensemble des rectangles; elle a en entrée le nom du rectangle; tous les rectangles de même nœud électrique sont selectionnés. Une composante électriquement connexe prend pour nom, le nom du rectangle, racine de l'arbre de *Fischer-Galler* associé.

Un table d'index assure **l'accès direct à un rectangle par son nom** ou *index (figure 2)*. L'accès à la racine de l'arbre de *Fischer-Galler* fournit ainsi le nom ou index de la composante connexe que constitue un nœud électrique. Cet accès est réalisé par l'exécution de **l'action récursive : rectangle := père (rectangle)** jusqu'à atteindre un rectangle qui n'ait plus de père*(figure 2)*.

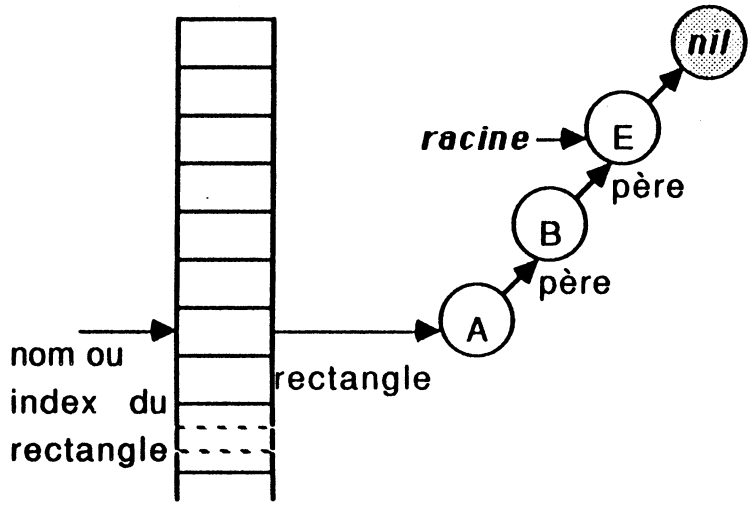


*Figure 2- **Accès au représentant d'un nœud électrique** à partir d'un rectangle du nœud - Cet accès est obtenu par la suite d'actions :*

1. accéder au rectangle par son nom ou index

*2. tant que **père** (rectangle) <> nil faire rectangle := **père** (rectangle). La fonction **père** dépend de l'implémentation choisie. Pour un layout structuré, ce représentant électrique est pour un rectangle A local à la cellule composée de A, B, E, etc.... Tout rectangle représentant électrique donne son nom au nœud électrique qu'il représente; il est ainsi facile de reconnaître l'équivalence entre deux rectangles d'une même cellule.*

Lorsque l'**on sélectionne un nœud électrique par son nom** tous les rectangles de la cellule sont séquentiellement considérés et le nom de leur composante connexe est recherché; si le nom du nœud électrique correspond à celui cherché alors le rectangle est sélectionné.

Après la phase d'extraction des divers nœuds electiques un **nœud électrique connexe est sélectionné par recherche géométrique** à **partir d'un rectangle de départ** par l'**action Spanner**.

L'action **Spanner sélectionne**, par recherche géométrique, à partir d'un rectangle de départ, l'une des parties connexes d'un nœud électrique. Cette action utilise la **technique "depth-first"** *(chap.3.§4.2.1.)* pour réaliser cette sélection. Elle a comme **entrée** un **rectangle du nœud** à selectionner. L'algorithme examine récursivement tous les rectangles qui touchent ce rectangle de départ et lui sont équivalents. La recherche des contacts est faite par recherche géométrique; la recherche d'équivalence est faite par action UNION_FIND.

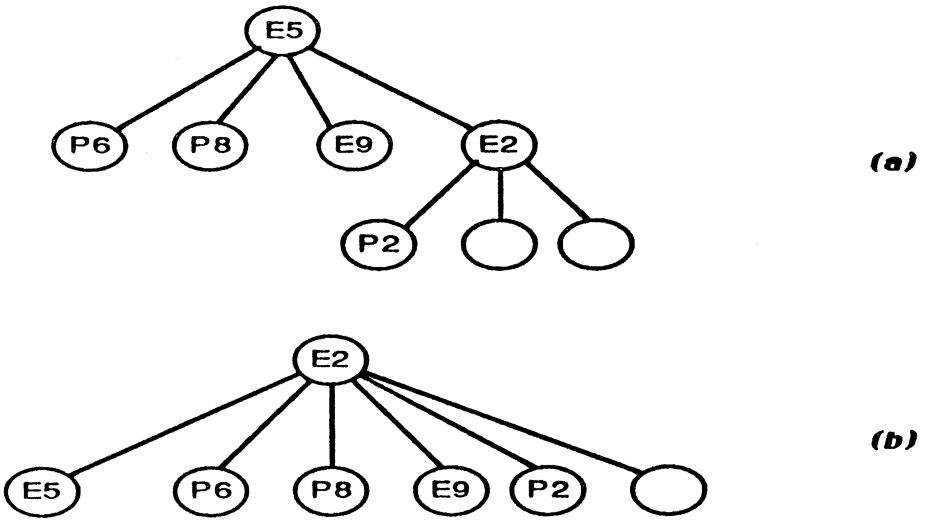


Figure 3 -***Représentation compacte du réseau des nœuds électriques et reconnaissance des nœuds externes*** -

(a) - Les nœuds électriques de chaque modèle sont représentés par des arbres de ***Fischer-Galler*** *; l'action* ***union-find*** *réalise la fusion de deux nœuds électriques : placement d'un arc entre les deux éléments équivalents et compression simultanée de ces arbres (en mode statique).*

(b) - Une compression plus complète des arbres de ***Fischer-Galler*** *est réalisée par un passage de tous les rectangles. Les éléments de plus petit index sont "remontés" à la racine de ces arbres; dans une conception hiérarchique certains rectangles représentent les "ports" ou accès de la cellule. Par convention, les ports ont des index inférieurs aux autres rectangles; ainsi les nœuds externes à une cellule sont facilement distingués puisque leur représentant électrique est un port..*

2. CONCEPTION STRUCTUREE ET RECONNAISSANCE DES NOEUDS ELECTRIQUES

L'entrée de l'extracteur est une description hiérarchique du layout; chaque cellule de layout contient l'information de niveau masque et l'information des sous cellules.

2.1. Les ports des sous-cellules

Dans l'ensemble des rectangles d'une cellule certains rectangles sont distingués comme étant les accès de la cellule; ces **éléments terminaux ou ports** de la cellule sont accessibles de l'extérieur et caractérisent alors l'interface d'utilisation de la cellule par les divers contextes qui vont l'utiliser.

2.2. La reconnaissance des nœuds électriques externes

Dans une analyse hiérarchique de layout structuré, il est nécessaire de **reconnaître si, dans une cellule, un nœud électrique est externe** c'est-à-dire si ses instances peuvent se fusionner avec d'autres nœuds des contextes

qui appellent cette cellule. Lors de la **phase de compression** des arbres de *Fischer-Galler*, un traitement séquentiel fait remonter un port à la racine de tout arbre de *Fischer-Galler* qui en contient au moins un. La relation d'équivalence restreinte au sous-ensemble des ports de la cellule est ainsi directe. L'index d'accès d'un rectangle induit une relation d'ordre total dans l'ensemble des rectangles et par convention, les ports sont inférieurs aux autres types de rectangles. Le traitement séquentiel de tous les rectangles fait ainsi remonter le rectangle de plus petit index à la racine des arbres de *Fischer-Galler;* ce rectangle représente ainsi le nœud électrique *(figure 3).* Alors un nœud électrique est externe si et seulement si il est représenté par un port.

2.3. Représentation compacte du réseau des nœuds électriques associé à une hiérarchie de conception

Les différents modèles de cellules utilisés par la hiérarchie de conception sont analysés de manière incrémentale par l'extracteur de graphe électrique. Dans une cellule, **chaque nœud électrique externe est représenté** de manière électrique **par un seul port** qui est alors le porteur de toutes les informations et valeurs internes liées au nœud. **La relation d'équivalence dans l'ensemble des ports d'un modèle est à réaliser dans les divers contextes qui utilisent le modèle.**

Les informations d'un nœud électrique externe d'un modèle sont ainsi sauvegardées dans la base de donnée des circuits comme des attributs du "port" qui le représente. La relation d'équivalence dans l'ensemble des ports est aussi conservée.

Chaque fois qu'un modèle est utilisé, les instances de ses ports constituent des **pins** pour la cellule appelante. Ces pins sont alors les porteurs, dans la cellule appelante, de l'information associée aux ports des cellules appelées. La relation d'équivalence dans l'ensemble des ports des cellules appelées est à réaliser dans les cellules appelantes sur l'ensemble des pins qui les instancient. On obtient ainsi de manière incrémentale une **représentation compacte du réseau des nœuds électriques associé à la hiérarchie de conception** *(figure 4).*

Soit une cellule constituée de sous-cellules. Lors de l'analyse hiérarchique de la connectivité électrique, un ensemble d'actions permettent d'**étudier la connectivité** électrique entre les deux **niveaux adjacents de la hiérarchie** de conception, représentés par la cellules et ses sous-cellules. Ainsi la transformation géométrique qui à un port fait correspondre, dans la cellule appelante, son instance de port - appelée pin - assure l'accès, dans la cellule appelante, au nœud électrique correspondant. Cette méthode utilise successivement une **recherche géométrique exacte** *(chap. 4.§3.6)* et la **réseau compact** des nœuds électriques, associé à la hiérarchie de conception.

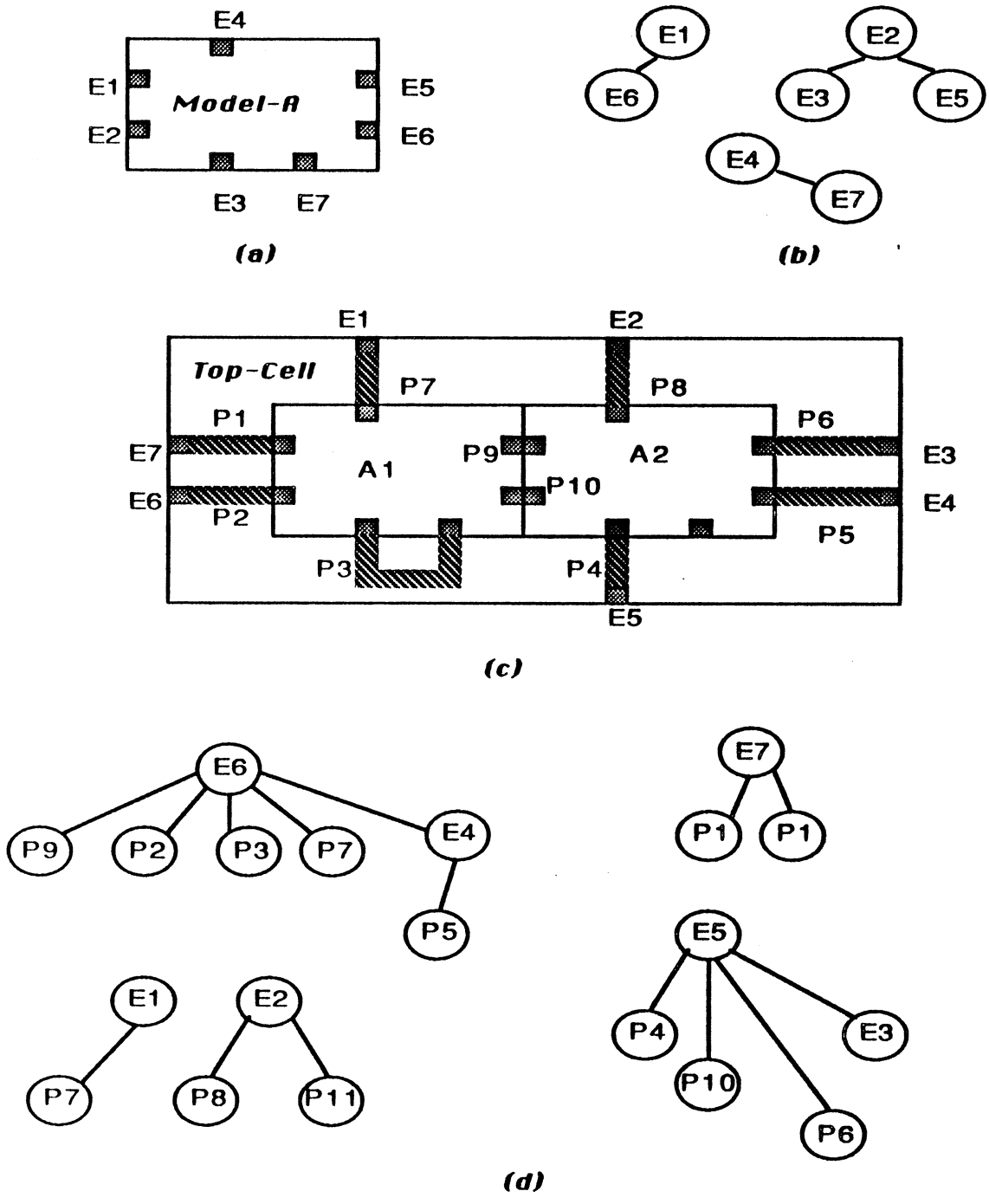


Figure 4 - ***Réalisation des équivalences électriques dans la hiérarchie de conception*** - *Les équivalences électriques des ports de chaque modèle appelé sont réalisées par l'action* ***union-find*** *entre leurs instances dans la cellule appelante. Ces équivalences électriques sont maintenues dans la base de données.*

(a) : cellule modèle Model-A; (b) graphe d'équivalence des ports de Model-A

(c) : Les sous-cellules A1 et A2 de Top-Cell sont des instances du modèle model-A; les équivalences des ports de Model-A sont "propagées" dans les contextes qui appellent Model-A, soit ici top-Cell; ce qui produit pour Top-cell le graphe (d). La méthode Fast -Union-Find a généré le graphe (d) correspondant à un certain ordre d'entrée des équivalences produites. La phase finale de compression compactera les arbres de F.G. et fera remonter E4 au dessus de E6.

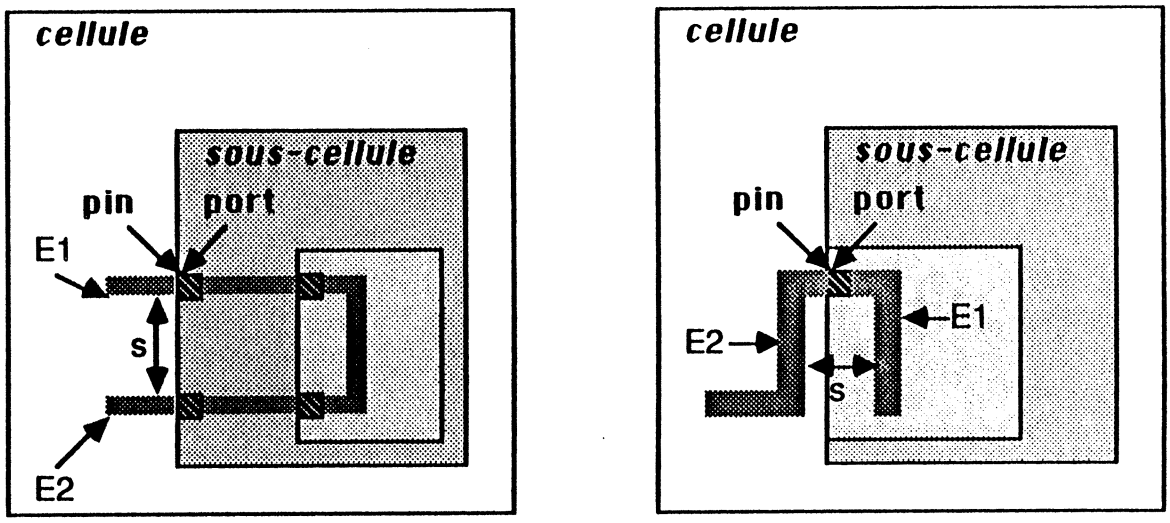


Figure 5 - ***analyse de la connectivité électrique à travers la hiérarchie du layout*** -

Cette analyse utilise le "réseau compact des nœuds électriques" et la recherche géométrique par quadtrees pour assurer une correspondance entre les nœuds électriques des divers modèles de cellule invoqués; cette correspondance est réalisée entre niveaux adjacents par la tranformation port -> pin ou son inverse et une recherche géométrique associée. Le "réseau compact des nœuds électriques" obtenu par la propagation hiérarchique - et ascendante - des équivalences de ports des divers modèles utilisés permet ainsi la reconnaissance des équivalences électriques entre des rectangles , tels que E1 et E2, à travers la hérarchie du layout.

(a) - l'équivalence électrique est reconnue immédiatement par l'action ***Union-Find*** *puisque E1 et E2 appartiennent à la même cellule et que toutes les équivalences ont été déjà propagées;*

(b) - l'équivalence électrique est analysé par la fonction ***représentant-nœud*** *à travers la hiérarchie du layout . (§2.5. & annexe 2.§2.5.). Cette reconnaissance évite de* ***signaler des fausses erreurs S*** *d'espacement entre des rectangles du layout appartenant à un même nœud électrique.*

Cette méthode est utilisée :

1. soit de manière ascendante - et alors la transformation port -> pin est utilisée, suivie de la **recherche géométrique exacte** de la pin dans la cellule appelante

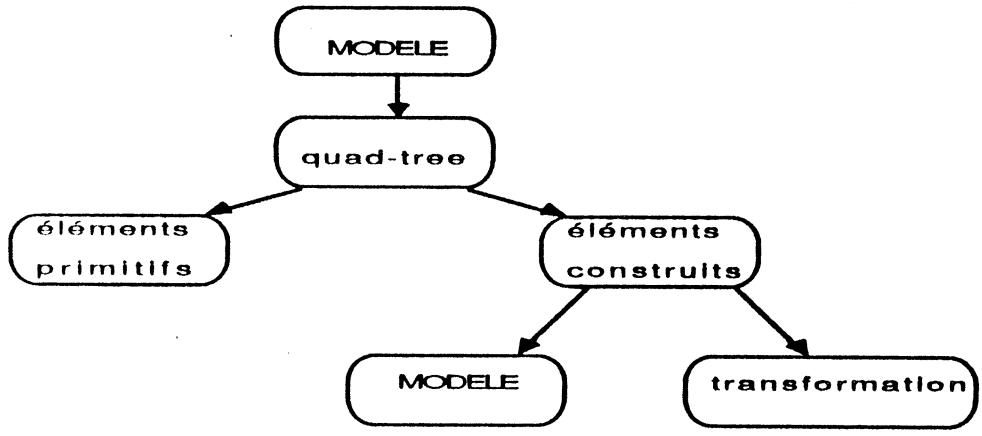
2. soit de manière desendante - et alors la transformation pin -> port, inverse de la transformation de la cellule en référence, est utilisée, suivie de la **recherche géométrique exacte** du port dans le modèle de cellule appelé.

Cette méthode est proposée comme base de l'analyse hiérarchique, ascendante ou descendante, de la connectivité électrique. Des algorithmes vont l'utiliser, et par récurrence entre les niveaux, analyser la connectivité électrique à travers toute la hiérarchie de conception. Ainsi les algorithmes **h-select-nom** et

h-spanner, qui vont être proposées en §2.4., selectionnent hiérarchiquement un nœud électrique. Ainsi la fonction **représentant-nœud**, qui sera proposée *en §2.5.*, étudie de manière ascendante l'équivalence électrique entre rectangles du layout plus ou moins enfoncés dans la hiérarchie de conception; la reconnaissance de ces équivalences évite ainsi à l'algorithme de vérification des règles de dessin, de signaler de fausses erreurs *(chapitre 6.§5.)*.

Cette méthode d'analyse hiérarchique de la connectivité électrique utilise des recherches géométriques, comme le fera aussi l'algorithme hiéarchique de vérification des régles de dessin. Ces recherches géométriques effectuées dans divers modèles de cellules de la hiérarchie du layout, utilisent la méthode de *recherche géométrique* proposée au *chapitre 4.* **L'interface utilisateur de la méthode de recherche géométrique par quadtrees,** qui a été alors proposée, permet de construire **un quadtree, pour chaque modèle de cellule intervenant** dans l'analyse. A cause du caractère incrémental de l'analyse hiérarchique, peu de cellules interviennent en général dans une seule analyse, et leurs quadtrees respectifs peuvent être générés au premier appel du modèle. On peut imaginer cependant tout au long du processus d'analyse hiérarchique qu'une hiérarchie de quadtrees est associée à la hiérarchie de conception du layout.

Une interface d'utilisation de quadtrees permet aux applications hiérarchiques de créer des **hiérarchies de quadtrees. Le quadtree associé à chaque modèle fait implicitement référence aux quadtrees des modèles de ses sous-cellules**. En effet les *éléments construits* (sous-cellules ou instances de symboles) sont placés dans le quadtree exactement comme des *éléments primaires* et font ainsi géométriquement référence aux modèles qu'ils instancient et en conséquence à leur quadtree associé.



Les quadtrees de cette forêt peuvent être créés ou supprimés par l'appel des actions / procédures de l'interface générale d'utilisation des quadtrees par les applications. *Un sommaire de cette interface est donné dans l'annexe 1.§.2.).*

2.4. L'accès géométrique à un nœud électrique en hiérarchie

Grâce à cette représentation compacte du réseau électrique associé à la hiérarchie de conception et à l'aide du système de recherche géométrique on analyse l'équivalence électrique de deux rectangles appartenant à différentes instances de modèles et différents niveaux de la hiérarchie de conception.

L'action **h-select-nom** réalise la **sélection d'un nœud électrique par son nom** dans une cellule hiérarchique. Cette selection s'effectue :

1. ***dans le niveau le plus haut de la cellule, par le nom du nœud électrique*** comme pour une cellule sans hiérarchie.

2. ***dans chaque sous-cellule au moyen de l'instance du port (pin) qui représente la partie instanciée du nœud recherché dans le modèle associé.*** L'instance de port et l'inverse de la transformation associée à cette instance permettent alors l'accès géométrique par **recherche géométrique exacte**, au *port* de la cellule et la *recherche par nom* des rectangles de cette cellule équivalents à ce port - si plusieurs ports sont équivalents, un seul est le représentant de la classe d'équivalence et donc du nœud électrique externe. ***Chaque rectangle ou atome ainsi sélectionné est alors repéré dans la cellule de plus haut niveau*** par sa transformation absolue. Le repérage absolu est mis à jour lorsque l'on descend dans la hiérarchie de conception par composition de la transformation absolue courante et de la transformation relative associée à la sous-cellule courante.

Cette **selection par nom** et **correspondance géométrique**, entre port et pin de deux niveaux adjacents de la hiérarchie, assure ainsi la récursivité du processus à travers la hiérarchie de la conception jusqu'à ce qu'il n'y ait plus de sous-cellule.

*Remarque : Lorsqu'un nœud électrique est connexe bien que "caché" en partie par la hiérarchie de conception, sa **sélection par recherche géométrique** est réalisée par l'action **h-spanner**. Comme pour les cellules sans hiérarchie cet algorithme utilise la technique "depth-first". Dans la technique "depth -first" il est necéssaire de "marquer" les éléments déjà selectionnés. Le marquage est réalisable par insertion des rectangles sélectionnés et transformés dans un quadtree spécifique. Il est ainsi possible de connaître par recherche géométrique si un rectangle est déjà trouvé par l'algorithme.*

*La **sélection par recherche géométrique d'une partie connexe d'un nœud électrique connexe** d'une cellule hiérarchique est réalisée :*

1. ***dans le niveau le plus haut de la cellule,** à partir d'un rectangle de départ du nœud électrique comme pour une cellule sans hiérarchie et la technique "depth-first" (chap2)*

2. ***dans chaque "sous-cellule atteinte" par la technique "depth-first"** au moyen de l'instance de port (pin) "atteinte" dans le modèle associé. L'instance de port et l'inverse de la transformation associée permettent alors l'accès géométrique au port du modèle et la recherche géométrique des rectangles du modèle de même nœud. Chaque rectangle ainsi sélectionné est alors transformé par le produit des transformations de la "pile de ses ancêtres" (repérage absolu) et **marqué** dans la cellule de plus haut niveau. Lorsque l'on trouve un port non marqué par la technique "depth-first" il faut alors faire la **recherche géométrique en amont** dans la hiérarchie et remonter dans le contexte appelant pour suivre de manière géométrique le nœud électrique; ce qui rend la **selection géométrique plus délicate à implémenter en hiérarchie** avec l'inconvénient de ne pouvoir obtenir ainsi que des parties connexes.*

2.5. L'équivalence électrique parmi les rectangles d'un layout structuré

Dans une analyse hiérarchique du layout qui vérifie les régles de dessin d'une conception il est nécessaire de reconnaître l'équivalence de deux rectangles assez proches l'un de l'autre pour produire une violation des régles, mais plus ou moins enfoncés dans la hiérarchie de conception.

Après l'extraction du réseau compact des nœuds électriques on reconnaît si deux rectangles sont équivalents en évitant la selection complète de leurs nœuds électriques respectifs. Lorsque ces deux rectangles appartiennent tous deux au niveau le plus élevé de la hiérarchie d'une cellule, on appelle **l'action UNION-FIND** pour le reconnaître. *(figure 5 - a).* Dans le cas contraire les deux rectangles sont équivalents s'ils appartiennent à un même nœud d'une cellule englobante qui est leur **plus proche ancêtre commun** dans la hiérarchie de conception. Il est possible de calculer pour chaque atome de la hiérarchie son **"représentant électrique de plus haut niveau"** appelé aussi **représentant électrique hiérarchique**; s'il est le même le deux rectangles sont équivalents *(figure 5 - b).* **L'action FIND** de la technique union-find est ainsi étendue à la hiérarchie de conception.

La fonction **représentant-nœud** *(annexe 2.§2.5.)* calcule le représentant électrique hiérarchique d'un rectangle en le remplaçant récursivement d'abord par son **représentant électrique local** et si cet équivalent est un **port** par **l'instance de port**, appelée **pin**, dans le contexte appelant. Le **représentant électrique hiérarchique** est atteint lorsque l'on atteint ainsi le haut de la hiérarchie ou lorsque l'équivalent obtenu devient interne à l'une ses **cellules englobantes** ou **ancêtres.**

Cette fonction **représentant-nœud** a comme **entrée un atome** et son **contexte hiérarchique** - i.e. la pîle du contexte d'appel ou **pîle d'ancêtres** constituée par toutes les références de ses sous-cellules englobantes. Elle a en **sortie** le **"représentant électrique de plus haut niveau"**, ou **représentant hiérarchique**, dans la hiérarchie c'est-à-dire un atome et son contexte hiérarchique - résidu de la pîle des ancêtres - *(annexe 2.§.2.5.).* Cette action recherche récursivement des équivalents electriques de plus haut niveau du rectangle en entrée par recherches :

1. de son représentant électrique local, racine de son arbre de Fischer-Galler. Si ce représentant électrique n'est pas un port alors <u>return</u> ...

2. de l'instance de port (pin) dans un contexte appelant. Cette recherche est une recherche géométrique exacte dans la cellule appelante - La transformation port ⇒ pin est alors utilisée (§2.3.) La fonction représentant-NOEUD alors est rappelée avec l'équivalent électrique ainsi obtenu et son contexte hiérarchique (*annexe 2.§2.5).*

Deux rectangles d'un layout structuré sont équivalents s'ils ont le même représentant hiérarchique. Pour le reconnaître, l'appel de la fonction **représentant-nœud** pour chacun des rectangles est alors nécessaire lorsque ces rectangles n'appartiennent pas au niveau le plus élevé de la hiérarchie de conception, sinon l'action **Union-find** reconnaît directement l'équivalence.

Deux rectangles d'un layout structuré sont équivalents s'ils ont le même représentant hiérarchique. Pour le reconnaître, l'appel de la fonction **représentant-nœud** pour chacun des rectangles est alors nécessaire lorsque ces rectangles n'appartiennent pas au niveau le plus élevé de la hiérarchie de conception, sinon l'action **Union-find** reconnaît directement l'équivalence.

2.6. La restriction topologique de la hiérarchie de conception

La restriction géométrique des hiérarchies de conceptions VLSI est réalisée implicitement par le simple fait que les recouvrements de géométrie dans les modèles obéissent aux régles générales de conception entre les atomes de géométrie qui décrivent l'ensemble des masques de ces conceptions. Mais dans une conception les recouvrements des sous-cellules ne sont pas à priori interdits puisqu'il est possible de vérifier les régles de dessin entre leurs éléments primaires à travers la hiérarchie de conception, au moyen des recherches géométriques hiérarchiques.

La restriction topologique des hiérarchies de conceptions VLSI est réalisée de manière explicite par la fonctionnalité des modèles. Il est ainsi interdit de créer de nouveaux transistors par le recouvrement des sous-cellules; un transistor ne peut être partagé entre deux sous-cellules...Ainsi apparaît clairement la notion d'interface des modèles, même lorsque cette interface paraît être définie par l'utilisation que l'on fait du modèle. Les **hiérarchies strictes**, dans lesquelles les données ne sont partagées qu'entre deux niveaux adjacents de la hiérarchie, ont été étudiées *au chapitre 2, §7.2.*

La non restriction du recouvrement des sous-cellules accroît la complexité de la vérification des régles de dessin (DRC hiérarchiques). La restriction topologique tend au contraire à diminuer la complexité des analyses hiérarchiques des propriétés électriques de la conception (ERC hiérarchiques). Dans l'analyse hiérarchique du layout ces activités sont complémentaires et permettent ensemble d'assurer un layout légal et sémantiquement correct.

Une cellule possède ainsi deux types de nœuds électriques, les nœuds électriques internes et les nœuds électriques externes. De manière induite, une cellule posséde deux types de rectangles, ceux qui appartiennent à des nœuds internes et ceux qui appartiennent à des nœuds externes. Les nœuds électriques externes sont les nœuds qui peuvent interagir dans un contexte appelant. Ils constituent ainsi l'**interface généralisée de la cellule** et sont sa partie externe. Les nœuds électriques internes ne peuvent pas interagir avec un contexte appelant sans violer les régles de connexion électrique et s'ils le faisaient l'extracteur de nœuds électriques ignorerait les équivalences électriques ou fusions de nœuds ainsi réalisées de façon illégale. Les nœuds électriques internes ne peuvent pas interagir avec un contexte appelant sans violer les régles de dessin entre atomes de géométrie car s'ils le faisaient le DRC hiérarchique signalerait alors des erreurs d'espacement entre des atomes qui n'appartiennent pas légalement au même nœud électrique.

Ainsi une cellule peut être vue comme formée d'une partie externe constituée par l'ensemble de ses nœuds externes et d'une partie interne constituée par l'ensemble de ses nœuds internes, même lorsqu'une partie de ces nœuds est

"cachée" par la hiérarchie de conception. Cette vue topologique et connectique de la cellule est différente de sa vue géométrique et hiérarchique *(figure 6).* Les analyses hiérarchiques qui extraient incrémentalement des **cellules Shells** représentent ainsi explicitement les parties externes des cellules hiérarchiques. Dans une *cellule Shell* [STE 84], [McCA 84], il n'y a pas de hiérarchie et la partie interne est vide; la cellule *Shell* représente exactement la partie externe ou interface généralisée de la cellule, et les vues topologiques et géométriques de ce type de cellule coïncident *(chap. 2.§7.1).*

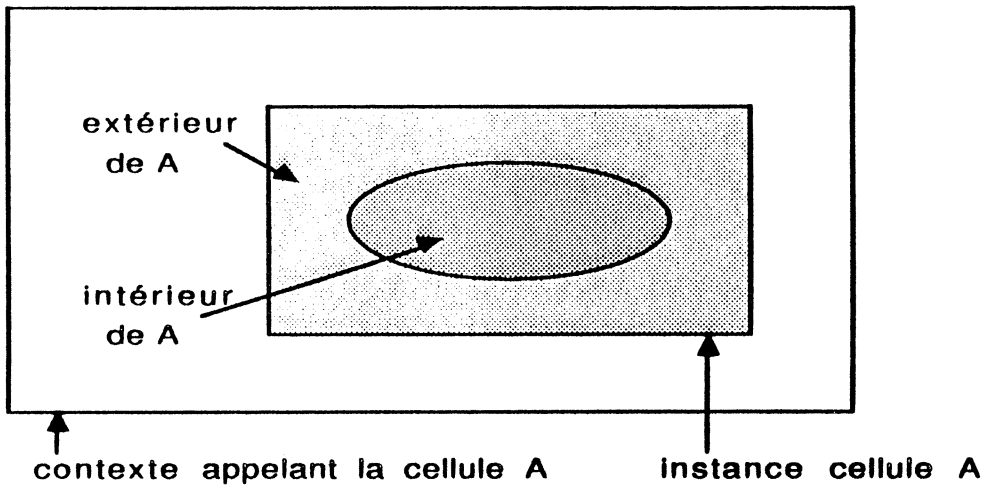


Figure 6 - Une Cellule A peut être vue comme formée d'une partie externe et d'une partie interne. La partie externe représente les nœuds électriques externes et la partie interne représente les nœuds électriques internes de A. Seule la partie externe de A peut intéragir avec le contexte qui l'utilise et produire des fusions de nœuds électriques (interface généralisée).

Notre approche de réduction de la hiérarchie est donc plus de nature topologique, connectique que géométrique. Le réseau compact de nœuds permet de reconnaître si un rectangle appartient à la partie externe de la cellule même s'il est profondément enfonçé dans la hiérarchie de conception. Ainsi la selection de la partie externe d'une cellule, proposée comme fonction d'édition, reconstitue de manière dynamique et algorithmique ***le Shell*** d'une cellule et assure un meilleur contrôle de sa connectivité à travers la profondeur de hiérarchie de conception *(Chap. 6.§.6.4.).*

3. L'EXTRACTION CONVENTIONNELLE DES PARAMETRES ELECTRIQUES

3.1. L'indépendance de la technologie

L'indépendance des algorithmes d'analyse du layout vis à vis de la technologie est réalisée par l'utilisation d'un paramètre technologique associé à chaque rectangle ou polygone simple d'un layout décrit physiquement ou symboliquement. Une couleur est ainsi associée à chaque rectangle. L'ensemble des couleurs ainsi que l'ensemble des relations électriques dans l'ensemble des couleurs sont alors

exprimés par des règles unaires ou binaires dans des fichiers technologiques et de manière interne par des tables de règles d'équivalence, de dessin, de paramètres électriques (unaires et binaires par rapport aux diverses couleurs). Les tables internes peuvent prendre en compte des couleurs de rectangles associés à des superpositions de niveaux physiques; des règles de garde et de connectivité peuvent être définies explicitement par les fichiers technologiques ou être déduites des règles plus élémentaires (entre niveaux physiques) que ces fichiers définissent. Ainsi les divers contacts et les transistors MOS de différents types seront directement pris en compte dans les tables internes auxquelles on se réfère.

3.2. Le calcul des paramètres électriques d'un nœud

La **capacité de surface** d'un nœud électrique par rapport au substrat est calculée par sommation des capacités de surface des divers matériaux utilisés. Son calcul est dérivé de celui de **l'aire** du nœud électrique; une pondération par unité de surface de la capacité par unité de surface des divers matériaux utilisés est prise en compte dans le calcul *(§4.2.3. du chap. 3).*

Le calcul de la **capacité de périmètre** d'un nœud électrique se calcule aussi par sommation des capacités de périmètre des divers matériaux utilisés. La valeur de la capacité par unité de longueur dépend des matériaux adjacents *(fig. 3).* Son calcul est dérivé de celui du **périmètre** du nœud électrique; une pondération par unité de longueur des capacité de périmètre des divers matériaux en interaction est prise en compte dans le calcul *(§4.2.3. du chap. 3).*

La **résistance** du nœud électrique est calculée en première approximation à partir de **l'aire** et du **périmètre** de nœud électrique *(§4.2.3. du chap. 3).*

Les calculs des paramètres électriques sont ainsi dérivés des calculs d'aire et de périmètre du nœud électrique à partir des rectangles ou atomes qui le constituent. Lorsque les rectangles du layout n'ont aucun recouvrement, l'aire d'un nœud électrique est alors la somme des aires de chacun des rectangles qui le composent. Lorsque les rectangles se recouvrent au moins partiellement, l'algorithme de calcul doit éviter de compter deux fois les aires de recouvrement des rectangles d'un nœud électrique.

Après la phase d'extraction des différents nœuds électriques, on calcule la surface d'un nœud électrique en parcourant l'ensemble des rectangles de la collection. Chaque rectangle de la collection est alors traité séquentiellement et l'intersection de ce rectangle courant avec les autres rectangles est étudiée par *recherche géométrique*; la partie d'aire de ce rectangle non couverte par un rectangle de même nœud déjà traité, donne lieu à une mise à jour de l'aire du noeud électrique associé. Après le traitement de tous les rectangles la surface de chaque nœud électrique est connue.

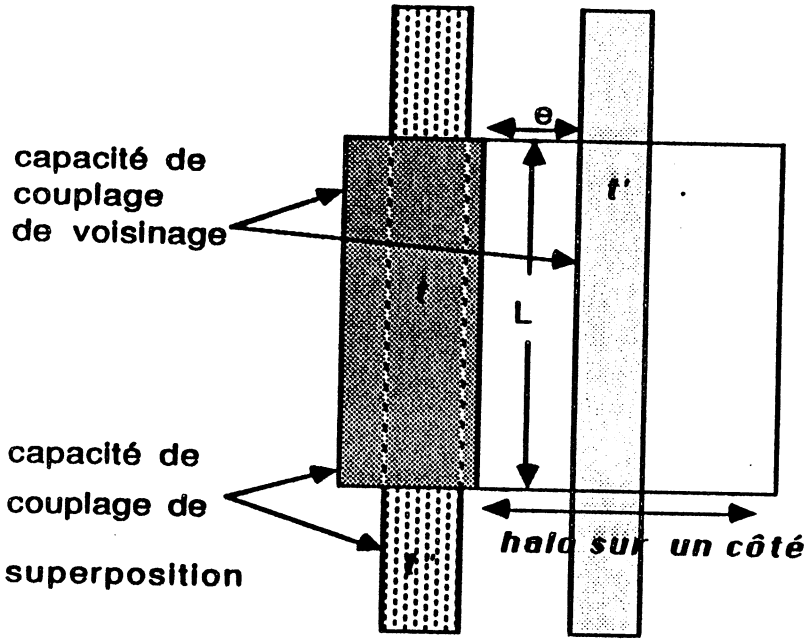
Le périmètre d'un nœud électrique se calcule de la même façon. La *recherche géométrique* et les *arbres de Fischer-Galler* permettent la sélection des rectangles adjacents d'un même nœud électrique; la reconnaissance de l'équivalence d'un rectangles adjacent avec rectangle traité est réalisée par l'intermédiaire de leurs arbres de *Fischer-Galler* respectifs *(chap. 3.§4.2.3.).*

3.3. L'extraction des transistors

Dans une technologie MOS, un transistor est supposé avoir une grille qui recouvre le canal, une source et un ou plusieurs drains connexes au canal. Les **terminaux d'un transistor sont la grille, la source et les drains**. Tous les rectangles qui forment le canal d'un transistor sont reconnus par l'algorithme de reconnaissance des divers nœuds électriques. Périmètre et surface du transistor sont aussi calculés de la même façon que pour un simple nœud électrique et permettent ainsi le calcul de ses dimensions électriques - largeur W et longueur L du transistor [MEA 81]. Une fois que les transistors sont extraits, la reconnaissance des rectangles adjacents au transistor détermine les différents nœuds connectés à leur source, à leurs drains et à leur grille *(chap. 3.§4.3.)*.

3.4. L'extraction des capacités entre nœuds

Des capacités de couplage internodales surviennent quand des connexions de différents niveaux de masques, appartenant à des nœuds électriques différents se recouvrent, ou quand des connexions appartenant à des nœuds électriques différents courent bord à bord *(figure 7)*.



Capacités de couplage d'un rectangle t du layout

Figure 7 - ***Extraction des capacités de couplage*** - *Extraction par recherche géométrique des capacités de couplage :*

- *de superposition (surfaces superposées)*
- *de voisinage (parties de périmètre en regard)*

Comme les capacités entre côtés parallèles diminuent rapidement, la recherche des éléments en interaction avec un côté de (t) est limitée à un halo de quelques lambda .

Après la phase de reconnaissance des nœuds électriques, le **calcul des capacités de couplage de recouvrement** est réalisé par la recherche des rectangles assez voisins d'un rectangle du layout pour provoquer une capacité de couplage non négligeable. Pour **chaque rectangle t** du plan de conception il faut :

a.1. rechercher géométriquement les recouvrements avec les autres rectangles

a.2. comparer le nœud électrique de chaque rectangle ainsi trouvé avec le nœud de t. S'ils sont différents, calculer une capacité de couplage entre ces deux nœuds (fonction des niveaux respectifs) des rectangles en interaction et de la surface réelle de recouvrement

a.3. pour être correctement calculée, la surface réelle de recouvrement doit être ajustée par la déduction des participations possibles des rectangles du même nœud électrique que t et déjà traités (chevauchements de rectangles).

Les **capacités de couplage entre côtés parallèles** sont calculées en **recherchant d'abord géométriquement les éléments des quatre "halos"** (zones d'interaction) **autour de chaque rectangle t**. Un ajustement du calcul de l'interaction de t avec un autre rectangle doit être aussi réalisé par la déduction des capacités internodales déjà comptées pour des rectangles t" équivalents à t et le couvrant au moins partiellement *(fig. 8 du chap. 4.§4.4).*

L'ajustement des capacités de couplage est ainsi assez complexe lorsque les rectangles qui composent les différents nœuds électriques ont des recouvrements. Un ajustement global du calcul des capacités internodales peut être réalisé selon l'algorithme **b** :

b.1. calculer les capacités internodales comme si les divers rectangles qui composent un nœud électrique n'avaient pas de recouvrement, par un traitement séquentiel de l'ensemble des rectangles de la cellule comme précédemment *(selon les étapes a.1 & a.2.)*

b.2. parcourir l'ensemble des rectangles en recherchant leurs intersections géométriques; calculer ainsi les capacités internodales des toutes les intersections de rectangles de même nœud électrique.

b.3. soustraire les résultats des calculs de capacités internodales faits en b.1. et b.2.

Les calculs d'aire et de périmètre des divers nœuds électriques sont ajustables globalement par une méthode semblable. Ce mode de calcul et d'ajustement global est ainsi applicable au calcul des **paramètres électriques nodaux** et **internodaux**.

4. L'EXTRACTION HIERARCHIQUE DES PARAMETRES ELECTRIQUES

Quand une cellule contient des sous-cellules, son circuit total est plus que l'union des circuits de ses sous cellules et de la géométrie de ses masques. L'extracteur doit faire les connexions entre les nœuds de ses sous cellules quand ils se superposent ou se juxtaposent. Il peut avoir besoin aussi d'ajuster les capacités parasites lorsqu'une description électrique détaillée est nécessaire.

Dans une cellule hiérarchique, imaginons que nous désirions encore calculer simplement le périmètre et la surface des nœuds électriques résultant de l'interaction de sous-cellules. Certains côtés du périmètre de chacun des nœuds en interaction peuvent disparaître comme résultante de l'interaction (juxtaposition ou recouvrement). Ainsi, la capacité de périmètre associée à ces côtés devra-t-elle être soustraite de la somme des capacités de périmètre des deux terminaux, exactement comme cela est fait pour un ensemble de rectangles ayant entre eux des intersections non vides. La seule différence est que le calcul du périmètre ou de la capacité de périmètre, est ici le résultat d'un calcul incrémental fait en deux temps :

1. extraction des paramètres des modèles appelés
2. extraction des paramètres de la cellule de plus haut niveau

En conséquence, le calcul des paramètres électriques (résistances et capacités) ne peut être entièrement refait pour chaque interaction de nœuds de sous-cellules et doit être sauvegardé pour les divers nœuds des divers modèles, indépendemment de la manière dont ce contexte (cellule appelante) les utilisera. De même, il est intéressant que les rectangles susceptibles d'intervenir dans une interaction soient préalablement connus ou reconnus et constituent l'interface des sous-symboles *(chap. 2.§7.1).* Les éléments terminaux des modèles sont donc porteurs de l'information géométrique et topologique (électrique) ainsi que de l'information de dimension (*surface* et *périmètre* pondérés par la technologie, du nœud électrique associé, pour le calcul incrémental des diverses résistances et capacités parasites).

Comme pour de simples rectangles du layout, deux ports disjoints peuvent être électriquement équivalents et la relation d'équivalence dans l'ensemble des ports doit être *résumée* lors de l'extraction du graphe électrique symbole. Sa sauvegarde comme un attribut associé à chaque terminal est alors réalisée dans la base de données des circuits.

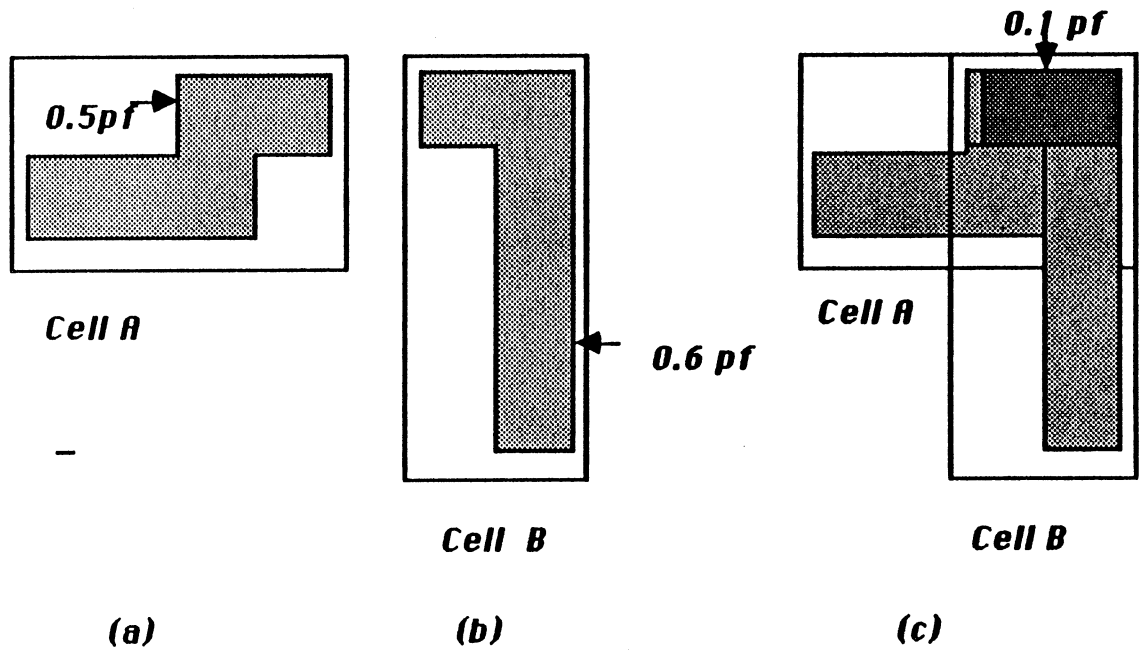


Figure 8 - ***Interaction de sous cellules et calcul des capacités de surface et de périmètre*** - *La somme et le périmètre des noeuds électriques résultant de l'interaction n'est pas la somme et le périmètre des noeuds électriques des sous cellules ainsi "instanciés" et unifiés ; le détermination du recouvrement des surfaces et périmètres des noeuds ainsi unifiés permet d'ajuster les capacités et résistances des noeuds résultants dans la cellule qui les a appelés. Les accès géométriques sont encore invoqués dans cette cellule.*

Ainsi, le processus d'extraction hiérarchique d'un circuit est rendu incrémental par l'extraction de tous les symboles créés indépendamment de la manière dont ils pourront être utilisés dans la hiérarchie. En conséquence de cette indépendance, les ajustements des connexions entre les ports de sous-cellules et aussi entre ports de sous-cellules et éléments primaires du layout de la cellule, devront être faits dans la cellule appelante. Les compensations seront faites pour le calcul des différents paramètres électriques nodaux et internodaux, exactement comme pour de simples rectangles ayant entre eux des intersections non vides de surface ou de périmètre *(§4.2.3. & §4.4. du chap. 3 - §3.2. et §3.4 du chap. 5).*

Les pins, instances de ports des sous-cellules, sont alors considérées comme de simples éléments primaires ou rectangles du layout; ces éléments particuliers sont porteurs d'une information d'équivalence électrique et éventuellement d'une information de dimensions du nœud électrique; la compensation des superpositions ou juxtapositions avec les autres instances de ports est faite lorsque l'on traite séquentiellement chaque rectangle du niveau le plus haut de la cellule extraite *(figure 8).*

Le calcul le plus complexe est alors celui des **capacités internodales** qui, lorsqu'elles sont nécessaires, exigent l'**analyse complète des interactions des sous-cellules** entre elles et avec les éléments de base du niveau le plus haut,

exactement comme pour le D.R.C *(chapitre 6.§5.4.)* à travers la hiérarchie de conception. L'algorithme général d'étude de ces interactions est alors celui du DRC hiérarchique avec l'analyse complète du voisinage (halos) de chaque élément, simple rectangle ou sous-cellule, qui compose la cellule extraite. Ce type de compensation est en général assez faible pour être négligé en première approximation [SCO 85] ; les modifications apportées alors aux capacités internodales restent dans la plupart des cas très faibles (sauf pour des bus partagés).

4.1. L'exploitation des répétitions

Après l'extraction d'un modèle de cellule, la relation d'équivalence dans l'ensemble de ses ports est connue, ainsi que les paramètres électriques des différents nœuds électriques externes auxquels ces ports appartiennent. Le maintien de ces informations topologiques et électriques permet de les instancier dans les contextes appelants, autant de fois que le modèle est utilisé.

Par exemple, l'extraction de tableaux de symboles (matriçage) est réalisée en trois étapes :

1. l'extraction du sous-symbole matricé,
2. la réalisation des instances géométriques et structurelles des ports des sous-cellules dans le plus haut niveau de la cellule,
3. l'extraction de l'ensemble des "rectangles" du plus haut niveau de la cellule (rectangle et instances de ports).

Ce qui constitue un gain de temps certain par rapport à une extraction conventionnelle comme cela a été démontré *au §6.2. du chapitre 2.*

4.2. Incrémentalité des extractions hiérarchiques

L'incrémentalité des extractions garantit que lorsqu'une cellule est modifiée ses sous-cellules n'ont pas besoin d'être réextraites. Cependant, si une sous-cellule change, il faut reextraire tous les parents qui l'instantient et les parents de ces parents récursivement. Une trace des dates de dernière extraction et de modification permet de reconnaître quand une cellule et ses ancêtres doivent être reextraits.

L'extraction d'une cellule nécessite alors :

1. l'extraction des modèles appelés,
2. l'ajout des instances des ports (pins) des sous-cellules et la réalisation de leurs équivalences dans le plus haut niveau de la cellule,
3. l'extraction de l'ensemble des "rectangles" du plus haut niveau de la cellule (rectangles primaires et pins).

Lorsque l'extraction est incrémentale l'étape 1 est réduite à une comparaison récursive des dates des sous-symboles.

4.3. Faisabilité de l'analyse incrémentale et hiérarchique

L'algorithme conventionnel *union-find* de gestion de la connectivité électrique peut ainsi être utilisée extensivement, de manière efficace, pour l'analyse hiérarchique et incrémentale. En effet, la complexité théorique de l'algorithme de base, qui recherche les contacts électriques de manière géométrique, est, sur un layout constitué de n éléments, entre O (n log n) et O (n $\log^2$ n), pour n *recherches ponctuelles (pics)* et en pratique de l'ordre de $O(n^{1.1})$ pour n recherches géométriques d'intersections ou de juxtaposition *(chapitre 4).* La complexité théorique de l'algorithme *Union-find* est considérée inférieure à O (n log n) pour n recherches d'équivalence sur ce layout *(chap. 3).* Ainsi l'analyse conventionnelle est réalisée en un temps de complexité en pratique très inférieur à O ($n^{3/2}$). Le nombre d'*éléments primaires* - ports de la cellule - à réanalyser, lorsqu'une cellule de n éléments primaires est instanciée, est au plus d'ordre $O(n^{1/2})$, si on considère ce nombre au plus proportionnel au périmétre du modèle. La combinaison de ces propriétés assure une analyse incrémentale performante puisque le produit de ces complexités est inférieur à 1 *(cf. §6.2 et §8. du Chap. 2).* La **faisabilité de la méthode** est ainsi assurée avec une complexité des temps cumulés de **l'analyse incrémentale** au plus en O (N) pour un circuit de N éléments, et un nombre total d'éléments à considérer dans l'analyse de la cellule de plus haut niveau, qui représente le circuit complet, au plus d'ordre $O(N^{1/2})$.

Chapitre 6

APPLICATION AUX FONCTIONS DE LA CAO

1. INTRODUCTION

Les méthodologies de conception des circuits VLSI en usage aujourd'hui ont, en ce qui concerne les vérifications, certains défauts. La plupart reportent les vérifications des règles de conception et les tests de connectivité électrique jusqu'à la fin du processus de conception. Ceci est nécessaire puisque la possibilité de recouvrement d'un élément du layout avec un élément déjà vérifié, enfoncé dans la profondeur de la hiérarchie de conception, est en général impossible à découvrir. La reconnaissance de l'équivalence électrique de deux éléments enfoncés ou non dans cette hiérarchie, est elle aussi difficile.

Les méthodes de recherche géométrique et d'équivalence proposées dans notre système de CAO nous permettent de proposer un **algorithme général d'analyse de layouts structurés** *(figure 2).* Cet algorithme général implémente en différentes phases les divers algorithmes proposés aux *chapitres 4 et 5.*

L'intérêt de ces méthodes pour les fonctions d'édition est aussi étudié. La prise en compte de la gestion dynamique des données a constitué un facteur de choix des algorithmes de base de recherche géométrique et d'équivalence. Ainsi diverses phases de l'algorithme général d'analyse du layout doivent pouvoir être réalisées de manière dynamique et interactive. Des algorithmes sont aussi proposés comme des fonctions de contrôle du layout édité. Les layouts édités sont de plus en plus flexibles et il faut ensuite adapter ces layouts à une technologie [ROG 85][OUS 84][ENT 85]. L'aide apportée par les méthodes proposées à l'édition de ces layouts flexibles est ainsi étudié; des algorithmes sont ainsi proposés pour adapter ces layouts à une technologie.

La *figure 1* présente l'algorithme général d'analyse du layout d'un circuit. Diverses phases de cet algorithme vont être étudiées dans la suite de ce chapitre. Ces diverses phases utilisent, comme algorithmes de base, les algorithmes de recherche géométrique et d'équivalence proposés *aux chapitres 4 et 5.*

Algorithme général d'analyse du layout d'un circuit -
Entrée : description géométrique du circuit;
Sortie : circuit extrait et vérifié -

1. ***Application de régles de réécriture selon la méthode de description du layout utilisée***
2. ***Génération d'une représentation compacte du réseau des nœuds électriques***
3. ***Vérification des règles de dessin ou D.R.C.***
4. ***Extraction et propagation des paramètres électriques***

Figure 1 - ***Algorithme général d'analyse du layout d'un circuit*** -
Les phases 1 et 2 utilisent les algorithmes et méthodes proposées aux chapitres 4 et 5. La phase 3 est constituée par l'algorithme de DRC qui sers présenté dans ce chapitre. Elle utilise la représentation compacte du réseau des noeuds électriques , et la méthode de recherche géométrique pour vérifier le layout ainsi décrit hiérarchiquement.

2. PARAMETRAGE PAR LA TECHNOLOGIE

Un paramètrage par la technologie a été recherché lors de la conception des algorithmes d'analyse et vérification hiérarchique du layout. Le paramètrage technologique est cependant hors du champ de cette étude et sa présentation est conceptuelle. Cette maquette d'implémentation analyse actuellement des layouts édités selon une **méthodologie STICK** *(Annexe 4)* [LEC 84], [MEA 80]. Le paramètrage technologique des règles de conception pour un ensemble de couleurs ou niveaux de base, et de couleurs réécrites, ou multi-niveaux, est réalisé par des tables de règles symboliques.

L'indépendance des algorithmes vis à vis de la technologie est ici réalisée par des fichiers technologiques qui définissent l'ensemble des couleurs (simples ou multiniveaux), l'ensemble des conditions qui paramètrent les règles et leur relation d'ordre, l'ensemble des règles. Ces fichiers technologiques sont représentés de manière interne par des hypermatrices (2D / 3D - matrices) *(figure 2)*. Pour l'ensemble des régles, des options générales sont choisies par défaut et seules les autres options sont à exprimer dans les fichiers technologiques sous une forme équivalente à **<couleur> * <couleur> [*<cond>] -> <valeur>**;

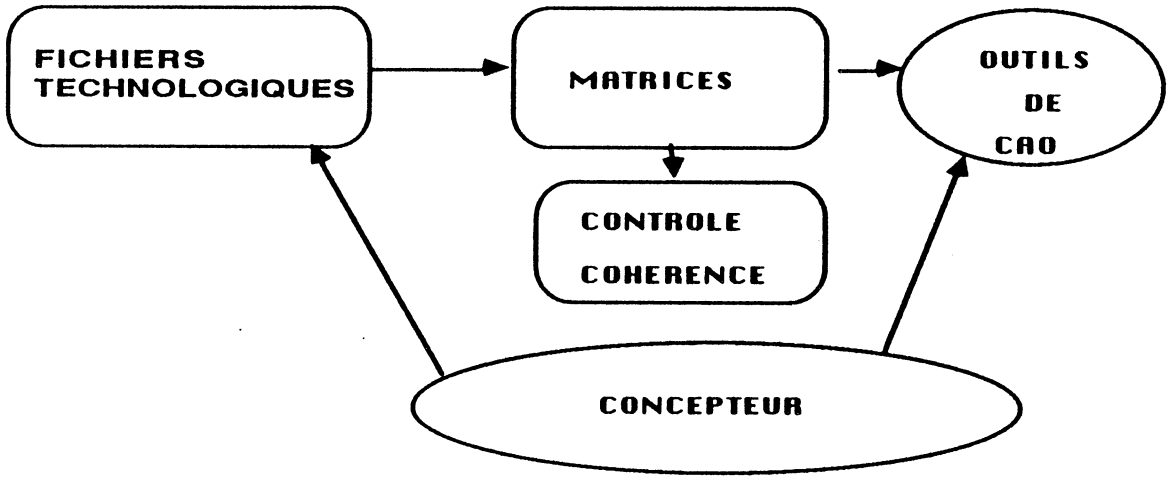


Figure 2 - **Les fichiers technologiques** - *Ils définissent les différentes* ***couleurs*** *utilisées par la technologie (niveaux physiques et pseudo-niveaux).*

- Un fichier technologique exprime les règles de réécriture éventuelles de la description des masques : lois de composition des couleurs actives, interdiction de chevauchement de couleurs actives, non interaction de couleurs en superposition

- Un 2e fichier technologique exprime les règles d'équivalences entre les couleurs d'objets ou motifs d'objets juxtaposés ou superposés, règles générales appliquées par le concepteur de circuit pour exprimer par une connexité géométrique la connectivité électrique.

- Un 3e fichier technologique exprime les règles de garde géométriques des divers motifs du layout : largeur minimum des motifs, espacement minimum des motifs ayant entre eux une contrainte d'espacement, recouvrement minimum, débordement minimum des motifs ayant entre eux une contrainte de connectivité.

Options par défaut - *initialisation des relations :*
- *pas de réécriture ni d'interdiction de chevauchement*
- *pas d'équivalence électrique entre couleurs* ***différentes***
- *pas de règle de garde entre motifs*

Le concepteur doit exprimer un ***ensemble minimal de règles*** *(réécriture, équivalence , gardes). La cohérence de ces règles est vérifiée par l'interface GARDE afin d'obtenir un ensemble cohérent de règles technologiques.*

Règles de réécriture *: < couleur> * <couleur> * [<**condition**>] —> <couleur>*

Règles d'équivalence : *<couleur> * <couleur> —> <valeur> ; valeur : -1..1 ;*

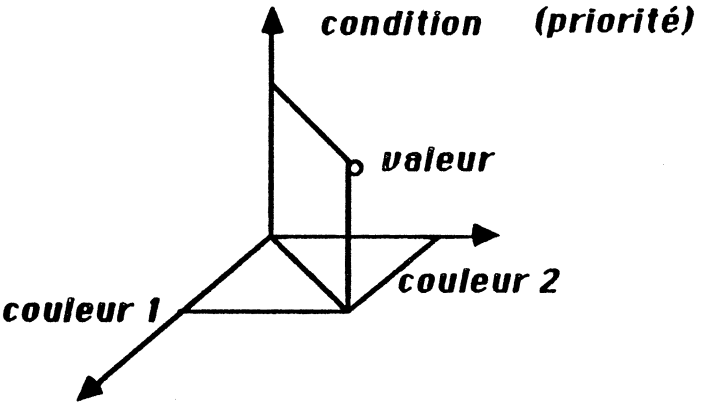
Règles d'espacement *: <couleur> * <couleur> * <**condition**> --> <valeur>;*
valeur : réel; un élément multineaux utilise plusieurs couleurs adjacentes;

Règles de connectivité *: <couleur> * <couleur> * <**condition**> --> <valeur>;*
(recouvrement et débordement) valeur : réel;

Priorités *: Les* ***conditions*** *qui paramètrent ces règles ont entre elles une relation d'ordre qui les rend non ambiguës. Elles sont de nature géométrique ou électrique et ces conditions ainsi que leur relation d'ordre est exprimée dans un fichier technologique.*

Règles unaires {de largeur}:
*<couleur> * <condition> * {minimum , maximum} --> <valeur>;*

Remarque *: recouvrement sans interaction et violation de recouvrement sont exprimées par des pseudo-couleurs.*



Représentation interne des règles technologiques *: Des* ***matrices*** *ou* ***hypermatrices*** *représentent de manière interne des fichiers technologiques. Ils sont chargés dynamiquement par les processeurs qui les utilisent. La non isométrie des gardes d'espacement du* ***symbolisme Stick*** *(Annexe 4) est représentée par l'utilisation de 2 couleurs contiguës dans les entrées de la matrice permettant ainsi d'exprimer des contraintes d'espacement différentes suivant l'axe et suivant le bout de l'élément ayant ces couleurs.*

Ces ***hypermatrices*** *sont en fait gérées de manière interne comme de simples vecteurs (1D), c.a.d. en évitant les redondances de symétrie des règles. Une fonction d' accès calcule alors l'adresse A (couleur1, couleur2) dans le vecteur des valeurs associées aux diverses conditions ou prédicats (géométriques ou d'équivalence).*

Langage de description *- Un langage de description [SMI 85] peut être utilisé pour décrire complètement la technologie. Les différents niveaux et pseudo-niveaux, les différents transistors et contacts, les différentes règles de conception et la translation des symboles sont ainsi décrits.*

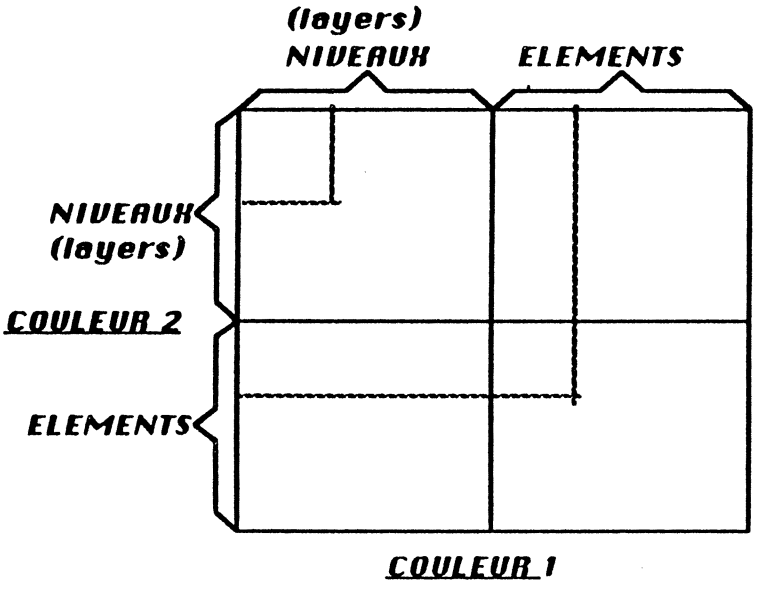


Figure 2 (suite) - ***Des niveaux physiques et des éléments multiniveaux sont utilisés .***

3. L'ANALYSE HIERARCHIQUE DU LAYOUT ET LA STRUCTURE DE DONNEES

3.1. L'analyse incrémentale et l'optimisation des algorithmes

Les algorithmes implémentés analysent la hiérarchie de conception de manière ascendante et commencent par analyser les cellules les plus imbriquées. Lorsque la réalisation d'une phase de l'algorithme général d'analyse du layout est finie, la cellule analysée est marquée dans la base de données. Le démarquage des cellules modifiées provoque la revérification automatique d'une partie de la hiérarchie de bas en haut. La technique récursive utilisée est naturelle. Elle consiste à extraire / traiter récursivement tous les symboles imbriqués préalablement au traitement du layout du symbole en cours d'extraction à un instant de l'algorithme et à les marquer une fois traités. **L'analyse incrémentale des cellules éditées et leur marquage, constitue donc un aspect important de l'optimisation de ces algorithmes**. Ainsi l'analyse d'une cellule du layout est réalisée en deux temps:

1. ***Les modèles de cellule utilisés par la cellule sont analysés s'ils le ne sont pas déjà***
2. ***La cellule est analysée sur son niveau le plus élevé et le résultat de l'analyse effectuée pour les sous-cellules est utilisé***
3. ***La cellule est marquée, date et type de traitement sont ainsi reportés avec les résultats de l'analyse***

3.2. La structure de donnée implémentée

Une structure de donnés prototype constituée de rectangles colorés, imbriqués en une **hiérarchie canonique** - sans rotation ni réflexion - a servi à une première implémentation des divers algorithmes d'analyse de layouts. Cette structure de données très simple a été ensuite enrichie afin de supporter l'implémentation des algorithmes sur des cellules imbriquées en une hiérarchie non canonique dans laquelle les symboles appelés subissent rotations et réflexions. **La méthodologie STICK** *(Annexe 4)* [MEA 80] utilisée pour l'édition de ces cellules a été prise en compte, mais d'autres méthodologies doivent pouvoir être aussi prises en compte par cette structure de données très générale. Les algorithmes proposés sont absolument indépendants de la base de données choisie pour le stockage des circuits et leur généralité permet de les adapter à d'autres méthodologies de conception. La structure de données utilisée de manière interne par les algorithmes est générée par un module d'interface avec la base de données CASSIOPEE [LEC 84], [BEY 82], afin de tester et mesurer l'efficacité des algorithmes sur des circuits existants. Le schéma de cette structure de données est présenté ci-après et une ***description en pascal est donnée dans l'annexe 2 - §1.***

La structure de données interne est une **bibliothèque formée de** trois ensembles :

1. **l'ensemble** (table) **des modèles de cellule / structures / symboles,**
2. **l'ensemble** (table) **de leurs interactions relatives dans les diverses instances de ces modèles**. *Les interactions des modèles de cellules sont vérifiées par le D.R.C. qui utilise la même structure générale de données*

(§5.1.).- Deux instances de cellules sont ***en interaction*** *si elles sont suffisamment proches pour causer une violation des règles de dessin du layout.*

3. l'ensemble (table) **de tous les éléments utilisés par tous les modèles considérés.**

<bibliothèque> ::= (<ensemble-de-structure>
<ensemble-d'interactions-de-modèles>
<ensemble-d'-élément>)

Structures / modèles et interactions de structures / modèles sont cherchés par hash-coding sur le numéro d'identification (nom) d'un des modèles dans leur table respective.

Chaque **élément** est accessible par son index / nom (entier > 0) dans la table des éléments. C'est **à priori un rectangle** ou un **atome** de géométrie d'une structure / modèle. Il est **soit une partie d'un dispositif électrique** (connexion, contact, transistor), **soit le rectangle enveloppe d'une instance de structure;** dans le premier cas c'est un **élément primaire**, dans le second cas c'est un **élément structuré** appelé aussi **élément construit.**

Chaque structure / symbole est formée d'éléments.

Chaque élément a un :

1. **<type>** ::= **<primaire> / <structure référencée> / <pin>,** chaque élément est soit un élément primaire, soit un élément construit c.a.d. une sous-cellule ou structure référencée
2. **<nom>, un nom / index** (entier) **de l'élément** dans la table des éléments
3. **<dad>, nom / index** (entier) **de son père (élément) au sens des arbres de F-Galler** dans la table des éléments. Ainsi, l'ensemble des arbres de Fischer-Galler créés par les multiples appels de l'action UNION - FIND sont repésentés par des **arcs (dad) dans l'ensemble des éléments**
6. **<father>, nom / index** (entier) de son **père hiérarchique** (symbole qui le crée, l'instancie)
5. **<bbox> ::= (<x1><x2><y1><y2>)** un **rectangle enveloppe minimal**
7. **<color>, une couleur** (type ordinal), paramètre technologique
8. **<dimensions>, une dimension** pour le calcul des paramètres électriques
9. **<refb>,** des références d'accès à la base de données (adresse)
10. **<erreur>,** éventuellement un marquage d'erreur (booleen), (ERC / réécriture...)

Chaque élément de type construit ou de type pin a un :

11. **<psref>,** référence /identificateur (entier) de la structure ainsi appelée / instanciée
12. **<topo>** ::= **<transformation>** adresse de la transformation opérée sur la structure en référence, exprimée relativement à la structure qui l'appelle / instancie (i.e. la plus interne)

<élément>::=
(<type><nom><dad><father><bbox><color><dimensions><refb>
<erreur><champ variable selon type>);
<champ variable selon type> ::= (<psref><topo> / < champs variable selon type>)

Chaque structure / symbole a un :

1. **<nom>**, nom / identificateur (entier)
2. **<unité>, une unité** (en microns)
3. **<bord>** ::= <liste de points>, peu utile, la structure est définie par ses éléments
4. **<bbox> ::= (<x1><x2><y1><y2>, un rectangle englobant,** (4 reels)
5. **<quadtree>, quadtree associé** (entier >0) pour les recherches géométriques
6. **<refb>**, des références d'accès à la base de données
7. **<ensemble-de-FLAG>**, divers marquages des traitements réalisés
8. **<ensemble-ELEMENT>**, ensemble d'éléments - intervalle dans table -
9. **<ensemble-ERREUR>**, un ensemble d'erreurs, (liste d'éléments en erreur)
10. **<graphe électrique>**, constitué des listes des représentants des nœuds électriques externes et internes, et de la liste des transistors.

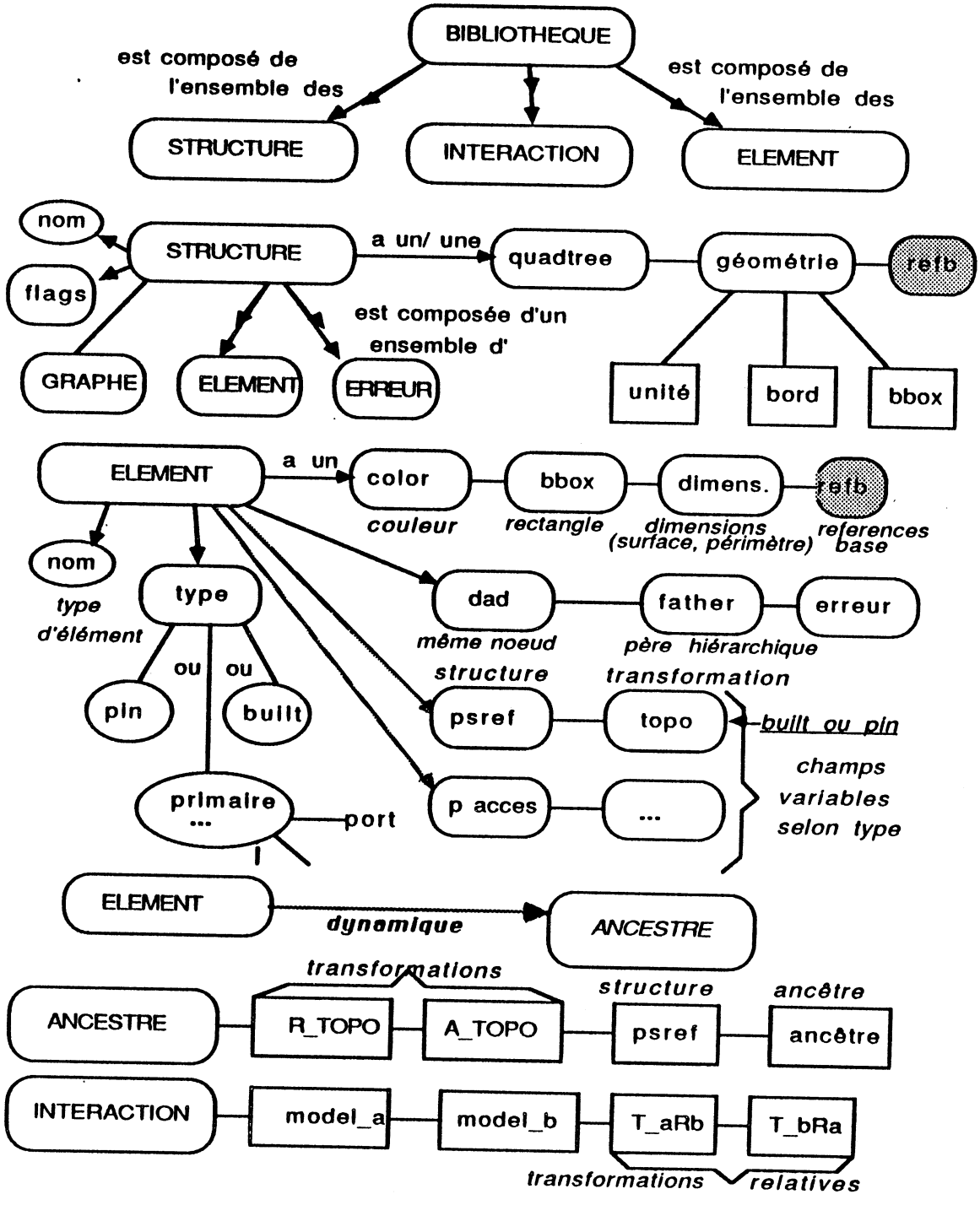
<structure> ::= (<nom><unité><bord><bbox><quadtree><refb>
<ensemble-de-FLAG><ensemble-ELEMENT><ensemble-ERREUR>)
<graphe électrique>

<ensemble-ERREUR> ::= <nil> / <erreur><ensemble-ERREUR>
<erreur> ::= (<nom><transformation>); nom de l'élément, transformation *(DRC)*
<nom> :: = <entier>; <transformation> ::= (<A><B><C><D><E><F>)
A, B, C, D, E, F : sont les coefficients reels et variables de la matrice associée *(§5.4.2)*
<topo> ::= <pointeur>, adresse de la tranformation
<graphe électrique>:::=<nœuds-externes><nœuds-internes>
<transistors>
<nœuds-externes>::=<nil> /<racine-nœud> <nœud-externes>
<nœuds-internes>::=<nil> /<racine-nœud> <nœud-internes>
<transistor> ::= <nil> / <racine-nœud> <liste-terminaux>

Pour une application comme le DRC hiérarchique *qui sera étudié au §5.1. chaque élément a une pile d' ancêtres; ce lien est dynamique et représente le contexte d'appel hiérarchique de l'élément*
<liste d'ancêtre>::= (<r_topo><a_topo><psref><liste d'ancêtre>)
<r_topo> ::= <transformation>; <a_topo> ::= <tranformation>
<psref> ::= <entier>; *la structure est accesssible par hash-coding dans la table des structures*
psref est le nom / index de la structure en référence (appelée), r_topo est sa transformation par rapport au repère lié à la structure qui l'instancie (la plus interne), a_topo est sa transformation absolue calculée dans le repère lié à la structure analysée de plus niveau. Les ancêtres de la liste (pile) sont empilés lorsque l'on descend dans la hiérarchie de la structure de plus haut niveau et dépilés lorsque l'on remonte dans la hiérarchie.

Pour une application comme le DRC hiérarchique, *les occurrences d'une même interaction relative de deux instances de structures doivent être reconnues*

<interaction> ::= <model_a><model_b><T_aRb><T_bRa> avec
<model_a> ::= <psref>; <model_b> ::= <psref>;
T_aRb ::= <transformation>; T_bRa ::= <transformation>;
model_a et model_b sont les noms / index des structures en interaction (DRC);
<T_aRb> est la transformation relative de la structure A par rapport au repère lié à B
<T_bRa> est la transformation relative de la structure B par rapport au repère lié à A
Ces 2 transformations sont inverses l'une de l'autre (§5.4.1)



Une description plus complète de cette structure en pascal est donnée dans *l'annexe 2.§1.* Le schéma ci-dessus donne une vue générale de cette structure de données. Les doubles flèches caractérisent les ensembles (set codasyl).

4. L'EXTRACTION DES NŒUDS ELECTRIQUES

4.1. Extracteur de graphe, principe

Le concepteur édite les symboles ou modèles utilisés dans sa conception en appliquant certaines règles graphiques, de juxtaposition ou superposition, pour réaliser transistors, nœuds équipotentiels et connexions aux ports des sous-cellules ou transistors utilisés. Une cellule est vue comme décrite par un ensemble de rectangles ou **éléments primaires**, et de sous-cellules ou **éléments construits**.

Chaque cellule est extraite indépendamment des contextes dans lesquels elle sera utilisée. La structure de chaque cellule est préservée par l'extracteur; elle est complétée par son graphe d'équivalence électrique. Un graphe d'équivalence est réalisé - par la construction d'arbres de Fischer-Galler - entre ses *éléments primaires;* les équivalences entre les instances de ports des sous-cellules sont aussi propagées dans ce graphe *(chapitre 5).* Les listes des nœuds electriques externes, des nœuds internes et des transistors réalisent alors **un résumé** de la cellule. Un algorithme récursif d'instanciation du graphe électrique permet ensuite, si nécessaire, la génération d'un graphe sans hiérarchie à partir de cette représentation compacte du réseau des nœuds électriques du layout. Ce graphe pourra servir d'entrée aux divers simulateurs.

4.2. Reconnaissance des entités électriques

4.2.1. Règles de réécriture

Un simple algorithme de réécriture du layout édité résoud alors les intersections géométriques des *éléments primaires* ayant des couleurs susceptibles de provoquer des interactions, ce qui permet la reconnaissance éventuelle des transistors ou des violations de recouvrement *(chap. 3.§.4.3.).* Un fichier technologique exprime la loi de composition pour la réécriture dans l'ensemble des couleurs ainsi que leur absence d'interaction ou leur interdiction de recouvrement *(figure 2).*

4.2.2. Règles d'équivalence électrique

La reconnaissance des nœuds électriques utilise des **règles de mise en contact électrique** que le concepteur applique lorsqu'il édite les différentes cellules de sa conception. Un fichier technologique définit l'ensemble des couleurs et pseudo- couleurs des différents motifs du layout et leur règle de mise en contact. Une 2D-matrice - triangulaire- représente alors de manière interne l'ensemble de ces règles d'équivalence, loi de composition externe des couleurs dans l'ensemble {-1, 0, 1}; la valeur 1 caractérise l'équivalence, la valeur -1 l'interdiction de recouvrement. Ces règles d'équivalence génèrent une partition de l'ensemble des motifs du layout en classes d'équivalences ou nœuds électriques dans l'ensemble des motifs du layout. *(chap. 3.§4.2.3. & chap.5.§1.1.).*

Un simple algorithme résoud les intersections géométriques des divers motifs du layout et recherche l'éventuelle **équivalence électrique des motifs** qui se superposent ou se juxtaposent; lorsque deux motifs sont reconnus être électriquement équivalents, l'action **Union-Find** est invoquée, ce qui provoque la fusion des nœuds électriques, c'est-à-dire des arbres de **Fischer-Galler** associés. Les équivalences structurelles, fournies par la base de données des circuits, provoquent éventuellement de nouvelles fusions de nœuds électriques - UNION-FIND est encore invoqué. Les éléments en **erreur** sont signalés graphiquement lors de **l'édition** des cellules ainsi extraites. ***L'action Union-find*** *est donnée dans l'annexe 2 .§2.4.*

4.3. Adaptation des algorithmes à un layout structuré

Après le traitement des rectangles ou *éléments primaires* du layout, les équivalences des *pins* des sous-cellules sont réalisées aussi par l'action **Union-Find.** La réalisation du graphe d'équivalence obtenu dans l'ensemble des ports d'une cellule assure la propagation des équivalences dans ses divers contextes d'utilisation; ce graphe d'équivalence est maintenu dans la base de données des circuits.

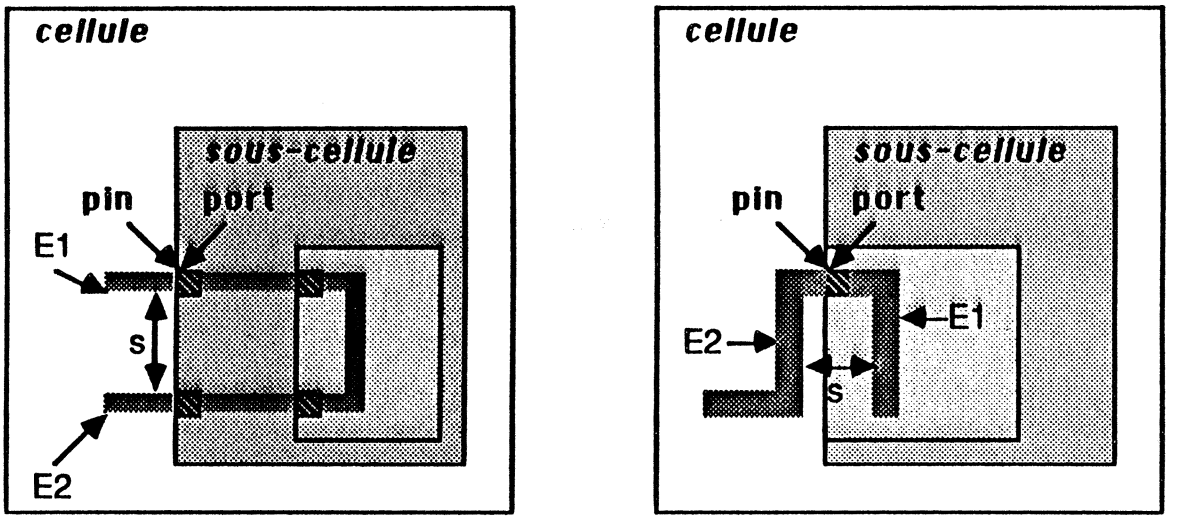


Figure 3 - ***Connexité, équivalences électriques et contraintes d'espacement.*** *- Le réseau compacte des nœuds électriques, obtenu par la propagation hiérarchique - et ascendante - des équivalences de ports des divers modèles utilisés permet ainsi la reconnaissance des équivalences électriques entre des rectangles , tels que E1 et E2, à travers la hérarchie du layout. Dans le cas (a) , l'équivalence électrique est reconnue immédiatement par l'action* ***Union-Find*** *puisque E1 et E2 appartiennent à la même cellule et que toutes les équivalences ont été déjà propagées; dans le cas (b) l'équivalence électrique est analysée par la* ***fonction représentant-nœud*** *à travers la hiérarchie du layout . (chapitre 5.§.2.5. & annexe2.§2.5.). La reconnaissance de l'équivalence électrique entre deux éléments primaires du layout , plus ou moins profondément enfoncés dans la hiérarchie de conception, évite la détection par le DRC de* ***fausses erreurs d'espacement s.***

Lorsque tous les nœuds électriques ont été ainsi reconnus, leurs arbres de Fischer-galler associés sont complètement compactés et un **résumé** du graphe électrique est alors réalisé. Le **réseau compacte** des nœuds électriques ainsi construit permet de reconnaître plus généralement que deux éléments portent le même signal, même quand ils sont profondément enfoncés dans différentes sous-cellules *(figure 3)*. La fonction **représentant-nœud**, invoquée pour chacun des éléments, assure cette reconnaissance; ceci est une **extension pour l'action FIND de la fonction Union-Find en hiérarchie.** *La fonction **repésentant-nœud** - présentée au chapitre 5 en §2.5. - est donnée dans l'annexe 2 en §2.5. **L'algorithme général de construction du réseau compacte de nœuds électriques** est donné dans l'annexe 2 en §2.1.*

4.4. Mise à plat du graphe électrique

Après la reconnaissance des nœuds électriques, les représentants des nœuds électriques sont mis dans des listes de nœuds externes et de nœuds internes; les transistors, éventuellement constitués de plusieurs rectangles, sont traités comme des nœuds électriques et leur représentant est placé dans la liste des transistors. Lorsque le réseau complet des nœuds d'un circuit, cellule après cellule, a ainsi été généré, **un algorithme récursif permet de le mettre complètement** à plat. Ainsi :

1. ***Les nœuds du niveau le plus haut sont préalablement instanciés;***
2. ***les sous-cellules sont ensuite traitées récursivement ;***

Pour chaque instance de cellule / modèle, ***tous les nœuds internes au modèle*** c'est-à-dire qui sont une partie de cette cellule mais ne sont pas une partie - en tant que nœud électrique - d'une cellule de plus haut niveau, ***sont instanciés*** *(nommés)*. On fait alors correspondre un <nom global> à une association (<nom local> / <contexte hiérarchique>) soit au moyen d'une table d'identificateurs, soit par une fonction biunivoque. ***L'algorithme de mise à plat du graphe électrique*** *est donné dans l'annexe 2 .§4.*

4.5. L'évaluation et l'optimisation des algorithmes

L'optimisation d'un extracteur conventionnel de la connectivité électrique par la méthode de recherche géométrique a montré l'intérêt présenté par l'utilisation de cette méthode. Les résultats obtenus, *présentés dans l'annexe 5,* ont permis de contrôler que des quadtrees à seuil de partition quaternaire élevé (environ 400 éléments) associé à un faible seuil de partition binaire (quelques éléments) optimisait assez bien mémoire interne et temps global de calcul de l'algorithme ; ces résultats ont ainsi confirmé la supériorité des **mixed-quadtrees** par rapport aux **purs quadtrees adaptatifs** [BER 85], [BER 86].

Cet extracteur conventionnel des nœuds électriques n'utilise pas la méthode *Union-find* et réalise la fusion de deux composantes connexes avec une recherche en N^2 des *rectangles à renommer (annexe 5.§2.)*.. Ceci explique les temps de traitement obtenus qui sont non proportionnels aux tailles des cellules traitées. Pour cet outil conventionnel, la méthode de recherche géométrique a considérablement amélioré la recherche géométrique des *contacts électriques.* La fusion des nœuds

électriques par l'algorithme **Union-find** permet une gestion beaucoup plus efficace des équivalences électriques puisque n opérations FIND sont réalisées en un temps bien meilleur que **O(nlogn)** *(chap. 3.§4.2.2.)*. La combinaison des deux méthodes de *recherche géométrique* et *d'équivalence* devrait ainsi assurer, dans l'analyse hiérarchique de la connectivité électrique, de bien meilleurs résultats. De plus l'analyse incrémentale des cellules éditées, réalisée par marquage du traitement effectué et report des principaux résultats dans la base de données, constitue un des aspects essentiels de l'optimisation de ces algorithmes d'analyse hiérarchique du layout. L'analyse incrémentale est une des conditions au traitement de gros circuits bien structurés.

5. VERIFICATION HIERARCHIQUE DES REGLES DE DESSIN

Il peut être nécessaire de respecter entre deux motifs du layout soit une **contrainte de connectivité**, soit une **contrainte d'espacement** *(figure 4)*. Une **hiérarchie des quadtrees** avec un quadtree associé à chaque modèle de cellule, permet la vérification d'un élément ou motif de base avec ses voisins, même profondément imbriqués dans la hiérarchie des cellules [BER 86].

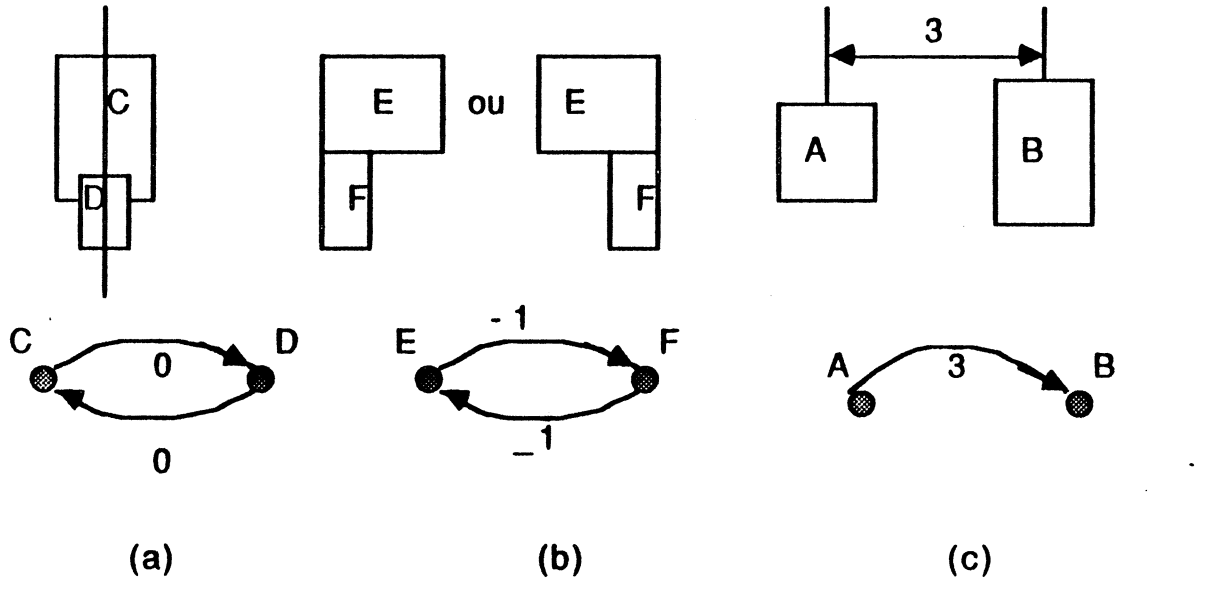


Figure 4 - ***Différents types de contraintes géométriques de conception*** *- Deux rectangles du layout peuvent avoir entre eux une* ***contrainte de connectivité*** *comme en (a) et (b) ou une* ***contrainte d'espacement*** *comme en (c). Dans la construction du graphe orienté des diverses contraintes des rectangles du layout, les contraintes d'espacement entre rectangles sont caractérisées par une contrainte de poids positif (a), et les contraintes de connectivité par une contrainte de poids négatif ou nul - espacement minimum ou maximum entre axes, positif ou négatif.*

5.1. Le D. R. C. - Principe

L'algorithme vérifie les **interactions** entre chaque paire d'*éléments primaires* dans une cellule ou dans ses sous-cellules, seulement s'ils sont si proches l'un de l'autre que certaines règles de conception puissent être violées. On dira dans ce cas que les éléments sont en interaction; cette interaction produira alors un contrôle de

l'algorithme. L'intérêt de la méthode de recherche géométrique proposée est sa capacité à sélectionner effectivement les éléments concernés, améliorant ainsi l'algorithme de base de filtrage hiérarchique donné dans [WHI 81] ; cet algorithme réalise donc une vérification hiérarchique effective - et non un filtrage - sur des hiérarchies n'ayant pas d'interdictions de recouvrement. Plusieurs occurrences de la même interaction relative de deux symboles ne sont vérifiées qu'une seule fois. Deux éléments à l'intérieur de la même sous-cellule sont supposés être déjà vérifiés sauf en cas de demande explicite du concepteur ; dans ce cas, certains éléments en erreur et appartenant à des nœuds externes d'une cellule vérifiée, peuvent être éventuellement **revérifiés** dans l'environnement qui les utilise. Ils sont alors considérés comme de fausses erreurs et instanciés, pour être résolus, dans le contexte d'utilisation *(§5.6.)*.

5.2. Analyse de connectivité et restriction des règles de connexion

Dans le but de réaliser des algorithmes efficaces de DRC, nous devons être capables d'**éviter de mauvais diagnostics** en **reconnaissant** lorsque deux éléments de base portent le **même signal**, même si ces éléments sont plus ou moins profondément enfoncés dans des sous-cellules voisines *(figure 3)*. Les conditions de connectivité électrique entre deux transistors ou entre un **transistor** et un **élément primaire d'un layout**, décrit avec un **symbolisme Stick** *(Annexe 4)*, génèrent un certain nombre de cas qu'il est aussi nécessaire de reconnaître à travers la hiérarchie de conception.

Quand on vérifie une cellule et que l'on considère les interactions de sous-cellules, deux *éléments primaires*, chacun dans une sous-cellule différente, peuvent être connectés si et seulement si ces éléments appartiennent à un nœud électrique du **niveau le plus haut** de la cellule. Quand on exécute l'algorithme de DRC et que l'on descend récursivement dans la profondeur de la hiérarchie de cellules jusqu'aux *éléments primaires*, notre propre méthode sera **d'empiler** pour chaque élément le **contexte d'appel** des instances de cellules ou **ancêtres** successivement utilisés dans l'instanciation de cet élément dans la hiérarchie; chaque ancêtre est ainsi caractérisé par l'identification de son modèle de cellule, sa **transformation relative** par rapport à la cellule qui l'utilise, sa **transformation absolue** (produit des transformations relatives de ses ancêtres) dans le référentiel associé à la **cellule de plus haut niveau** de la hiérarchie. La **pile des ancêtres** d'un élément constitue ainsi son **contexte hiérarchique** d'appel.

Grâce au réseau compacte de nœuds électriques précédemment extrait, nous pouvons reconnaître si deux éléments appartiennent à un même nœud électrique en calculant pour chacun d'eux son représentant électrique de plus haut niveau ou représentant électrique hiérarchique. **Le représentant électrique hiérarchique** d'un *élément primaire* est retourné par la fonction **représentant-nœud** (*cf. chap. 5.§2.5.)*. La fonction **représentant-nœud** a en entrée ***l'élément*** et son ***contexte hiérarchique;*** elle a en sortie *le* ***représentant électrique*** et son ***contexte hiérarchique.*** Deux éléments qui ont le même représentant électrique hiérarchique sont équivalents.

5.3. Le D.R.C. de base

Les routines de plus bas niveau du DRC vérifient les diverses contraintes entre les objets de base. Les **contraintes d'espacement** entre deux éléments de base (transistor, contact, connexion...) dépendent de leur **couleur**, de leur **position relative** et de leur éventuelle **relation électrique** [MCGR 80]. L'union des quatre contraintes d'espacement **bord à bord** de deux éléments rectangulaires définit un **rectangle d'espacement** *(figure 5)*. Pour un élément, la **région de recherche des interactions** est alors un rectangle fonction de la garde maximum correspondant à sa couleur (maximum d'une colonne de la matrice des gardes).

5.3.1. La forme générale des règles de dessin

L'ensemble des contraintes géométriques (régles de garde) des motifs du layout est fourni par un fichier technologique qui les définit pour un ensemble de couleurs (niveaux ou pseudo-niveaux) donné par le concepteur. L'ensemble des gardes d'espacement minimum doit être cohérent avec la métrique utilisée et assurer localement la transitivité des gardes d'espacement($r_{12} + r_{23} >= r_{13}$).

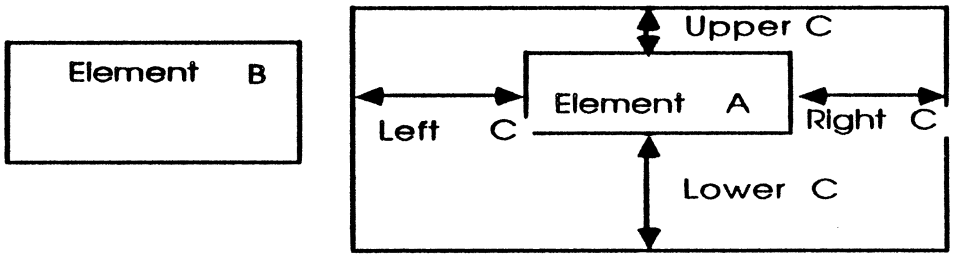


*Figure 5 - **DRC de base et non isométrie des gardes d'éléments multiniveaux.** Les routines de bas niveau vérifient deux **éléments primaires** (non structurés hiérarchiquement) . Leurs couleurs respectives et leur relation de connectivité électrique définissent les 4 contraintes d'espacement **bord à bord,** c'est-à-dire un **rectangle d'espacement.** La prise en compte d'éléments complexes et multi-niveaux tels que les transistors, entraîne en fait une **non isométrie des règles** et ainsi 4 valeurs différentes de garde d'espacement pour Left C, Right C, Upper C, Lower C. La réalisation d'une seule de ces 4 contraintes est nécessaire et le seul mouvement des objets suffit à changer la règle qui doivent satisfaire . La **région de recherche de possibles violations** - dite **région d'interaction** - est précalculée pour chaque couleur ; elle est déterminée par le maximun des gardes à respecter pour cette couleur (maximum de la colonne <couleur> dans la matrice des gardes). Remarque : Les règles d'espacement diagonal ne sont pas prises en compte.*

Une **contrainte** ou **garde d'espacement** entre deux éléments s'exprime selon la forme générale :

<couleur1> * <couleur2> * {<condition>} --> <valeur>;

elle exprime :

si condition (élément1, élément2) alors

distance (élément1, élément2) >= valeur (couleur1, couleur2);

Cependant la plupart des D.R.C. formalisent l'ensemble des contraintes entre les figures qui composent le layout, par des règles entre les arêtes qui composent les figures; ces règles prennent en compte toutes les règles de dessin. Ces règles sont généralement aussi plus faciles à vérifier.

Une **contrainte** ou **garde d'espacement** entre deux arêtes s'exprime selon la forme générale : **<couleur1> * <couleur2> * {<condition>} --> <valeur>;**

elle exprime :

si condition (arête$_1$, arête$_2$) alors

distance (arête$_1$, arête$_2$) >= valeur (couleur1, couleur2);

Pour l'élément A, *de la figure 5*, l'ensemble des 4 contraintes d'espacements **arête à arête** constitue un rectangle d'espacement interdit à B; le rectangle d'espacement de A dépend des *couleurs* des éléments A et B.

La réalisation d'une seule des quatre contraintes **bord à bord** suffit à assurer la non violation du **rectangle d'espacement** caractérisé par l'union des contraintes **bord à bord** pour tous les cas possibles de positions des éléments A et B. Sur un layout physique, les quatre contraintes d'espacement **bord à bord** sont égales généralement, ce qui n'est pas le cas en général pour un layout symbolique [SCHL 83]. Pour un layout symbolique, la prise en compte d'éléments complexes et multi-niveaux tels que les transistors, entraîne en fait une **non isométrie des règles** de ces transistors avec les autres éléments du layout *(figure 5);* la formulation de ces règles **bord à bord** distingue **l'axe** et **le bout** du transistor, ainsi que le côté éventuellement connecté (distance de coude ...). L'agrégat des règles technologiques du **layout physique** sont alors pris en compte par des règles qui correspondent au **savoir faire des concepteurs** dans le but d'obtenir un layout légal et assez bien optimisé [ROSET 82]; les rectangles d'espacement deviennent alors asymétriques, caractérisant la non-isométrie de ces règles. La distance réelle des éléments (A) et (B) est alors plus ambiguë, car elle n'est plus déduite d'une des normes usuelles du plan de conception (norme du max / norme euclidienne); en conséquence, afin de trouver un compromis entre la légalité et l'optimisation du layout, certaines **règles particulières** de positionnement entre les transistors et d'autres types d'objets du layout, doivent être prises en compte, soit par des **rectangles d'espacement non bornés sur trois de leurs côtés**, soit par des **rectangles d'espacement élargis aux bouts** des transistors *(figure 6).*

5.3.2. Classement et priorité des règles conditionnelles

Diverses **conditions paramètrent les règles de dessin** du layout ou règles de garde. Une **priorité** entre ces règles est **nécessaire** pour éviter les ambiguïtés. Divers compacteurs et DRC établissent ainsi une relation d'ordre entre les conditions; cette relation d'ordre entre les conditions évite les ambiguïtés et constitue de plus, différents niveaux d'élimination des **candidats à de possibles violations**. L'algorithme de vérification effectue ainsi une analyse de plus en plus détaillée des éléments en interaction, analyse pouvant conduire à la translation des symboles au niveau physique; dix niveaux d'élimination sont ainsi pris en compte dans [DRA 85].

Les conditions qui paramètrent des **règles de dessin** d'une description du layout selon un **symbolisme Stick,** sont des conditions d'orientation relative des éléments de base, leur éventuelle relation électrique, ainsi que certaines règles plus fines de positions relative des éléments. Le principe de la formalisation de ces règles est donné *dans l'annexe 4.* Afin d'éviter l'ambiguïté entre ces multiples règles, il est nécessaire d'établir une **priorité entre les conditions**. Les conditions qui paramètrent les règles de garde sont de type topologique ou électrique et le seul mouvement relatif des objets suffit en général à changer la garde qu'ils doivent satisfaire *(figure 5).*

Afin d'optimiser le DRC de base, les **conditions les plus simples** à vérifier ont **la plus grande priorité**, ce qui réalise ainsi un filtrage des éléments à **revérifier** en tenant compte des autres conditions. Les **conditions géométriques** sont ainsi plus prioritaires que les **conditions électriques**; cette priorité permet de ne considérer la **relation de connectivité électrique** parfois assez complexe entre deux objets *(Stick, Annexe 4)*, que lorsque ceux-ci sont déjà **géométriquement en erreur**. Cette priorité des conditions géométriques est aussi très intéressante pour optimiser les vérifications entre les objets de base d'un layout structuré, que la méthodologie soit symbolique ou non; en effet, la **vérification géométrique** d'un objet de base (A) avec les objets d'une sous-cellule (B) est réalisable directement dans le repère attaché à la sous-cellule (B), ce qui évite la transformation géométrique de tous les objets de la sous-cellule (B) en interaction avec l'objet de base (A).

La **priorité des vérifications pour les gardes d'espacement** *(méthodologie Stick - Annexe 4)* sera donc la suivante :

*1. **vérification géométrique** et prise en compte des seules **gardes géométriques** simples ;*
*2. vérification des objets **géométriquement en erreur** et prise en compte des gardes ayant une **condition de connectivité électrique**;*
*3. vérification des objets en erreur et prise en compte des **conditions topologiques et géométriques plus complexes** et liées à la condition de connectivité électrique de ces objets.*

La reconnaissance des **équivalences électriques** dans une même cellule non structurée, grâce à la simple gestion des arbres de Fischer-Galler et sur un layout structuré grâce à la fonction **représentant-nœud**, permet ainsi d'éviter les mauvais diagnostics dans la vérification des contraintes d'espacement d'éléments équipotentiels (fils et transistors coudés ..). Plus généralement, la représentation compacte du réseau des nœuds électriques préalablement extrait et maintenu pour la partie du layout vérifiée, permet l'étude complète de la **relation de connectivité électrique** entre deux objets quelconques imbriqués dans la hiérarchie du layout; cela permet la prise en compte des conditions du **type 2** et éventuellement de **type 3**.

Reconnaître qu'un élément du layout tel qu'un contact ou une connexion équipotentiel à un des accès du transistor est placé du même côté que l'accès auquel il est connecté, illustre la prise en compte des conditions du type 3 dans l'application générale des règles de gardes (DRC ou compacteur). Le filtre réalisé par la **priorité** des règles de **type 1 et 2** ne devrait générer que très peu de vérifications de **type 3.**

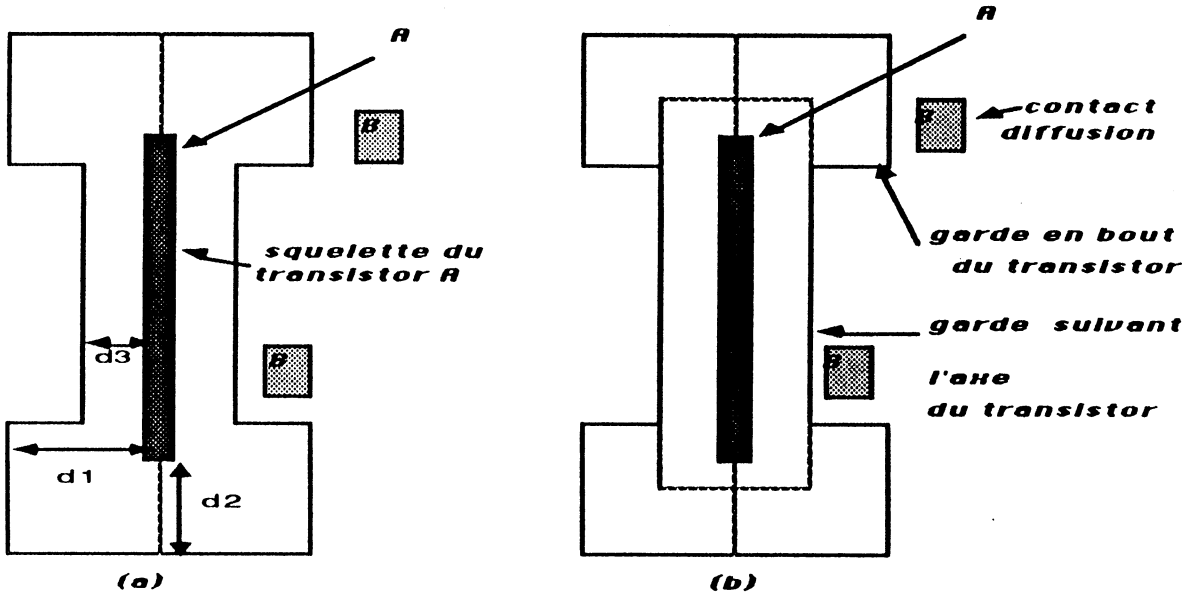


Figure 6 - (a) : contraintes d'espacement rencontrées dans certaines règles de garde du ***symbolisme Stick*** *[ROSET 82] avec un* ***transistor*** *- gardes d'espacement entre le squelette d'un transistor et un contact électriquement connecté à l'un de ses terminaux de diffusion. Leur vérification est faite selon le schéma (b) : vérification séparée de contraintes par rapport à l'axe et aux bouts de l'élément (A). Ce type de règle traduit la* ***difficulté d'exprimer des règles de garde simples (élément, élément)*** *lorsque l'on désire obtenir un layout assez optimisé.*

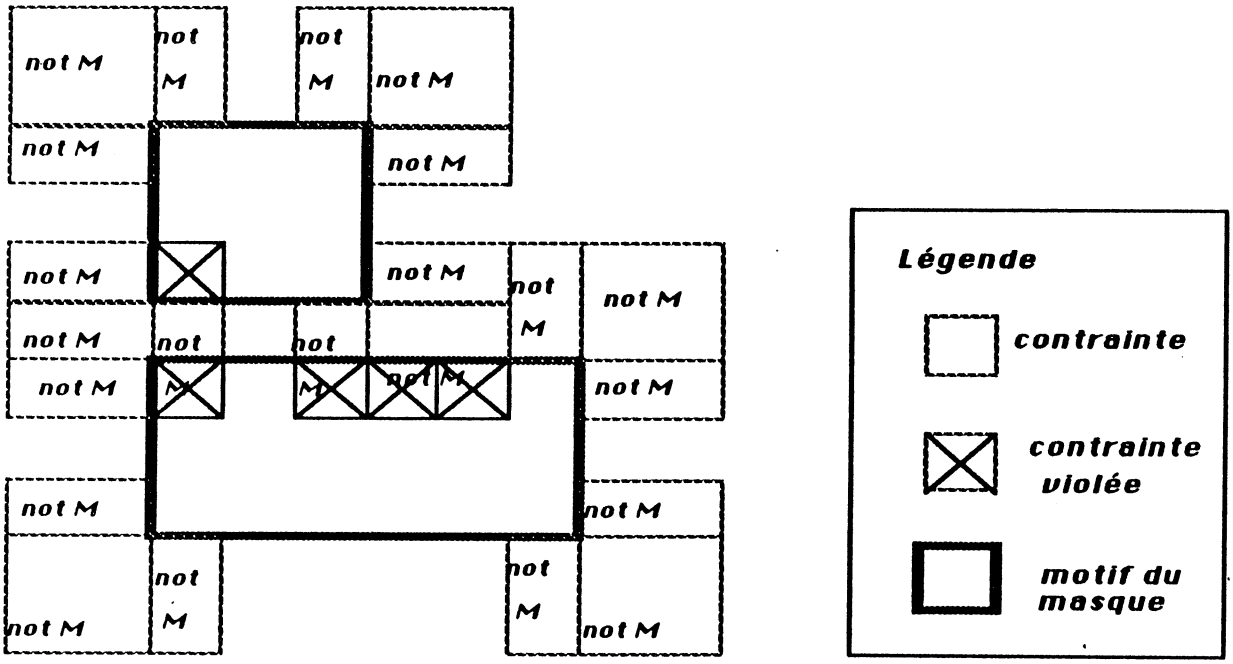


Figure 7 - LYRA - Les contraintes ne sont pas portées par les arêtes (edge based DRC) mais par les coins des divers motifs du layout (point based DRC). D'abord utilisé dans le système CAESAR [ARN 82], son principe fut repris par divers systèmes dont MAGIC [OUS 84 & HAM] & [TAY 84] et le système VIVID [ENT 85].

La **déformation du rectangle d'espacement** produit par l'union des gardes d'espacement gardes d'espacement entre le squelette d'un transistor et un contact électriquement connecté à l'un de ses terminaux de diffusion, caractérise aussi ces règles de **type 3** *(figure 6)*. Cette contrainte d'espacement est décomposable en la vérification de contraintes beaucoup plus simples associées à des rectangles d'espacement autour de l'axe et des bouts du transistor; cette décomposition nous rapproche alors de la méthode utilisée par le **vérificateur LYRA** (point or corner based DRC) [ARN 82]. LYRA vérifie en effet les **coins** des matériaux. Ce mode de vérification a été assez largement utilisé [TAY 84] [ENT 85]; il a été implémenté dans l'éditeur MAGIC [OUS 84 & HAM] pour vérifier des layout symboliques édités selon la technique **Corner-stitching** [OUS 84]. La vérification locale des motifs par leurs coins constitue aussi un ensemble suffisant de vérifications.

Dans des méthodologies de conception de layouts physiques, les contraintes de recouvrement et débordement, sont aussi à vérifier. Ces règles ont la même forme générale que les contraintes d'espacement. La même structure de données et le même algorithme de vérification des règles de dessin doit assurer leur vérification. Leur étude dépasse le cadre de cette implémentation, puisque le sous-système présenté est testé actuellement sur des descriptions symboliques, qui utilisent donc pour les transistors, des symboles corrects par construction, et par conséquent libres d'erreurs de débordement et recouvrement.

5.4. Le DRC hiérarchique

5.4.1. Principe de l'algorithme

Toute cellule éditée ou modifiée est vérifiée par l'algorithme de DRC hiérarchique selon les principes suivants :

1. Chaque sous-symbole instancié (élément construit) est vérifié récursivement. *Lorsque les modèles ont été vérifiés de manière incrémentale, une simple comparaison récursive des dates de vérification est réalisée*

2. Chaque élément simple ou structuré du niveau le plus élevé de cette cellule est vérifié par recherche géométrique *des éléments assez voisins pour causer des violations des règles de garde.* ***Les éléments ainsi sélectionnés sont dits être en interaction avec l'élément que l'on traite.***

L'algorithme **vérifier-symbol** réalise la vérification d'une cellule ou symbole selon la méthode décrite en *figure 8.* Un certain voisinage est ainsi recherché et examiné pour chaque élément. Toutes les interactions suffisamment proches l'une de l'autre pour provoquer d'éventuelles violations des règles de dessin sont ainsi examinées par l'action **vérifier-interaction**. Afin de vérifier l'interaction de deux éléments (A) et (B) dont l'un au moins (B) est structuré il est alors nécessaire de remplacer l'élément structuré (B) par les éléments qui le composent et de rechercher parmi eux les candidats à des violations avec (A). Grâce à cette méthode on élimine très vite les faux candidats et on finit par ne plus considérer que les *éléments primaires* suffisamment proches l'un de l'autre pour causer de réelles violations des régles de dessin. ***L'algorithme vérifier-symbol est donné dans l'annexe 2 en §2.6.***

L'action **vérifier-interaction** a en **entrée** les deux ***éléments primaires*** ou ***structurés*** et leur ***contexte hiérarchique***. Ce *contexte hiérarchique* est constitué de la *pile* des instances de symboles qui réalisent une instance de l'élément dans la hiérarchie de conception. Le *contexte hiérarchique* qui *instancie* un élément est aussi appelé **pile d'ancêtres** si on considère comme **ancêtres** de l'élément les différentes instances de cellules ainsi empilées. Lorsque l'on vérifie l'élément (A) avec chacun des éléments de (B) assez voisins, la *pile* du voisin de (A) dans (B) subit ainsi un empilement à l'appel de *vérifier-interaction* et un dépilement à son retour *lorsque ce voisin est un élément structuré.* ***L'algorithme vérifier-interaction est donné dans l'annexe 2 en §2.6.***

Plusieurs types d'interactions sont ainsi examinées :

1. L'interaction de deux *éléments primaires* provoque la vérification des règles de garde.

2. L'interaction de deux *éléments structurés* est vérifiée en recherchant dans l'une des structures référencées (B) les objets assez voisins de l'autre (A) par utilisation de son quadtree et le calcul de la position de (A) dans le repère attaché à (B), c'est-à-dire le repérage de (A) relativement à (B). Les objets du voisinage (A) dans (B) sont ensuite récursivement vérifiés avec (A).

3. L'interaction de deux éléments dont un seul est *structuré* (B) est vérifiée en recherchant dans (B) le voisinage de (A) comme en 2, et en le vérifiant

aussi récursivement avec (A) ; l'algorithme vérifie d'abord entre (A) et son voisinage dans (B) les *contraintes de type 1* , c'est-à-dire les contraintes de *nature géométrique*, réalisant ainsi un filtrage géométrique des interactions (évitant ainsi de nombreuses transformations géométriques). Les interactions des éléments **géométriquement en erreur** sont ensuite récursivement vérifiées.

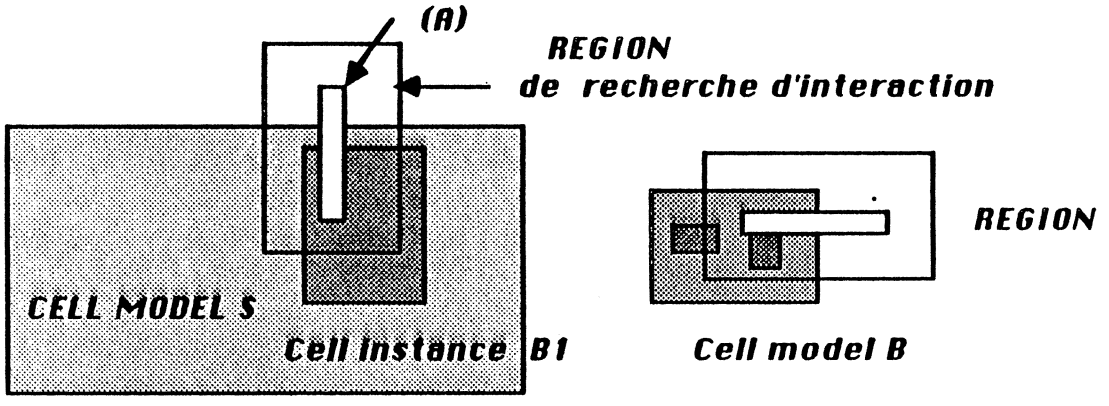


*Figure 8 - **Le DRC hiérarchique** - Chaque élément du plus haut niveau de la cellule vérifiée est vérifié avec chacun de ses **proches voisins** par l'action **vérifier-symbol** même si ceux-ci sont profondément cachés dans des cellules emboîtées. L'algorithme de ce D.R.C. hiérarchique est donné dans l'annexe 2.§2.6.*

1. *La recherche du voisinage d'un élément (A) dans une cellule (S) est réalisée dans (S) par une simple recherche géométrique*
2. *(A) est alors vérifié par l'action **vérifier-interaction** avec chaque élément (B) de ce voisinage, par calcul de la position de (A) dans le repère lié à (B) lorsque (B) est construit; par calcul de **la position** de (B) dans le repère lié à (A) lorsque (A) est construit et (B) est primaire; par vérification des gardes d'espacement lorsque (A) et (B) sont primaires.*
   *Le **contexte d'appel d'un** des éléments en interaction est empilé lors de chaque appel de l'action **vérifier-interaction** et dépilé à chaque retour de cette action; les références des symboles instanciés ainsi que les transformations absolues et relatives de ces symboles (par rapport aux cellules déjà empilées de plus haut niveau et de plus bas niveau) sont ainsi empilés et dépilés dans la **pile des ancêtres.***
   *La **position de (A) dans le repère lié à (B)** est calculée avec la mise à jour de la pile des ancêtres après vérification que le contexte d'au moins un des éléments courants en interaction a été modifié en changeant de voisin; (A) est alors instancié dans (B) (par exemple) par l'application de la transformation **T_aRb** = R (Tb) * Ta en notation fonctionnelle ; Ta : transformation qui caractérise la **position absolue de (A)**, R (Tb) : **inverse** de la transformation qui caractérise la **position absolue de (B).***
   *La transformation T_aRb permet d'instancier virtuellement (A) dans (B) , (A) étant réellement repéré dans la cellule qui l'instancie (père hiérarchique). Elle permet aussi de **reconnaître les occurrences d'une même interaction** et ainsi **d'éviter les redondances de vérification.***
   Le marquage de la vérification de l'interaction relative de (A) et (B) est réalisée soit par la sauvegarde de T_aRb soit par la mise à jour du quadtree de (B) par le rectangle enveloppe de (A) réellement instancié par T_aRb.

Les **éléments en erreur** sont marqués et instanciés par l'algorithme dans les cellules vérifiées et sélectionnables lors de la réédition de cette cellule. Les paires d'objets en erreur doivent pouvoir être reconnues de manière intéractive par une fonction réalisant la vérification interactive d'un objet sélectionné de la cellule éditée.

La **reconnaissance des diverses occurrences d'interactions relatives de symboles** construits et dont l'interaction est déjà vérifiée par l'algorithme, doit assurer la vérification rapide de cellules répétitives constituées de nombreuses occurrences des mêmes interactions.

Les recherches géométriques utilisées par l'algorithme ont été paramétrées de manière à prendre indirectement en compte le traitement réalisé. Ainsi, seuls les éléments ayant des couleurs susceptibles d'interagir avec l'élément vérifié sont sélectionnés dans l'extraction de son voisinage (par quadtree). Un pseudo-quadtree destiné à rechercher les seules interactions d'éléments construits devrait aussi améliorer la sélection. Mais le meilleur compromis est un problème ouvert que l'étude du comportement de l'algorithme sur des cas très divers doit résoudre.

La base de données supporte le marquage des cellules vérifiées, la gestion des équivalences électriques réalisées préalablement par l'extraction des nœuds électriques, en particulier les équivalences de ports que l'algorithme utilise à nouveau pour l'examen des relations de connectivité électrique.

5.4.2. La structure de données

Les algorithmes d'extraction de graphe électrique et de DRC sont en fait paramétrés dans un même algorithme hiérarchique. Une même structure de données est utilisée pour les deux traitements. ***L'algorithme** complet **de vérification** est donné dans l'annexe 2 en §2.6.*

Les filtres hiérarchiques, DRC et compacteurs utilisent en général une **hiérarchie canonique** [WHI 80, 81], [ENT 85], dans laquelle les sous-cellules sont des symboles instanciés uniquement par des translations. L'exclusion de transformations telles que rotations et réflexions est nécessitée par l'utilisation de listes triées dans les symboles utilisés. La transformation de la hiérarchie des cellules en une **hiérarchie canonique** accroît considérablement le nombre de symboles; la reconnaissance des **occurrences d'interaction de symboles** est aussi plus complexe et accroît le nombre de vérifications effectivement réalisées.

La structure de données utilisée par l'algorithme général paramétré par le DRC/ ERC et l'extraction des nœuds électriques a déjà été présentée. Cette structure de données prend en compte une hiérarchie non canonique de symboles / cellules. Une sous-cellule ou instance de symbole est pour cette structure de données **un symbole** et une **transformation** repérée par rapport au repère attaché à la cellule qui l'appelle.

Les **interaction de symboles** sont marquées par l'algorithme de DRC lors de leur vérification afin de ne vérifier qu'une seule fois une même occurence

d'interactions. Une **même occurrence d'interactions** est caractérisée par deux instances de symboles dans la **même position relative**. Le marquage est réalisé pour deux symboles A et B par le repérage de B relativement au repère attaché à A et le repérage de A relativement au repère attaché à B, c'est-à-dire le calcul des transformations $tA * tB^{-1}$ et $tB * tA^{-1}$, tA et tB étant les repérages de A et B dans le repère absolu.

Les transformations et changements de repère sont calculés en coordonnées homogènes [NEW 78 & SPRO], [PAV 82]. Un ensemble de matrices 3D de la forme

$$\begin{vmatrix} A & B & C \\ D & E & F \\ 0 & 0 & 1 \end{vmatrix}$$

permettant le calcul des transformations sur les coordonnées réelles selon la notation fonctionnelle :

$$\begin{vmatrix} A & B & C \\ D & E & F \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} X \\ Y \\ 1 \end{vmatrix} = \begin{vmatrix} X' \\ Y' \\ 1 \end{vmatrix}$$

toutes les variables étant réelles (6 paramètres effectifs). Cette méthode permet de réaliser la composition de transformations par multiplication de leurs matrices associées et l'inversion d'une transformation par inversion de la matrice associée (groupe multiplicatif non commutatif).

5.5. Optimisation de l'algorithme

5.5.1. Optimisation générale du D.R.C. hiérarchique

Les *recherches géométriques* d'interactions et voisinages sont optimales lorsque les paramètres de partition quaternaire et binaire des quadtrees utilisés sont bien ajustés aux régions de sélection. Pour l'optimisation d'un DRC conventionnel vérifiant un layout symbolique non hiérarchique, le meilleur ajustement correspondait à l'égalité des surfaces de partition et de sélection [BER 85]. Les partitions binaires utilisent moins de mémoire que les partitions quaternaires; ainsi les quadtrees à large seuil de partition quaternaire (800 à 200 rectangles) et faible seuil de partition binaire (environ 10 rectangles), réalisent pour l'algorithme général de DRC et d'extraction de nœuds électriques, une bonne optimisation générale des recherches topologiques et de la place que les quadtrees utilisés occupent *(annexe 4).*

Lorsqu'une **interaction** de symboles a été vérifiée, elle est marquée par **repérage relatif** de l'un des des symboles dans le **repère attaché à l'autre**. La composition des transformations nécessaires à ce repérage est calculée pour chaque interaction de symboles; elle est ensuite utilisée pour les *recherches géométriques* de voisinages de l'un des symboles dans l'autre.

Le marquage de la vérification de l'interaction de deux symboles est aussi réalisable par insertion dans le quadtree associé à l'un des symboles du **rectangle enveloppe** de l'autre. La phase d'optimisation des algorithmes doit permettre de choisir la meiileure solution

5.5.2. Optimisation du DRC de base

Le choix d'une bonne technique pour le DRC de base qui vérifie l'interaction de deux éléments non structurés, contribue à améliorer la vérification générale de layouts structurés *(chapitre 2).*

La détection des violations de chevauchement a été réalisée lors de l'extraction des nœuds électriques; la *recherche géométrique* des éléments en interaction génère **parfois** de **larges recherches géométriques**.

Les motifs du layout peuvent être considérés comme constitués par un ensemble d'arêtes orientées; la formalisation et la vérification des règles de dessin à partir des arêtes des motifs du layout est l'approche la plus répandue. Une autre approche consiste à faire porter a l'ensemble des coins des motifs les contraintes du dessin. Ainsi le vérificateur LYRA [ARN 82] contrôle les **coins des motifs** et non leurs **arêtes**. L'utilisation d'une telle approche pour la formalisation et la vérification des règles d'un **D.R.C. symbolique** de type **Stick** *(annexe 4)* est un problème ouvert. On constate en effet à l'implémentation qu'il est très difficile de vérifier le bon espacement de deux **motifs rectangulaires** du layout *(figure 5),* et que la vérification séparée des espacements de leurs **arêtes** est au moins 4 fois plus simple; de plus certaines règles de dessin sont assez inextricables même dans une optique de vérification des arêtes *(figure 6).* La formalisation des règles de dessin d'un **symbolisme Stick** serait-elle ainsi améliorée? De récents D.R.C. et compacteurs utilisent une telle approche pour la formalisation et la vérification de layouts symboliques [OUS 84 & HAM],[TAY 84], et pour l'adaptation de layouts flexibles [ENT 85].

La bonne formalisation des règles de dessin dans les fichiers technologiques, la détection préalable des incohérences de ces règles sur la matrice interne associée, l'utilisation d'une métrique usuelle du plan de conception, le paramètrage des conditions qui activent ces règles et la création d'une priorité entre ces conditions, doivent permettre la réalisation d'un DRC de base performant et une élimination rapide de la plupart des éléments candidats à de possibles violations. La transitivité des régles d'espacement évite certaines redondances de vérification. Le **principe de l'ombrage** utilisé par les compacteurs *(figure 9)* est une application de ce principe.

L'optimisation générale du DRC hiérarchique dépasse la simple application d'un ensemble de règles au moyen d'un algorithme intelligent. L'utilisation actuelle des **méthodes de compactage** influence les modes d'édition du layout [WES 82], [ENT 85], [ROG 85], [ROS 83 & WES]. Ces méthodes ont pour clef le **positionnement relatif** des différents dispositifs électriques **prévérifiés** et l'**adaptation aux règles** de dessin de ces **layouts topologiques et flexibles** grâce à des **méthodes de recherche géométrique** [ROS 85], [OUS 84] ; les compacteurs et vérificateurs hiérarchiques sont ainsi de plus en plus utilisés [ENT 85], [SCO 84], [TAY 84]. La description de la géométrie d'une conception devient ainsi plus topologique que métrique et l'indépendance par rapport à la technologie augmente. En contrepartie, ce paramètrage de la technologie exige une formalisation rigoureuse des règles géométriques et topologiques (connectique) de conception pour garantir la correction du layout généré [BAL 82], [GRE 86].

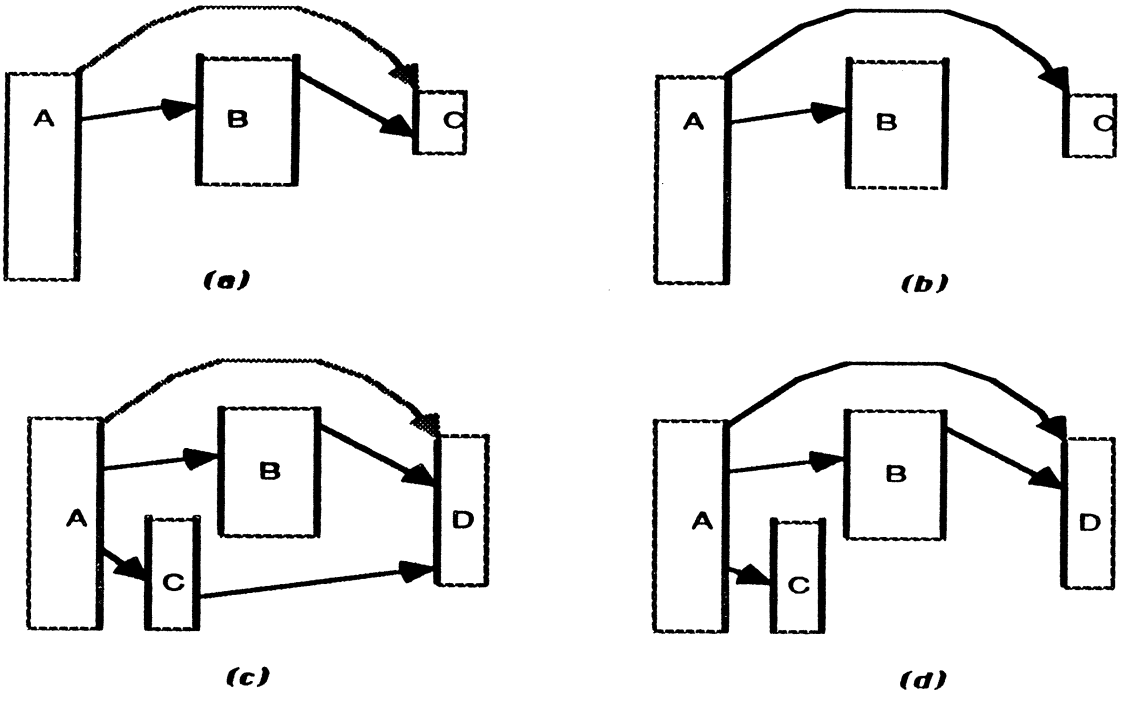


Figure 9 - **Technique de l'ombrage (shadowing) d'un élément** -

Soient A , B , C et D des éléments appartenant à différents niveaux.

10.a - Bien qu'une contrainte d'espacement existe entre A et C , cette contrainte est redondante puisque B couvre C et qu'il existe des contraintes d'espacement entre A et B ainsi qu'entre B et C .

10.b - la contrainte d'espacement entre A et C n'est pas redondante bien que B couvre C, puisqu'il n'existe pas de contrainte entre B et C.

10.c - la contrainte d'espacement entre A et D est redondante, puisqu'il n'existe pas de contrainte entre C et D.

10.d - la contrainte d'espacement entre A et D n'est pas redondante.

Formulation des règles de conception *- La technique de l'ombrage est utilisée pour réduire la complexité de la génération du graphe des contraintes des 1D-compacteurs [BAL 82], [HSU 80]. En (b) et (c) l'utilisation de la technique de l'ombrage srait erronnée, puisque la transitivité des contraintes n'est pas assurée à cause de l'**absence de contrainte** avec les rectangles qui réalisent un **ombrage** (absence de visibilité). Les règles de conception sont alors formulées de manière à **utiliser l'ombrage sans restriction** , comme en (a) et (c) [SCHL 83] , [SCHI 85], [WOL 84].*

Optimisation du DRC *- Pour la technologie CMOS les contraintes d'espacement entre les niveaux intérieurs aux caissons et les niveaux extérieurs sont ainsi redondantes (ombrage du caisson); les contraintes d'espacement formulées par rapport aux arêtes du caisson pour ces niveaux sont suffisantes; les recherche géométriques d'interaction du DRC sont ainsi plus locales et plus performantes.*

5.6. Fausses erreurs produites par la vérification incrémentale

Comme cela a été observé *au chapitre 2 (en §5.4.) l'analyse hiérarchique et incrémentale* présente l'avantage de détecter précocement la plupart des erreurs. Cependant elle a aussi pour inconvénients, la détection possible de **fausses erreurs**, qu'il nécessaire de distinguer des vraies erreurs, en attendant une analyse plus complète de la hiérarchie de conception. Ainsi une cellule peut ne pas être correcte par elle-même mais être correcte dans le contexte qui l'utilise *(figure 10).* On propose une solution assez simple et incrémentale pour le traitement et la résolution des *fausses erreurs.* Une fois reconnues comme telles, ces *fausses erreurs* peuvent être instanciées dans les contextes qui utilisent ce modèle de cellule, exactement comme sont instanciés ses éléments terminaux (ports). Ces éléments en erreur sont alors rajoutés à l'ensemble des ports, éventuellement distingués, et le graphe d'équivalence du *résumé de l'analyse de la cellule* est aussi complété; l'instance de ce *résumé de la cellule* et la réalisation de son graphe d'équivalence dans le contexte utilisateur doit alors permettre la résolution de ces *fausses erreurs (chapitre 2.§5.5).*

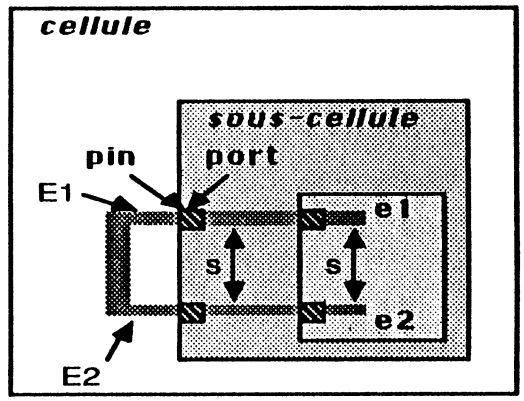


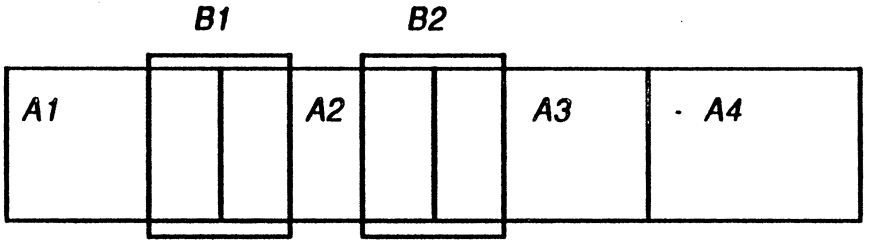
*Figure 10 - **Les fausses erreurs** - Une cellule peut ne pas être correcte par elle-même mais être correcte dans le contexte qui l'utilise. Ainsi **e1** et **e2** ont entre eux une erreur d'espacement **s**, levée par l'analyse de la hiérachie, deux niveaux plus haut (E1 et E2 ne sont pas en erreur). Ainsi l'analyse incrémentale est une cause du signalement de **fausses erreurs**. Une fois reconnues ces fausses erreurs sont instanciées de manière spécifique dans les contextes qui utilisent ce modèle de cellule, de manière récursive et ascendante, jusqu'à leur résolution.*

Il est à remarquer que, dans une cellule, seuls les *éléments primaires* qui appartiennent à des *nœuds externes* peuvent produire de *fausses erreurs.* En effet les *élément primaires* qui appartiennent à des *nœuds internes* ne peuvent produire que des *erreurs locales* non résolues par une utilisation particulière de la cellule.

Il est à remarquer que la plupart des erreurs d'espacement habituellement générées par les D.R.C. hiérarchiques à cause d'une analyse incomplète de la connectivité électrique interne de la cellule analysée ont été évitées par la propagation hiérarchique des équivalences électriques. Ainsi les fausses erreurs générées par l'analyse incrémentale résultent de la méconnaissance de la connectivité externe de la cellule analysée.

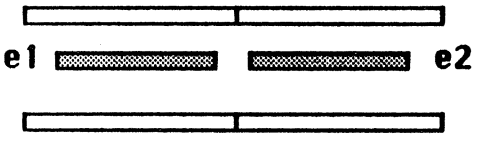
5.7. Erreurs non détectées par le D.R.C. hiérarchique

La vérification de l'interaction de sous-cellules est réalisée directement dans le contexte d'utilisation. Les interactions de modèles de cellules déjà vérifiées sont marquées par l'algorithme de D.R.C. afin d'optimiser la vérification de layouts réguliers. Cependant elles peuvent provoquer la non détection de certaines erreurs par le D.R.C. hiérarchique *(figure 11).*



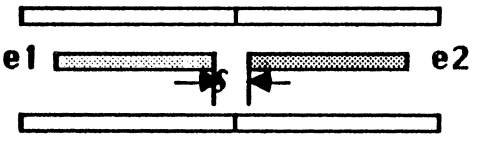
(a) cellule originelle

(b) éléments primaires de A **éléments primaires de B**



(A1, A2)

(c) vérification de l'interaction (A1,A2) qui est légale



(A3, A4)

(d) non vérification de l'interaction (A1,A2) qui n'est pas légale et non détection de l'erreur s

*Figure 11- **La non détection de certaines erreurs** et la nécessité d'un critère plus complet pour la reconnaissance des interactions de sous-cellules déjà vérifiées. Hypothèses : (A1, A2, A3, A4) instances de A; (B1, B2) instances de B; les cellules A et B contiennent la géométrie montrée en (b) et la juxtaposition de A3 et A4 crée , en **(d), une violation d'espacement s, non reconnue par le DRC hiérarchique** , si on considère leur interaction comme déjà vérifiée, en **(c)** . En effet les sous-cellules A3 et A4 ont un graphe de connexion différent de celui des sous-cellules A1 et A2 (pour lequel **e1 et e2** sont électriquement équivalents).*

Pour éviter la détection de fausses erreurs , il est alors nécessaire - dans hiérarchie qui autorise les recouvrements de sous-cellules - d'avoir un critère de reconnaissance des interactions déjà vérifiées, qui prenne en compte la manière dont les sous-cellules sont interconnectées , lorsqu'elles ont une même position relative.

Lorsque le D.R.C. hiérarchique vérifie l'interaction des éléments primaires enfoncés dans deux sous-cellules, il réalise, si nécessaire, une analyse de connectivité électrique. Or la reconnaissance des diverses occurrences d'une interaction de modèles déjà vérifiés, est réalisée de manière purement géométrique *(cf. §5.4.2.)*. Il suffit alors de modifier la manière dont ces cellules sont connectées pour provoquer une utilisation différente de l'interaction de ces 2 modèles avec cependant une même position relative. La position relative des 2 modèles de cellules constitue alors un critère insuffisant pour assurer la non revérification de cette nouvelle interaction.

Une solution simple aux diverses utilisations de modèles de cellules, dans une même position relative, est leur revérification systématique. Si cette méthode peut être préférable à une mise à plat et vérification conventionnelle, elle est cependant coûteuse en temps.

Un critère plus complet permettrait cependant de reconnaître qu'il est inutile de revérifier une interaction de sous-cellules. Ainsi l'interaction de deux sous-cellules sera considérée comme déjà vérifiée, si les deux modèles de cellules ont déjà été vérifiées dans la même position relative et avec les mêmes interconnexions. Le graphe d'équivalence de leurs points de connexions - pins - dans le contexte d'utilisation est donc aussi à comparer. Cette comparaison n'est effectuée que si les modèles de cellules ont la même position relative. Dans le cas où l'utilisation de ces cellules est différente, soit de manière géométrique, soit de manière connectique, leur interaction est alors à vérifier.

6. L'APPLICATION AUX FONCTIONS D'EDITION

Lors du processus d'édition du layout, le concepteur utilise des règles de superposition et de juxtaposition des motifs correspondant aux règles de réécriture et d'équivalence des fichiers technologiques. Ces lois de composition associées à la connectivité géométrique des motifs, suffisent presque à décrire les différents dispositifs électriques que le layout représente, et allègent la tâche de spécification.

La connectivité des motifs est contrôlée par l'algorithme d'extraction des nœuds électriques. Cette vérification étant faite, l'espacement des motifs n'ayant pas entre eux une contrainte de connectivité, est contrôlé par l'algorithme de DRC hiérarchique.

Post-processeurs ou interactifs, les algorithmes d'extraction et de vérification du layout d'une conception en fournissent un contrôle quasi continu. La mise à jour dans la base de données des informations extraites et des vérifications en permet une utilisation **"on line"** et **très locale**. Ainsi de petites parties du layout, cellule, fenêtre de travail, simple motif, dispositif électrique ..., peuvent être analysées de manière interactive et **dynamique**.

Interactifs ou post-processeurs, ces algorithmes permettent l'utilisation de **fonctions globales de synthèse** et **de contrôle** de la connectivité électrique, ainsi que la vérification du bon **dimensionnement** des divers dispositifs électriques réalisés au travers de la hiérarchie de conception.

6.1. Visualisation d'un noeud électrique

Le concepteur doit pouvoir visualiser un nœud électrique par la sélection d'un élément quelconque qui la compose. L'action **h-spanner** invoquée réalise à partir d'un **rectangle de départ**, la sélection, par recherche géométrique récursive, d'une partie connexe d'un nœud électrique, dans la profondeur de la hiérarchie du layout édité *(chap. 5.§2.4.)*.

La sélection du nœud électrique par son nom correspond à la recherche de tous les éléments qui ont le même représentant électrique. L'action **h-select-nom** assure la sélection complète d'un nœud électrique, à travers la hiérarchie de conception, **à partir d'un rectangle de départ** *(chap. 5.§2.4.)*.

6.2. Contrôle des interactions d'un objet du layout édité

Une fonction d'édition assure la vérification de l'interaction d'un *élément primaire* par rapport à son contexte. Cette fonction est utilisée pour placer correctement un nouvel élément - par exemple une connexion devant traverser une succession de sous-cellules - ou pour connaître les erreurs causée par un élément signalé en erreur après le passage d'un D.R.C. post-processeur. La position de l'élément vérifié est calculée par l'algorithme de DRC par rapport aux sous-cellules en interaction avec lui, et de nouvelles interactions sont découvertes dans ces sous-cellules... Les éléments en erreur sont affichés à l'écran dès que la fonction de vérification est terminée.

6.3. Fenêtre de travail et contrôle des interactions

Une fenêtre de travail peut aussi être indiquée par le concepteur afin d'effectuer la vérification en profondeur de tous les éléments appartenant à cette fenêtre de travail et les éléments en erreur sont affichables à l'écran à la fin de la vérification. Le concepteur ne vérifie ainsi qu'une toute petite partie de la conception *(annexe 2)*.

6.4. La fonction Shell, contrôle global de la connectivité électrique

Certains systèmes [STEV 84], [MCCA 84], [WON 85], [WAG 84] créent, lorsqu'ils extraient les nœuds et paramètres électriques d'un symbole, un nouveau symbole constituant une vue externe du symbole extrait. Ces symboles duaux - hallo ou shell ou abstract [SCH 86] - du symbole extrait, permettent la réalisation d'analyses incrémentales du layout de cellules - extraction et D.R.C.- ainsi qu'un contrôle général de la connectivité électrique par simple visualisation de ces **cellules duales**. Ainsi, dans [MCCA 84], le SHELL de chaque cellule est constitué de toute la géométrie du layout de ses nœuds externes. Dans notre système d'édition, une fonction **Shell** construit par un algorithme (descendant) une vue interactive de la **partie externe** d'une cellule. L'algorithme utilise le *réseau compacte des nœuds électriques* associé aux sous-cellules extraites pour instancier complètement chaque *nœud électrique externe (chapitre 5 - 2.3.)*. Cet algorithme est dual de l'algorithme de mise à plat du graphe électrique *en §4.4. **L'algorithme Shell** est donné dans l'annexe 2 en §3.*

7. L'ADAPTATION DU LAYOUT A UN ENSEMBLE DE REGLES

Les circuits **dessinés à la main** produisent un layout assez compact grâce au **savoir faire** des concepteurs, difficile à formaliser, mais la correction des erreurs est fastidieuse et alourdit considérablement le travail de conception. Les layouts ainsi édités dépendent de la technologie utilisée et doivent donc être réédités lorsque celle-ci change. Ainsi, depuis quelques années, les **programmes de compactage** et **d'adaptation** des layouts édités **à une technologie** sont-ils de plus en plus utilisés. Ces techniques s'affinent et modifient profondément le processus d'édition du layout; l'utilisation de **grilles virtuelles** ou de **plans "corner stitched"**, permettent le positionnement relatif, de symboles ou motifs indépendemment de la technologie utilisée.

Que les compacteurs opèrent sur des layouts symboliques ou physiques, ils ne changent jamais la topologie globale du circuit, caractérisée par **l'ordre relatif de ses composants**. Ils utilisent les **contraintes de connectivité** et **d'espacement** fournies par l'utilisateur et la technologie pour minimiser et adapter l'espace occupé par le layout; l'optmisation de **l'espace vide** et des interconnexions entre composants sont diversement pris en compte par les fonctions objectifs de ces compacteurs. Ces applications en général contractantes, font subir de nombreuses déformations aux interconnexions (jogs).

Les **contraintes d'espacement** sont paramétrées par des conditions géométriques, topologiques ou électriques et le seul mouvement relatif des objets suffit en général à changer la contrainte qu'ils doivent satisfaire (quatre contraintes possibles bord à bord..) ce qui oblige les **2D-compacteurs** tels que [SHI 86] à gérer dynamiquement le graphe des contraintes.

7.1. Recherche géométrique et génération du graphe des contraintes

L'étape la plus coûteuse des algorithmes de compaction est la construction du graphe des contraintes [LIA 83], [CHO 85]. Cependant, ce domaine a été un des moins explorés de la compaction [HED 85]. Les compacteurs utilisent des techniques **d'ombrage**, c'est-à-dire de visibilité des objets intervenant dans la construction du graphe des contraintes afin de le simplifier [WOL 84] *(figure 9)*.

Les 1D-compacteurs minimisent séparément chacune des dimensions du layout; le **groupage** des objets fortement connectés dans le sens perpendiculaire à la compaction, réduit le nombre de **groupes intervenants** et donc le nombre de comparaisons possibles; ce groupage supprime les cycles du graphe caractéristiques des contraintes de connectivité, et assure ainsi une résolution plus simple du problème [HSU 79].

Les méthodes de recherche géométrique sont des outils très utiles pour la réduction générale de la redondance du graphe des contraintes [ROS 85], [ROG 85]. Elles permettent l'identification dynamique de la connectivité et évitent la création de listes d'adjacence dont la complexité est en E^2 pour E objets; elles permettent ainsi la détermination des **groupes** intervenants. Les méthodes de recherche

géométrique accélèrent ainsi la phase de construction du graphe des contraintes, avec des complexités très inférieures à $O(E^2)$. Les méthodes de recherche géométrique permettent aussi une gestion dynamique du graphe des contraintes pour les 2D-compacteurs [SHI 86].

Les **compacteurs hiérarchiques** doivent profiter de la hiérarchie des quadtrees pour des recherches d'interactions hiérarchiques afin d'assurer la transparence des sous-cellules, comme pour le DRC hiérarchique.

7.2. Recherche géométrique, graphe d'équivalence et gestion de l'espace vide

La réduction de **l'espace vide**, but commun à tous les compacteurs, est résolue par la gestion explicite par la structure de données, de **l'espace vide** laissé entre les **éléments du layout** [OUS 84], [NAN 86], et par l'utilisation d'algorithmes qui synthétisent cet espace.

La méthode des **lignes de partage** identifie les **chaînes d'espace vide** qui apparaissent dans un layout physique [DUN 78], [NAN 86], ou symbolique [WES 81] *(figure 13).*

Les 2D-compacteurs synthétisent **les îlots formés entre les chaînes d'éléments**. Selon le principe d'invariance de la topologie du layout, **un élément ne peut être contraint que par les éléments bornant les îlots qui lui sont adjacents** [KED 83], [WOL 84], [WAN 84] *(figure 14).*

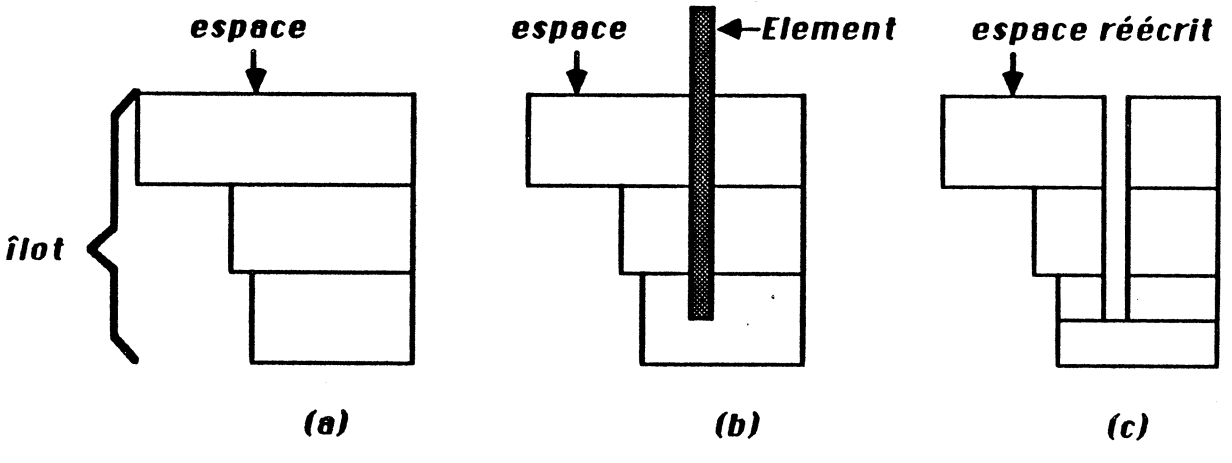


*Figure 12 - Lorsqu'un élément est inséré dans une cellule, la réécriture incrémentale des **pavés d'espacement** permet de connaître l'espace laissé vide; le processus de réécriture commence alors avec **un seul pavé** de couleur blanche (espace) dans le cas où la cellule est rectangulaire. Les **pavés d'espacement** sont maintenus dans le quadtree de la cellule et les différents **îlots**, formés entre les chaînes d'éléments, sont reconnus par l'algorithme **union-find** lors de l'extraction des noeuds de la géométrie des masques. Les suppressions d'éléments sont simples puisqu'il suffit de remplacer l'élément par un rectangle d'espacement et d'unifier - avec union-find - les **îlots** adjacents obtenus par recherche géométrique. L'évaluation de cette méthode de synthèse de l'espace pour des cas spécifiques - 2D-compacteurs - est un **problème ouvert**.*

L'application des règles de conception et la maintenance des relations d'adjacence sont habituellement utilisées par ces algorithmes de synthèse de l'espace. La **maintenance explicite** dans la structure de données **de l'espace laissé par les composants du layout** est réalisée de façon incrémentale en recouvrant cet espace par un ensemble de **pavés d'espacement**; l'insertion d'un élément provoque une *recherche géométrique* et une réécriture des **pavés** qu'il couvre, dans le quadtree associé à la cellule éditée *(figure 12 & chap. 4 en §2.4.)*.

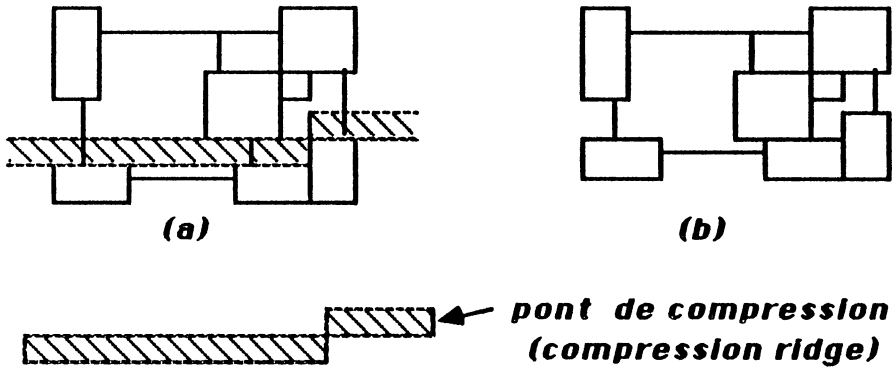


Figure 13 - ***Méthode des lignes de partage*** - *Une synthèse de l'espace vide est réalisée par la méthode des lignes de partage.*
(a) : pont de compression prévu; (b) : résultat de la compression ; [DUN 78, DUN 80, WES 81].

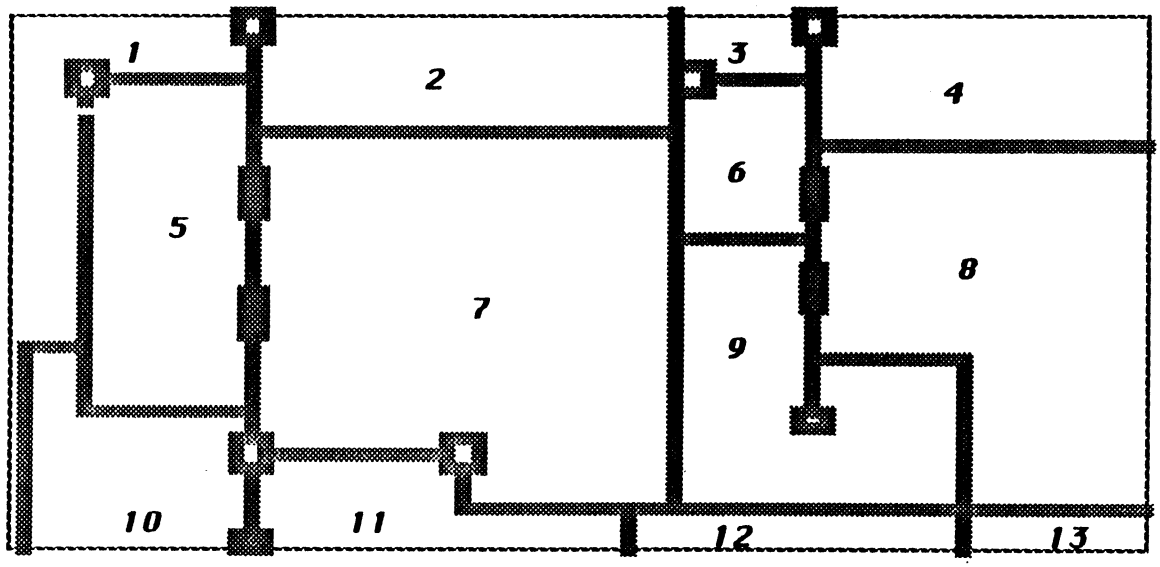


Figure 14 - ***Méthode des îlots*** - *Un élément ne peut être contraint que par les éléments des* ***îlots*** *qui lui sont adjacents (les différents* ***îlots*** *sont numérotés), des plans différents sont utilisés pour le métal et pour les niveaux qui interagissent entre eux , les éléments multiniveaux peuvent appartenir à différents plans [KED 82], [WAN 84]. La synthèse de l'espace vide nécessite alors la gestion de cet espace dans la structure de données. La reconstitution des* ***îlots*** *nécessite les même méthodes que l'extraction des noeuds de la géométrie du layout.*

Les **îlots** formés entre les motifs du layout sont alors traités comme de simples nœuds électriques. La partition complète du layout en **dispositifs électriques** et **îlots** est ainsi réalisée lors de la construction des nœuds électriques et avec la technique **Union-find**; un **arbre de Fischer-Galler** est ainsi associé à chaque **îlot**. Cette technique assure alors une synthèse de l'espace très efficace, selon la complexité de la technique **Union-find** *(chap. 3 en §4.2.2.)*. La sélection d'un *îlot* est alors réalisée de la même manière que la sélection d'un nœud électrique et par le même algorithme, à partir d'un rectangle de départ *(chap. 5.§2)*. Après la phase de reconnaissance des *îlots*, la recherche géométrique assure pour un **élément du layout**, la reconnaissance des **îlots** qui lui sont **adjacents**; cet élément sera **contraint** par les éléments qui bordent ces *îlots*.

7.3. Edition et adaptation interactive du layout

De récents algorithmes utilisent des méthodes dynamiques d'**adaptation de layouts flexibles.** Le **repoussage** récursif des éléments du layout constitue un des aspects de ces techniques [SCO 84]

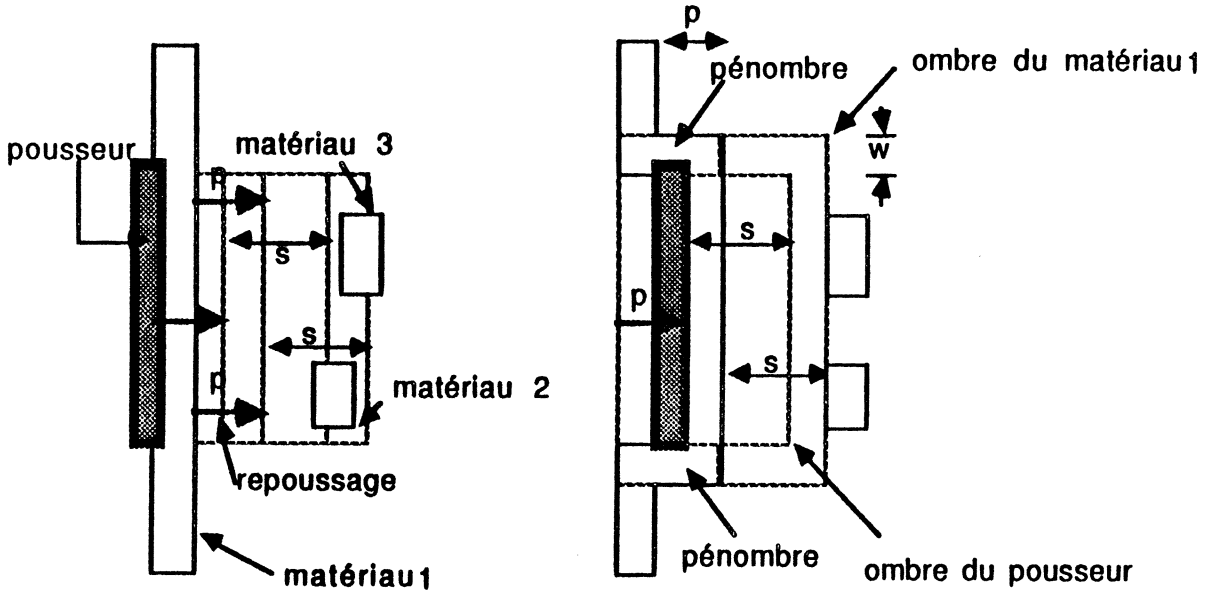


Figure 15 - ***Technique du repoussage*** *- Une arête appelée* ***pousseur*** *subit la translation* ***p*** *indiquée par le vecteur* ***repoussage*** *; quand le pousseur se déplace tous les matériaux dans l'aire qu'il balaie doivent être déplacés; de plus les différents matériaux ont entre eux une* ***contrainte d'espacement s*** *s'ils appartiennent tous au même niveau; il faut donc chercher de manière géométrique l'aire balayée par le pousseur augmentée de l'aire de l'aire d'interaction du pousseur ; la* ***région de recherche*** *a donc une dimension* ***s+p,*** *l'autre est la hauteur du pousseur; elle est appelée* ***ombre du l'arête poussée.*** *L'ensemble des matériaux de l'ombre ainsi poussés deviennent à leur tour pousseurs et le processus de repoussage s'effectue ainsi récursivement.*

Si seulement les matériaux dans l'ombre du pousseur sont déplacés le résultat peut être une déconnexion électrique (rupture du matériau 1). Pour éviter ceci il est aussi nécessaire de déplacer le matériau dans la ***pénombre*** *située juste au-dessus et au-dessous de l'aire balayée et de* ***hauteur W*** *, largeur du matériau. Ceci a pour effet d'insérer des pliures automatiques du matériau (jogs). Les arêtes ainsi déplacées deviennent aussi à leur tour pousseur.*

Un large barreau placé d'un côté d'une cellule et poussé, la compacte entièrement dans une dimension, compte tenu des règles de conception. A un instant de l'algorithme, tout objet poussé provoque le **repoussage** des objets dans son **ombre** *(figure 15)* et **l'étirement** des objets dans sa **pénombre. Ombre** et **pénombre** de chaque objet ainsi en mouvement sont cherchées de manière géométrique et les objets ainsi trouvés sont déplacés ou étirés, ce qui provoque de nouveaux mouvements et recherches géométriques.

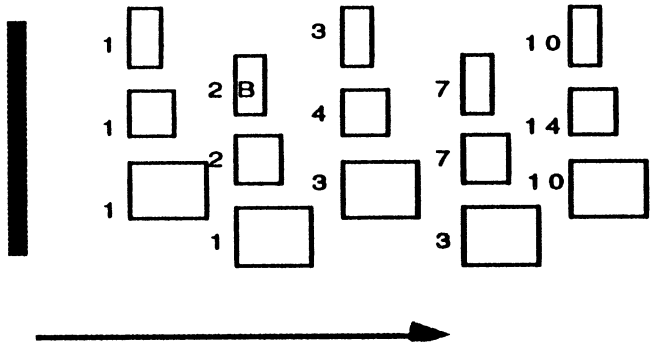


*Figure 16 - **Stabilité de la topologie du layout et repoussage linéaire** - Cette structure en treillis produit pour l'algorithme de **repoussage** récursif ci-dessus un comportement de pire cas exponentiel . L'objet B bouge quand A bouge puis légèrement plus tard quand C bouge. Les nombres à gauche de chaque objet montrent le nombre de déplacements de chaque objet. Mais le **repoussage** préserve globalement la topologie du layout et les relations de voisinage entre les éléments ; cela permet de **chercher de manière géométrique ombres et pénombres sur le layout en entrée** et de remplacer la récursivité en profondeur par la mise à jour d'un **échéancier des éléments** qui doivent être déplacés et de déplacer ces éléments une seule fois en une phase finale [SCO84].*

Cependant, l'algorithme récursif de *repoussage* en profondeur d'abord (depth-first) a un très **mauvais comportement de pire cas** car si on considère des structures en treillis *(figure 16);* le *repoussage* de N rangées demande environ 2^N mouvements.

La transformation de l'algorithme par une technique de recherche en largeur d'abord (width-first) et la **maintenance d'une queue prioritaire - échéancier -** des éléments qui doivent bouger dans **l'ordre croissant des abscisses,** doivent permettre son implémentation interactive. Ombre et pénombre de l'objet le plus prioritaire sont alors cherchées et les objets sélectionnés sont éventuellement ajoutés à l'échéancier et la **quantité de mouvement** (translation) à satisfaire mise à jour. **Ombre et pénombre sont cherchées sur le layout initial** (et non sur le layout modifié) à l'aide des accès topologiques et le layout est mis à jour lorsque la position finale de chaque objet est connue. En supposant que les recherches d'ombre et de pénombre soient réalisées en un temps constant, que la mise à jour de la variation de mouvement soit très négligeable par rapport aux recherches d'ombre et aux déplacements d'objets, **l'algorithme de repoussage devient** alors **quasi proportionnel au nombre d'éléments déplacés**, ce qui doit permettre son utilisation comme fonction d'édition interactive.

Lorsque le layout est structuré, la transparence des sous-cellules est réalisée, comme pour l'algorithme de DRC hiérarchique par la méthode de recherche géométrique. En conséquence, les recherches d'ombre et de pénombre peuvent être faites sur des hiérarchies de cellules non canoniques; l'application des règles de dessin pour les éléments ainsi trouvés dans les cellules permettent soit le calcul du mouvement final de rigides sous-cellules, soit la déformation de sous-cellules molles et de leur contenu, exactement comme si le layout était expansé - créant alors de nouveaux symboles.

Les techniques de recherche géométrique assurent la reconnaissance dynamique des relations d'adjacence entre les motifs du layout. La réduction, l'agrandissement et le déplacement d'un motif provoquent automatiquement contraction, expansion, déplacement de la géométrie qui l'entoure lorsque la stabilité des relations d'adjacence est recherchée; le layout produit est ainsi contracté ou expansé ou déplacé avec une technique semblable à celle présentée pour le *repoussage*. Une des clefs de la flexibilité du layout est le repérage relatif des éléments adjacents et la conservation de leurs positions relatives.

Les diverses techniques de recherche géométrique d'interaction - ombre et pénombre - les techniques de *repoussage,* doivent pouvoir être utilisées aussi par des algorithmes de compaction et de placement de type **recuit simulé** [DREY 87], [SIA 87] pour une optimisation globale de ces méthodes.

Que le concepteur utilise une méthodologie symbolique plus ou moins proche du layout physique, que sa conception soit plus ou moins structurée, le "feed back" apporté par de tels outils d'édition doit considérablement alléger sa tâche.

Chapitre 7

EFICACITE DES ALGORITHMES PROPOSES

POUR LA CAO DES VLSI

1. INTRODUCTION

Le travail décrit dans cette thèse fut entrepris pour améliorer dans une première phase les *outils conventionnels* d'analyse de layouts non structurés. L'amélioration sensible des performances de ces processeurs a ainsi contribué à la validation de la méthode de *recherche géométrique* proposée au *chapitre 4. Un extracteur* de nœuds électriques et un *D.R.C. conventionnels* ont été ainsi très sensiblement améliorés. Diverses formes et tailles de *cellules Stick* non hiérarchiques ont ainsi été analysées. Les plus grosses cellules ont été obtenues par *mise à plat* de la hiérarchie. Pour ces grosses cellules, le comportement des quadtrees proposés a ainsi pu être observé. Les principaux résultats de *l'optimisation des outils conventionnels* par la méthode de *recherche géométrique* sont présentés *en §2.* La recherche expérimentale d'une méthode efficace d'analyse de layouts structurés est expliquée *en §3.* L'application de la méthode de recherche géométrique, proposée au *chapitre 4,* aux fonctions d'un nouvel *éditeur* de circuits, est décrite *en §4.* Le développement de cet éditeur devrait permettre la réalisation de fonctions plus "sophistiquées", proposées *au chapitre 6,* pour l'édition et *l'analyse dynamique de layouts flexibles* ainsi que pour leur *adaptation* à une technologie.

2. L'AMELIORATION DES OUTILS CONVENTIONNELS D'ANALYSE DU LAYOUT

Nous allons présenter les résultats expérimentaux obtenus dans l'amélioration des outils conventionnels par l'utilisation du sous-système de *recherche géométrique* proposé et évalué *au chapitre 4.* L'évaluation théorique du sous-système de *recherche et de gestion des équivalences électriques,* proposé au *chapitre 5,* doit ainsi assurer, dans une première phase, une amélioration sensible des outils conventionnels. Cette dernière optimisation assurera ainsi une optimisation plus complète des outils conventionnels d'analyse de layouts non structurés.

2.1. L'amélioration des outils conventionnels par la méthode de recherche géométrique

Quelques cellules d'environ *80 transistors* réalisées dans un *symbolisme Stick* ont été utilisées pour mettre au point les quadtrees et optimiser les *algorithmes conventionnels de vérification des régles de dessin (D.R.C.)* et *d'extraction des nœuds électriques (E.N.E.).* Les résultats présentés en *figure 4* sont les différents temps de traitement obtenus pour ces outils conventionnels avec des quadtrees différemment paramétrés . *Qd (p, q)* signifie *quadtree de résolution quaternaire p et de résolution binaire q (chapitre 4).* Les meilleurs temps obtenus pour les **D.R.C.** correspondaient pour les *quadtrees (3,3)* à l'égalité des dimensions des régions de sélection du D.R.C. et de partition des quadtrees *(colonne 3).* Les *mixed-quadtrees* qui optimisent les **D.R.C.** produisent aussi de bons résultats pour les **E.N.E.** *(colonne 4).*

taille de la cellule	QD (p,q)	temps du D.R.C.	temps de l' E.N.E.
78 transistors (691 rectangles)	(0,0)	298 s	350 s
	(3,0)	101 s	120 s
	(3,3)	60 s	30 s
	(0,5)	61 s	30 s
	(2,4)	59 s	29 s

Figure 1 - **Optimisation d'algorithmes conventionnels de Vérification des Régles de Dessin (D.R.C.) et d'Extraction des Nœuds Electriques - (E.N.E.) -**

Remarque : *Le nombre de rectangles élémentaires dans une région de sélection de cet outil conventionnel de DRC est de 27 à 5 éléments près, pour ce D.R.C. ; les régions de sélection sont définies par la plus grande garde à vérifier pour l'ensemble du layout , ce qui pour le D.R.C. n'est pas optimal.*

taille de la cellule	temps du D.R.C.	temps de l' E.N.E.	temps de création	temps DRC avant optimisation	temps ERC avant optimisation
78 transistors	59 s	29 s	2.5 s	*298 s*	*350 s*
390 transistors	362 s	210 s	12 s	*7450 s*	*8750 s*
10 transistors	4.6 s	3 s	0.3 s	4.6 s	5.4 s

Figure 2 - **Temps de vérification des Régles de Dessin (D.R.C.) et d'Extraction des Nœuds Electriques (E.R.C.) en fonction de la taille des cellules -**

Les cellules précédemment utilisées pour trouver les meilleurs quadtrees, ont été ensuite *juxtaposées et mises à plat* afin d'étudier le *comportement des quadtrees* pour de plus *grosses cellules*. Lorsque **N cellules** sont ainsi juxtaposées, le **temps du D.RC. conventionnel** obtenu pour ce layout est **d'environ $N^{1.1}$**, ce qui confirme le bon comportement de l'algorithme de recherche géométrique. La *figure 2* donne, en colonnes 2 et 3, les temps du D.R.C. et de l'E.N.E. pour les cellules de base de 78 transistors et pour des *cellules formées de 5 cellules de base - d'environ 400 transistors.* Les *colonnes 5 et 6* permettent la comparaison avec les outils D.R.C. et E.R.C. conventionnels avant leur optimisation; ces temps ont été obtenus en utilisant *l'outil conventionnel avant*

géométriques. Le gain réalisé par l'utilisation des quadtrees pour ces outils est exprimé dans la *figure 3.* La *colonne 4* de la *figure 2* donne les *temps de création* des quadtrees qui optimisent les D.R.C. et E.N.E. ; il est à remarquer qu'ils constituent un faible pourcentage - **< 8 %** - du temps de traitement d'une cellule complète.

taille de la cellule	***Gain D.R.C.***	***Gain E.N.E***
10 transistors	0 %	45 %
78 transistors	80 %	92 %
390 transistors	95 %	98 %

Figure 3 - ***Gains obtenus par l'amélioration des algorithmes conventionnels de D.R.C. et d'E.N.E.*** -

Les *mixed-quadtrees* implémentés sont efficaces pour le *D.R.C conventionnel* à partir de *10 transistors* et à partir de *quelques transistors pour l'E.N.E.* Pour de plus *grosses cellules* les *gains* deviennent vite *très intéressants* si l'on examine alors les gains obtenus par rapport à l'utilisation du même outil avant son amélioration *(figure 3).*

Remarque : L'extraction des nœuds électriques pour cet outil conventionnel n'utilisait pas la méthode ***Union-find*** *et la* ***fusion*** *des composantes connexes utilisait un algorithme en* N^2 *pour une cellule de N éléments. La méthode* ***Union-find*** *doit assurer des performances bien meilleures aux algorithmes d'analyse hiérarchique actuellement développés.*

L'optimisation de ces processeurs conventionnels sur ces layouts non structurés a ainsi permis de trouver les meilleurs paramètres de partition des quadtrees qu'ils utilisent. Comme les régions de sélection de l'ERC et du DRC ont des dimensions bien différentes, l'optimisation générale d'un accès géométrique qui permette à un seul processeur d'effectuer ERC et DRC est plus complexe. Fort heureusement, l'évaluation générale des quadtrees réalisés et l'évaluation ci-dessus ont montré l'intérêt des *quadtrees à faible partition quaternaire (400 à 800 rectangles)* et *forte partition binaire (environ 10 rectangles).* Ce type de *mixed-quadtree* produit une bonne optimisation de l'espace mémoire car il est moins volumineux qu'un quadtree de même performance; de plus il produit un temps de calcul presque optimal pour l'analyse générale du layout (E.N.E. / D.R.C. / E.R.C...).

2.2. L'amélioration des outils conventionnels par la méthode de gestion équivalences électriques

L'efficacité de la technique Union-find a constitué un important critère pour son choix. Les outils conventionnels n'ont cependant pas profité de cette amélioration supplémentaire et la gestion des équivalences électriques est toujours de l'ordre de O (N^2) pour une cellule de N éléments.

3. L'ANALYSE HIERARCHIQUE DU LAYOUT

3.1. Graphe d'équivalence et extension de la technique Union-find

Le *chapitre 5* a proposé aussi une méthode d'analyse hiérarchique de la *connectivité et des propriétés électriques* sur des layouts qui sont, pour la relation d'équivalence électrique, structurés selon une **hiérarchie stricte**. En conséquence l'algorithme **Union-find** de gestion de la connectivité électrique a pu être extensivement utilisée pour l'analyse hiérarchique. La complexité de l'algorithme **union-find** - meilleure que O (N log N) pour N recherches d'équivalence *sur une collection de N éléments primaires* - et la faible proportion d'éléments primaires des sous-cellules à considérer dans l'analyse incrémentale d'une cellule assurent ainsi une analyse incrémentale performante *(cf. chap2.§6.2.)*.

En effet, *sur une collection de n éléments primaires,* la complexité théorique de l'algorithme de recherche géométrique est pour n *recherches ponctuelles (pics)*, entre O (n log n) et O (n $\log^2$ n); elle est en pratique de l'ordre de O ($n^{1.1}$), et toujours très inférieure à O($n^{3/2}$) *(chapitre 4 & 5).* La complexité théorique de l'algorithme *Union-find* est considérée meilleure que O (n logn) pour n recherches d'équivalence sur cette collection *(chap. 3).* Ainsi, sur une cellule de n éléments primaires, l'analyse conventionnelle est réalisée pratiquement en un temps de l'ordre de O ($n^{1.1}$) *(annexe 5).* Le nombre d'*éléments primaires* - ports de la cellule - à réanalyser, lorsqu'une cellule de n éléments primaires est instanciée, est au plus d'ordre O ($n^{1/2}$), si on considère ce nombre au plus proportionnel au périmétre du modèle. La combinaison de ces propriétés assure une analyse incrémentale performante puisque le produit de ces complexités est toujours très inférieur à 1 *(chap. 3.§6.2).* La **faisabilité de la méthode** est ainsi assurée avec une complexité des temps cumulés de **l'analyse incrémentale** au plus en O (N) pour un circuit de N éléments, et un nombre total d'éléments à considérer dans l'analyse de la cellule de plus haut niveau, qui représente le circuit complet, au plus d'ordre O($N^{1/2}$).

3.2. La vérification des règles de dessin par la technique des halos

L'ignorance ou *l'expansion des sous-structures* utilisées dans une description structurée du layout, a constitué une étape intermédiaire d'extension des *outils conventionnels* améliorés par la *recherche géométrique.* Cette étape a permis en particulier la validation de l'accès géométrique sur de bien plus grosses cellules; le temps de vérification s'est révélé être *quasi proportionnel* au nombre de cellules juxtaposées et expansées ($N^{1.1}$ pour N cellules terminales) sur des hiérarchies à deux niveaux *(§2.1).*

Après cette première phase d'amélioration des outils conventionnels de vérification, la technique des **halos** fut essayé dans le but d'améliorer encore la vérification de layouts hiérarchiques *sans recouvrement de sous-cellules (chap. 2).* Après la vérification d'une cellule, son halo était alors construit par filtrage des éléments primaires inclus dans son halo, afin de remplacer la cellule dans les niveaux supérieurs de la hiérarchie *(fig. 4).* Un algorithme de construction dynamique du halo d'une cellule, qui évite la sauvegarde de cellules halos, est donné dans *l'annexe 2.*; cependant utilisé comme tel il constitue alors plutôt un filtrage hiérarchique, avec comme inconvénient la génération de fausses erreurs *(chap.2).* Diverses informations hiérarchiques devraient aussi être prises en compte afin d'éviter cet inconvénient, ce qui rend alors la méthode plus complexe et moins performante.

La *figure 5* présente la *taille des halos* qu'il est nécessaire d'extraire pour diverses tailles de cellules afin d'assurer un D.R.C. hiérarchique selon la méthode présentée sur la *figure 4.* La *colonne 1* indique la *taille des cellules.* La *colonne 2* indique la *taille des halos* correspondants en pourcentage par rapport à la taille des cellules, et en conséquence le *supplément de mémoire* utilisé. La *colonne 3* indique le *gain partiel* réalisé pour le D.R.C. d'une cellule qui remplacerait cette instance de cellule par son halo *(cf. figure 2).*

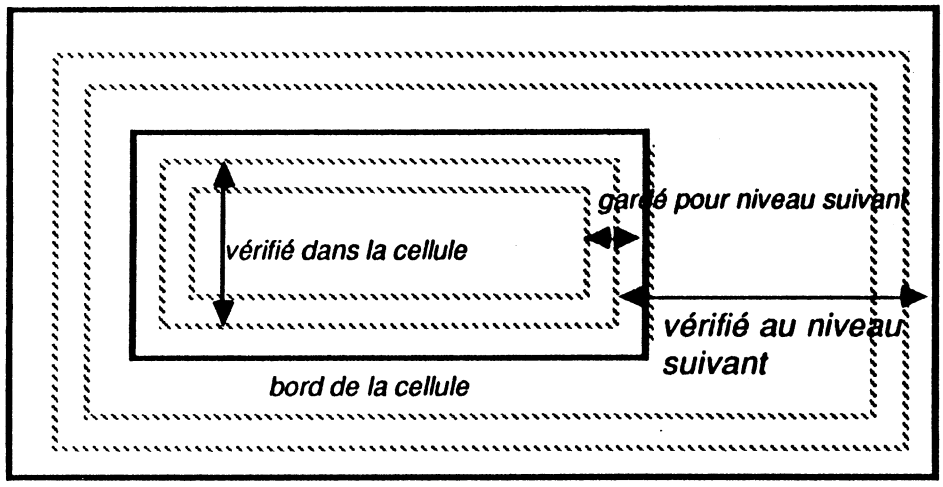


Figure 4 - ***Une méthode de halos utilisée pour la vérification des règles de dessin*** *- Extraction d'un* **halo** *représentant la cellule dans le contexte qui l'utilise, halo large de deux fois la règle maximale pour le DRC hiérarchique [WAG 84]. Le* **halo** *constitue un* ***résumé*** *de la cellule pour les contextes qui vont utiliser cette cellule. Il est ainsi extrait après l'analyse et la vérification de la cellule et sauvegardé dans la base de données des circuits. Sa construction dépend de* ***l'analyse effectuée*** *et des ressources nécessaires à une utilisation extensive de l'outil conventionnel d'analyse. Sa construction dépend aussi du type de méthodologie utilisée pour structurer la conception. Ainsi pour un layout structuré sans restrictions des recouvrements de sous-cellules, le* **halo** *de la cellule est aussi large que la cellule, et l'analyse hiérarchique devient alors inefficace. Une transformation de la description des tableaux de petites cellules en une autre, plus hiérarchique, du même circuit est nécessaire [WAG 84].*

L'extension du ***D.R.C. Stick conventionnel*** par la *technique des halos* ne peut être envisagée qu'avec des halos larges d'au moins de 2 fois la règle maximale des règles de dessin. Il est difficile d'utiliser autrement ce D.R.C. conventionnel qui n'effectue pas entre deux éléments des vérifications purement binaires et doit éventuellement analyser leur voisinage. Or la règle de garde maximale d'un ensemble de règles Stick est de 6.5 λ – avec λ = 3.00 microns; cette garde maximale est donc pour des cellules de surface 100 λ * 100 λ et d'environ 80 transistors relativement élevée. Les valeurs des contraintes maximales observées sur des ensembles de règles de dessin des layouts physiques paraissent généralement bien inférieures à cette valeur.

L'extension du ***D.R.C. Stick conventionnel,*** par la *technique des halos,* nous a donc paru peu intéressante pour la vérification de circuits constitués essentiellement de petites cellules. En effet, pour des cellules constitués par des tableaux de petites cellules, le gain peut être nul et même globalement négatif. L'analyse de ces layouts peut être alors réalisée par groupage d'un certain nombre de cellules [WAG 84]. La reconnaissance de la régularité et l'étude des interactions relatives entre sous-cellules sont alors aussi utiles *(Chap. 2).*

taille de la cellule	***taille du halo***	***réduction analyse instance***
10 transistors	100 %	0 %
20 transistors	100%	23 %
78 transistors	46 %	75 %

Figure 5 - ***Taille des halos en fonction de la taille des cellules*** *(colonne 1) et réduction partielle de la vérification d'une instance de cette cellule dans le contexte qui l'utilise. Remarque : Une réduction du temps de vérification peut être obtenue avec une cellule halo égale à la cellule elle-même grâce à méthode présentée en figure 4; seuls les éléments d'une sous-cellule situés à une distance du bord inférieure à la garde maximale sont revérifiés.*

3.3. L'algorithme général proposé pour la vérification hiérarchique

Comme l'outil conventionnel de vérification des règles de dessin stick ne pouvait être efficacement utilisé sur tous les types de conceptions structurées, la

conception d'un outil général de vérification hiérarchique et *incrémentale* a donc été entreprise - *chapitre 5 et 6* - selon des principes présentés au *chap. 2.§6.* La réalisation de cet outil de CAO est en cours. Son évaluation expérimentale confirmera son efficacité théorique. Cet outil général d'analyse incrémentale et hiérarchique réalise, sur une même structure de données, extraction de connectivité électrique, vérification des règles de conception et extraction des paramètres électriques. L'extension interactive de cet outil de CAO a été prévue lors de sa conception puisque *efficacité et gestion dynamique* des données ont constitué deux critères de choix et de conception des *méthodes de base de recherche géométrique et d'équivalence électrique.*

L'indépendance, par rapport à la technologie et par rapport à la base de données des circuits, de cet outil d'analyse hiérarchique, doit permettre son développement dans le système de CAO en cours de conception - *extension du projet CVS et éditeur généralisé.*

3.4. Complexité théorique de l'algorithme de vérification hiérarchique

La complexité théorique de l'algorithme de *recherche géométrique* est pour n *recherches ponctuelles* sur layout constitué de n éléments, entre O (n log n) et O (n $\log^2$ n) *(chapitre 4 & 5).* Ainsi la *vérification conventionnelle* des règles de D.R.C. est réalisée avec un temps, en pratique de l'ordre de O ($N^{1.1}$), et toujours très inférieur à O ($n^{3/2}$) pour un layout de n éléments *(annexe 5).*

Dans la *méthode des halos,* le nombre *d'éléments primaires* à vérifier, lorsqu'une cellule de n éléments primaires est instanciée, est au plus d'ordre O($n^{1/2}$), si on considère ce nombre au plus proportionnel au périmétre du modèle. La combinaison de ces propriétés assure une analyse incrémentale performante puisque le produit de ces complexités est inférieur à 1 *(chap. 3.§6.2).* La **faisabilité de la méthode** est ainsi assurée avec une complexité des temps cumulés de **l'analyse incrémentale** au plus en O (N) pour un circuit de N éléments, et un nombre total d'éléments à considérer dans l'analyse de la cellule de plus haut niveau, qui représente le circuit complet, au plus d'ordre O($N^{1/2}$).

Diverses causes ont été observés dans les cas réels, qui produisent un mauvais comportement à la *technique des halos,* qui paraît cependant avoir un bon comportement théorique *(chapitre 2).* L'une de ces causes est la taille des cellules; en effet cette technique ne commence à être efficace que pour des cellules contenant un nombre d'éléments N suffisamment grand, pour qu'il y ait une réduction dans le temps d'analyse. Une autre cause est l'application de la valeur de la contrainte maximale pour la construction des halos, principalement dans le cas du D.R.C. symbolique précédemment considéré. Une formulation très locale des règles dessin devrait diminuer la valeur de la règle maximale.

Dans une cellule l'algorithme de D.R.C. hiérarchique proposé recherche par voisinage les interactions d'éléments, et atteint ainsi récursivement tous les *éléments primaires* suffisamment proches l'un de l'autre pour produire une violation. Les recherches de voisinages sont effectuées en prenant soit le maximum de toutes contraintes pour les éléments structurés, soit le maximum des contraintes associées à chaque type, ou couleur, *d'élément primaire.* On peut donc affirmer que le nombre

d'éléments primaires vérifiés, lors du test incrémental d'une cellule, par l'algorithme de D.R.C. proposé au *chapitre 6,* est inférieur à celui qui aurait été vérifié par la *technique des halos.* La faisabilité théorique de la technique des halos, démontrée *au chapitre 2,* assure alors la faisabilité de l'algorithme hiérarchique de D.R.C. proposé au *chapitre 6.*

Pour des hiérarchies cohérentes, la **faisabilité de la technique** des halos est assurée avec une complexité des temps cumulés de **l'analyse incrémentale** au plus en O (N) pour un circuit de N éléments, et un nombre total d'éléments à considérer dans l'analyse de la cellule de plus haut niveau, qui représente le circuit complet, au plus d'ordre $O(N^{1/2})$. Le D.R.C. hiérarchique proposé par analyse des interactions dans la profondeur de la hiérarchie a donc, pour des hiérarchies strictes, une efficacité théorique au moins équivalente.

L'algorithme de D.R.C. hiérarchique proposé assure de plus la vérification des hiérarchies de layouts non cohérentes, qui autorisent les recouvrements de sous-cellules. Il évite la gestion ou la reconstitution de cellules halos. Cet algorithme permet, en plus de l'analyse incrémentale, une **vérification très locale** des circuits, par utilisation du sous-système de recherche géométrique.

L'algorithme de D.R.C. hiérarchique proposé à, par rapport à une technique de halos, l'avantage de reconnaître et de **ne vérifier qu'une seule fois une même interaction de sous-cellules.** Cet algorithme a ainsi un comportement bien meilleur que la *technique des halos,* pour des circuits très répétitifs mais pas nécessairement très hiérarchiques, en particulier pour les tableaux de cellules; le nombre d'éléments effectivement vérifiés pour ces tableaux est alors considérablement réduit.

3.5. La qualité des descriptions hiérarchiques

Le *chapitre 2* a permis d'observer que certaines qualités d'une description hiérarchique assurent l'efficacité de toutes les stratégies d'analyse hiérarchique. Trois critères de qualités sont ainsi à considérer :

1. **la régularité** de la description, qui peut être le rapport du nombre de transistors dessinés au nombre de transistors effectifs, ou encore le rapport de la longueur de la description hiérarchique par la longueur de la description non hiérarchique équivalente.
2. la **cohérence** de la description, mesurée par un faible ratio de recouvrement ou rapport moyen des recouvrements de chaque modèle de cellule.
3. la **qualité de la partition hiérarchique**, caractérisée par une partition des données propre à assurer une analyse optimale

Ces critères de qualité mesurent ainsi la qualité des descriptions hiérarchiques pour une prédiction des performances de leur analyse hiérarchique et incrémentale. L'influence d'une transformation de la hiérarchie de certaines conceptions peut ainsi profitable à certaines analyses hiérarchiques; elle sera alors mesurable par le gain de performance pour l'analyse considérée, et sera éventuellement prévisible par l'analyse de la qualité de la hiérarchie. Par exemple une description plus hiérarchique d'un tableau de cellules est profitable à l'analyse de ce tableau par la méthode des halos *(chapitre 2).*

4. L'APPLICATION AUX FONCTIONS D'EDITION

4.1. La réalisation d'un éditeur de circuit performant

La recherche géométrique par hiérarchie de quadtrees a été, suite à l'évaluation son évaluation expérimentale, choisie comme méthode de recherche dans le développement d'un nouvel éditeur de circuits, en interface avec la base de données *COSMIC du projet CVS [JUL 86].* Le sous-système de recherche géométrique a été intégré selon l'interface générale d'utilisation présentée au *chapitre 4 et dans l'annexe 1.* Qualitativement, ce sous-système de recherche offre une bonne prestation pour diverses fonctions de base de cet éditeur multi-fenêtres; quantitativement, son évaluation dans cet éditeur est actuellement hors de portée de cette étude. Cet éditeur est ainsi capable d'assurer efficacement la visibilité du layout à travers la hiérarchie de conception.

Parallèlement, le principe de *l'extracteur conventionnel* de connectivité électrique réalisé dans l'ancien système intégré de CAO, *CASSIOPEE [BEY 82], [LEC 85], [BEY 82],* a été repris pour développer un *extracteur de graphe* qui analyse le layout d'une cellule éditée selon certaines règles de dessin et de mise en contact électrique. Son but est d'extraire de *manière géométrique* les *informations de connectivité électrique* au moyen d'un *post-processeur,* afin de compléter l'information structurelle de la cellule dans la base de données des circuits - *base COSMIC du projet CVS [JUL 86].*

4.2. Analyse, adaptation et contrôle interactifs des layouts édités

L'édition de *layouts symboliques et flexibles* est une des clefs de l'adaptation de ces layouts à diverses technologies. *L'analyse incrémentale* des layouts structurés est une des clefs de l'*analyse dynamique* et interactive des cellules en cours d'édition, au fur et à mesure de l'introduction des données, car elle permet de travailler sur un nombre de données très restreint *(chap. 2).*

Les deux techniques algorithmiques de base de *recherches géométrique* et *d'équivalence* utilisées par l'analyse hiérarchique, s'adaptent complètement à la *dynamique* du processus d'édition. Les algorithmes d'analyse et de vérification hiérarchique qui ont été développés seront ainsi facilement adaptables à l'analyse et la vérification interactives et locales des layouts de VLSI *(chap. 6.§6).*

Sans réorganisation particulière, les techniques algorithmiques de base de *recherches géométrique* et *d'équivalence* utilisées par l'analyse hiérarchique, sont transparentes à l'utilisateur et doivent permettre l'implantation interactive de fonctions d'édition et de contrôle performantes et "sophistiquées" *(chapitre 6.§6.).* Ces techniques assurent l'édition de *layouts flexibles,* et *l'adaptation* de ce *layout flexible* à une technologie afin de générer le *layout physique* correspondant. Elles font ainsi considérablement évoluer les techniques d'édition de layouts des conceptions VLSI.

CONCLUSION

1. CONTRIBUTIONS DES RECHERCHES A L'EDITION ET A L'ANALYSE DES LAYOUTS DE VLSI

L'accroissement de la complexité des circuits VLSI a provoqué un accroissement considérable des tâches de description, d'analyse et de vérification des masques. Il a donc été nécessaire d'utiliser des algorithmes efficaces pour l'édition interactive et l'analyse conventionnelle des descriptions des masques des circuits (layout).

Cette thèse a ainsi exploré diverses méthodes des **recherche géométrique**. Les méthodes qui permettaient d'effectuer des recherches dynamiques, sur des données évolutives ont été particulièrement étudiées *(chap. 3)*. Des **quadtrees adaptatifs** particuliers appelés **mixed-quadtrees** ont ainsi été conçus, évalués et testés *(chap. 4)*. Avec cette structure de données, la complexité théorique de l'algorithme de recherche géométrique est pour n *recherches ponctuelles (pics)* sur un layout non hiérarchique constitué de n éléments, entre O (n log n) et O (n $\log^2 n$); elle est en pratique en O ($n^{1.1}$) pour n recherches de même aire *(annexe 5)*. Cette méthode de recherche géométrique a été testée graphiquement dans l'éditeur de circuits du système intégré de CAO ***Cassiopée,*** *réalisé au CNET de Grenoble [BEY 82], [LEC 84].* Son intégration dans ce système de CAO, est restée alors partielle et expérimentale puisque le système de CAO *Cassiopée* était destiné à être remplacé, en partie, par le *projet Européen* ***CVT*** , et suivi par le *projet Européen* ***CVS***.

Un *vérificateur* de règles de dessin et un *extracteur* de graphe électrique *conventionnels* ont été considérablement améliorés par l'utilisation des recherches géométriques *(chap. 7 & annexe 5)*. La complexité du *vérificateur* conventionnel est en pratique de l'ordre de O ($n^{1.1}$) pour un layout de n éléments *(annexe 5)*. La complexité de ces analyses est toujours très inférieure à O ($n^{3/2}$).

Cette thèse a aussi exploré diverses méthodes des **recherche d'équivalence électrique** afin de reconnaître si deux **éléments atomiques** (non structurés) du layout sont électriquement connectés. La méthode **Union-Find** a paru avoir, sur un ensemble de rectangles, un bon comportement dynamique; elle a une complexité théorique meilleure que O (n log n) pour n recherches d'équivalence électrique sur un layout non hiérarchique constitué de n éléments atomiques. Cette méthode a donc été développée afin d'améliorer l'analyse conventionnelle des conceptions VLSI.

Dans la but de réduire la tâche de description des layouts de VLSI, des descriptions hiérarchiques sont utilisées. Pour analyser et vérifier des layouts ainsi structurés, des méthodes d'analyse hiérarchique et incrémentale ont été développées *(chap. 2 , chap. 5 & chap. 6)*.

Cette thèse a exploré les diverses hiérarchies de description des masques d'un circuit et les diverses stratégies utilisées pour les analyser de manière efficace. Les méthodes de conception restrictives rendent les analyses hiérarchiques moins complexes; cependant des méthodes générales essaient aussi de réduire la complexité des analyses hiérarchiques.

On a remarqué que la plupart des stratégies d'analyse hiérarchique utilisent principalement des restrictions géométriques; ces stratégies analysent alors des **hiérarchies cohérentes** dans lesquelles les sous-cellules ne peuvent avoir de recouvrement; ces *hiérarchies cohérentes* peuvent être éventuellement obtenues par transformation des hiérarchies non restrictives *(chap. 2).*

Cependant des restrictions fonctionnelles et structurelles sont aussi implicitement ou explicitement utilisées pour réduire la complexité des descriptions hiérarchiques et de leur analyse. Un grand nombre de hiérarchies sont ainsi transformables en hiérarchies dans lesquelles les données ne peuvent être partagées qu'entre deux niveaux de hiérarchie adjacents. Ces hiérarchies ont alors été appelées **hiérarchies strictes;** ce type de hiérarchie réduit la complexité de l'extraction hiérarchique et incrémentale du réseau électrique *(chap. 3 & chap. 5).* Ce type de hiérarchie permet une extension de l'algorithme d'équivalence **Union-Find** *(chapitre 2)* et la construction d'une *représentation compacte et hiérarchique* du réseau électrique que le layout représente *(chap. 5).* Equivalences et paramètres électriques sont ainsi propagées de manière ascendante à travers la hiérarchie de conception. Chaque modèle de cellule défini lors de la construction hiérarchique du circuit est extrait indépendamment des contextes qui l'utiliseront sous forme d'**instance.** La structure de chaque cellule est ainsi préservée par l'extracteur; elle est complétée par son graphe électrique qui **résume** pour les contextes qui vont **l'instancier** l'analyse électrique réalisée sur sa définition *(chap.5 & 6).* Un algorithme récursif de *mise à plat* du réseau électrique obtenu permet ensuite, si nécessaire la génération d'un graphe électrique sans hiérarchie. Le graphe électrique sans hiérarchie ainsi obtenu pourra servir d'entrée aux divers simulateurs *(chap. 6).*

Cependant, la non limitation des recouvrements de sous-cellules rend les extractions plus complexes et plus approximatives *(chap. 5).* Pour ces hiérarchies, dites **hiérarchies incohérentes,** il est impossible de *résumer* les résultats d'une vérification hiérarchique des règles de dessin par l'extraction d'un **halo** capable de remplacer la cellule dans les contextes qui l'utilisent, et cependant plus petit que la cellule elle-même. L'analyse et la vérification des interactions géométriques doivent alors être réalisées de manière hiérarchique et incrémentale, dans la *profondeur de la hiérarchie* de conception *(chap. 2 & chap. 6).*

Une **hiérarchie de quadtrees** a alors été associée à la hiérarchie de description d'un layout. La méthode de *recherche géométrique* a ainsi été complétée par une **interface utilisateur** permettant la création d'une *forêt de quadtrees (chap. 4).* Cette hiérarchie de quadtrees permet une analyse efficace des relations d'adjacence et de voisinage dans la *profondeur de la hiérarchie (chap. 4)* . Il est ainsi possible de vérifier directement ces interactions par rapport à un ensemble de règles de dessin liées à la technologie utilisée pour réaliser le circuit.

On a donc développé un algorithme de vérification hiérarchique des règles de dessin, qui vérifie les interactions géométriques dans la profondeur de la hiérarchie de conception *(chap. 6).* Le sous-système de recherche géométrique et la représentation compacte du réseau électrique permettent de reconnaître l'éventuelle connexion électrique de deux éléments atomiques du layout, plus ou moins imbriqués dans la hiérarchie de conception *(chap. 5).* Cette reconnaissance évite

les mauvais diagnostics et constitue une extension hiérarchique de l'algorithme d'équivalence *Union-find (chap. 6).*

L'analyse hiérarchique et incrémentale du layout est réalisée par des tests hiérarchiques effectués à chaque étape de sa construction. Cette *analyse incrémentale* du layout d'un circuit, permet de travailler sur un ensemble de données considérablement réduit par rapport à une simple analyse conventionnelle, et aussi par rapport à une simple analyse hiérarchique *(chap. 2).* Le temps global de l'analyse complète du circuit est de plus fragmenté et réparti tout au long du processus de conception *(chap. 2).* Lorsque les hiérarchies sont *cohérentes,* ou *presque cohérentes,* le nombre d'éléments atomiques à réanalyser lorsqu'une cellule est instanciée dans un contexte est assez faible pour produire, globalement, une analyse hiérarchique au plus en O (N) pour un layout constitué par N éléments atomiques*(chap. 2, chap. 5 & chap. 7).* Le temps ainsi dépensé à l'analyse de la conception se répartit tout au long du processus de conception et peut de plus être très inférieur à O (N) pour un circuit très répétitif ; pour de tels circuits, des temps d'analyse en O ($N^{1/2}$) sont généralement observables et pour des circuits très réguliers des temps bien meilleurs encore peuvent être obtenus.

En valeur absolue, l'analyse hiérarchique est d'autant plus performante que le nombre d'éléments à réanalyser est plus faible et que les algorithmes de base utilisés pour la recherche des interactions géométriques et la reconnaissance des équivalences électriques entre les éléments atomiques sont plus performants. Dans une *hiérarchie séparée* chaque *cellule non terminale* est constituée uniquement d'instances de cellules. La hiérarchie séparée est aussi une hiérarchie cohérente; lors de *l'analyse incrémentale* d'une *cellule non terminale* d'une hiérarchie séparée, le nombre d'éléments atomiques à réanalyser est pour chaque instance de cellule est au plus en O ($N^{1/2}$), et donc le nombre total d'éléments atomiques à analyser est aussi au plus en O ($N^{1/2}$), pour une cellule contenant N *éléments atomiques.* Ainsi, la faisablité et l'efficacité théorique des analyses hiérarchiques est assurée pour ces hiérarchies, et probablement aussi pour une classe beaucoup plus larges de bonnes hiérarchies, encore *assez cohérentes* et *suffisamment séparées (chap. 2 & chap. 7).*

Les sous-systèmes de recherche géométrique et d'équivalence qui ont été développés, évalués et testés ont ainsi été particulièrement optimisés afin d'assurer un comportement maximal des algorithmes hiérarchiques développés *(chap. 4, 5 & 6).* Pour des *hiérarchies cohérentes,* le sous-système de recherche géométrique assure, pour chaque test incrémental du vérificateur hiérarchique développé, la sélection et la vérification d'un nombre d'interactions, en valeur absolue, très inférieur à celui produit par une *technique de halos.* En effet une technique de halos utilise la garde de dessin maximale pour rechercher les interactions, alors que dans l'algorithme proposé, la recherche des interactions est induite par la vérification de chaque élément atomique selon son type (niveau ou pseudo niveau de masque); de plus deux éléments d'une même cellule ne sont vérifiés qu'une seule fois. Ainsi la faisabilité de la technique des halos pour les *hiérarchies cohérentes* implique la faisabilité de l'algorithme de vérification hiérarchique *proposé au chapitre 6 (chap. 2 & 7).*

La plupart des hiérarchies de conception sont assez *faiblement incohérentes ;* leur *incohérence* se mesure par un ratio de recouvrement *(chap. 7).* Ces

hiérarchies devraient ainsi être aussi vérifiées de manière efficace par le vérificateur hiérarchique développé pour vérifier les hiérarchies incohérentes *(chap. 6).*

La reconnaissance de la régularité des descriptions hiérarchiques améliore encore sensiblement les analyses hiérarchiques de la plupart des circuits et a été largement prise en compte par les algorithmes développés *(chap. 6).* Une certaine qualité de la description hiérarchique d'un layout constitue un facteur déterminant dans l'efficacité de son analyse hiérarchique et la transformation de la description d'un layout en une autre plus hiérarchique et équivalente peut améliorer, de manière significative, l'analyse et la vérification du circuit *(chap. 2 & chap. 7).*

2. LES PERSPECTIVES

Si l'on a prouvé de manière théorique la faisabilité, avec une bonne complexité, des analyses et vérifications hiérarchiques et incrémentales des layouts de VLSI, la preuve pratique de l'efficacité des algorithmes d'analyse hiérarchique développés est encore à faire. Les diverses optimisations prises en compte par ces algorithmes doivent être ajustées afin d'obtenir une optimisation globale des analyses et vérifications hiérarchiques sur un grand nombre de circuits et pour une large classe de conceptions.

L'évaluation expérimentale du comportement des algorithmes hiérarchiques sur des cas réels demande la réalisation d'une interface avec la base de données ***COSMIC*** *du projet Européen* ***CVS*** *[JUL 86],* réalisée dans le département *RCA du CNET de Grenoble,* et dont une première version est actuellement utilisable. Le comportement des algorithmes d'analyse et vérification hiérarchique qui ont été développés devra ainsi être étudié sur un ensemble de circuits très divers. Il serait intéressant de mesurer, par un algorithme hiérarchique, la **qualité de la description** hiérarchique d'un layout et de la comparer à l'efficacité de son analyse hiérarchique. Trois qualités sont alors à considérer, la régularité, la cohérence, et la qualité de la partition hiérarchique des données. La transformation préalable de certaines hiérarchies pourraient alors s'avérer profitable à certains tests hiérarchiques et constituer un raffinement de ces méthodes d'analyse *(chap. 2 & 7).*

Les règles de dessin symbolique utilisées par le vérificateur conventionnel ont paru difficiles à adapter à une vérification hiérarchique; l'application exhaustive de ces règles génère de larges et coûteuses recherches géométriques et des vérifications très redondantes. Une simplification des vérifications de base a donc été recherchée. On a aussi cherché à réduire le nombre de contraintes à vérifier en effectuant des vérifications très locales. Les vérifications de base les plus locales et les moins exhaustives semblent les plus profitables aux analyses hiérarchiques; elles sont plus simples à implémenter et induisent de meilleures performances. D'autres modèlisations des règles de dessin ont ainsi été observées et une modèlisation très différente et très locale a été proposée *(chap. 6).* Une étude plus exhaustive des diverses modélisations des règles de dessin permettra d'évaluer leur influence dans l'efficacité de l'algorithme de vérification. Cette étude profitera également aux algorithmes de compactage et d'adaptation des *layouts flexibles* à un ensemble de règles de dessin *(chap. 6).*

Les techniques algorithmiques de base ont été implémentées sous la forme de **sous-systèmes de recherche géométrique et d'équivalence**; des interfaces d'utilisation permettent à divers systèmes et outils de CAO, d'intégrer facilement ces sous-systèmes.

Le **sous-système de recherche géométrique,** par hiérarchie de quadtrees, a été, suite son évaluation expérimentale, choisi comme méthode de recherche géométrique dans le développement d'un nouvel éditeur de circuits, en interface avec la base de données ***COSMIC*** *du projet Européen* ***CVS*** *[JUL 86],* destiné à remplacer le *système Cassiopée.* **Le** *sous-système de recherche géométrique* a été intégré selon **l'interface générale d'utilisation présentée au *chapitre 4 et dans l'annexe 1.*** Qualitativement, ce sous-système de recherche a paru offrir une bonne prestation pour réaliser les diverses fonctions de base de cet éditeur généralisé et multi-fenêtres; quantitativement, son évaluation dans cet éditeur est actuellement hors de portée de cette étude. Actuellement le *sous-système de recherche géométrique et d'équivalence* assure une édition simplifiée et efficace, selon un ensemble de règles de dessin de mise en contact électrique, des diverses cellules qui composent un layout structuré ; un post-processeur réalise ensuite l'extraction de la connectivité electrique par recherche géométrique et la mise à jour, dans la base de données COSMIC, de l'information structurelle.

Les deux techniques algorithmiques de base de *recherches géométrique* et *d'équivalence* utilisées par l'analyse hiérarchique, s'adaptent complètement à la *dynamique* du processus d'édition. L'adaptation interactive des algorithmes d'analyse et de vérification hiérarchiques développés assurera la réalisation de tests très locaux. Sans réorganisation particulière, ces techniques sont transparentes à l'utilisateur et doivent permettre l'implantation interactive de fonctions d'édition, d'analyse et de contrôle performantes et "sophistiquées", locales ou au contraire très globales *(chapitre 6).*

Les divers algorithmes qui ont été proposés et développés seront ainsi en partie repris dans le cadre du *projet Européen* ***CVS,*** *par le département de Recherche en Conception Assistée du CNET de Grenoble.* Des fonctions d'édition plus "sophistiquées" devraient ainsi être proposées aux concepteurs. Ces techniques devraient assurer l'édition de *layouts flexibles,* ainsi que *l'adaptation* de ce *layout flexible* à un ensemble de règles de dessin liées à une technologie *(chap. 6).* L'utilisation de ces outils de CAO fera ainsi considérablement évoluer les techniques d'édition de layouts des conceptions VLSI, et il serait intéressant de construire un système de CAO à partir de ces mécanismes.

ANNEXE 1

LE QUADTREE

STRUCTURES DE DONNEES ET ALGORITHMES

1. LE QUADTREE : STRUCTURE DE DONNEES

1.1. Le quadtree et les éléments insérés dans un quadtree

```
type  root = record
          rac : ^quad_node;                    { racine du quadtree }
          thresh : quad_treshold;              { paramètres de la partition des données / résolution}
          bbox : tyrect;                       { X1,Y1, X2,Y2 }
          ident : integer;                     { identification du quadtree }
      end;
type proot = ^root;

type  quad-node = record                       { les différents nœuds d'un quadtree }
      case  nd : noeud of
          qtree : ( x-médian, y-médian, ul, ur, ll, lr,  father : ^quad_node); { nœud quaernaire }
          btree : ( first, left, right, dad : ^quad_node);          { nœud binaires }
          elem : ( item : tyitem; prev,  next : ^quad_node);        { élément }
          lib :   ( suiv : ^quad_node);                             { nœud libre }
      end;

type tyitem = record
          numelm: ptr_enreg;                   { référence à l'enregistrement de l'élément }
          itemb : tyrect;                      { rectangle enveloppe de l'élément }
      end;
```

Types utilisés par les programmes d'application

```
type tyrect = record                           { coordonnées des rectangles }
          x1, y1, x2, y2 : real;
      end;
```

1.2. Les enregistrements et la référence des éléments insérés dans un quadtree sont définis par l'application; exemple :

```
ptr-enreg = ^element;
element = record
          ..
          end;
```

1.3. Eléments sélectionnés par une recherche géométrique :

```
type  elem-list = record                       { éventuellement rendus dans une liste }
          typ : integer;                       { couverture / intersection : - claire : =1, - possible : =0 }
          elem : ptr_enreg;                    { référence à l'élément selectionné }
          next : ^elem_list;                   { prochain élément sélectionné }
      end;
type ptr-list = ^elem-list;                    { liste des éléments sélectionnés par une recherche }
```

1.4. Paramétrage du quadtree, définition de la résolution

```
type typaram = record                          { paramètres de partition  quarenaire et binaire }
      case util : integer of                   { partition absolue = 0; partition relative = 1 }
          0 : ( q-seg, b-seg : real);          { seuils de partitions quaternaire et binaire (dimension) }
          1 : ( q-depth, b-depth : integer);   { résolution fixe des partitions quaternaires et binaires }
      end;
quad-tresh = typaram;
```

Remarque : *Les types énumérés ont été évités pour utilisation par desprogrammes d'application écrits en langage C ou ADA de ce systeme écrit en pascal.*

2. INTERFACE D'UTILISATION

Actions offertes par l'interface du systeme de recherche géométrique aux programmes d'application

action **begin-qd** (qd-thresh); **{ Interface }**
{ cette action initialise l'ensemble des quad-trees et leur résolution par défaut }
entrée : **qd-thresh** : paramètres de partition des données, donné par défaut pour l'ensemble des quadtrees.

action **end-qd**; **{ Interface }**
{ cette action ferme l'ensemble des quad-trees }

action **create-qd** (qd-ident, qd-bbox, qd-thresh); **{ Interface }**
*{ cette action créé le quadtree d'une collection d'éléments / cellule et **définit sa résolution** }*
entrée : **qd-ident** : identification du quadtree, **qd-bbox** : rectangle enveloppe de collection d'éléments, **qd-thresh**: seuils de partage.

action **mode-create-qd** (qd-ident, qd-bbox); **{ Interface }**
*{ cette action créé le quadtree d'une collection d'éléments / cellule avec une **résolution définie** globalement pour l'**ensemble des quadtrees** par l'action **begin-qd** d'initialisation de cette forêt }*
entrée : **qd-ident** : identification du quadtree, **qd-bbox** : rectangle enveloppe de collection d'éléments.

action **add-qd** (qd-ident, item-bbox, item-ref); **{ Interface }**
{ cette action place un élément dans le quadtree d'une collection d'éléménts / cellule }
entrée : **qd-ident** : identification du quadtree, **item-bbox** : rectangle enveloppe de l'élément, **item-ref** : adresse de l'enregistrement de l'élément.

action **supr-qd** (qd-ident, item-bbox, item-ref); **{ Interface }**
{ cette action supprime un élément du quadtree d'une collection d'éléménts / cellule }
entrée : **qd-ident** : identification du quadtree, **item-bbox** : rectangle enveloppe de l'élément, **item-ref** : adresse de l'enregistrement de l'élément.

action **delete-qd** (qd-ident); (* quadtree deletion *) **{ Interface }**
{ cette action supprime complètement un quadtree }
entrée : **qd-ident** : identification du quadtree.

action **select-qd** (qd-ident, region, var liste); **{ Interface }**
*{ cette action est une **recherche géométrique** des éléments couvrant (au moins partiellement) région dans la collection d'éléments / cellule de quadtree qd-ident }*
entrée : **qd-ident** : identification du quadtree, **région** : région de recherche,
sortie : **liste** : liste des éléments trouvés.

action **access-qd** (qd-ident, item-bbox, var liste); **{ Interface }**
*{ cette action est une **recherche** géométrique **exacte** des éléments de rectangle enveloppe donné, dans la collection d'éléments / cellule de quadtree qd-ident }*
entrée : **qd-ident** : identification du quadtree, **item-bbox** : rectangle enveloppe,
sortie :**liste** : liste des éléments trouvés.

3.LES ALGORITHMES DE MISE A JOUR ET DE RECHERCHE

3.1. Algorithmes d'Insertion dans un quadtree, principe

```
type root = record
      rac : ^quad_node;                    { racine du quadtree }
      thresh : quad_treshold;              { paramètres de la partition des données / résolution}
      bbox : tyrect;      { X1,Y1, X2,Y2 }
      Ident : integer;    { identification du quadtree }
   end;

action create-qd ( qd-ident, qd-bbox, qd-thresh );                    { Interface}
  { cette action créé le quadtree d'une collection d'éléments / cellule et définit sa résolution }
   entrée : qd-Ident : identification du quadtree, qd-bbox : rectangle enveloppe de collection
      d'éléments, qd-thresh: seuils de partage.
begin
  root := qd-root ( qd-ident ); if root<> nil then erreur ('create-qd', message)
  else begin allocation ( root, qd-ident, qd-bbox, qd-thresh); mettre-dans-index ( qd-ident, root); end;
end;

action add-qd ( qd-ident, item-bbox, item-ref);                       { Interface }
  { cette action place un élément dans le quadtree d'une collection d'éléménts / cellule }
   entrée : qd-Ident : identification du quadtree, Item-bbox : rectangle enveloppe de l'élément,
      Item-ref : adresse de l'enregistrement de l'élément.
begin
  root := qd-root ( qd-ident );
  if root<> nil then put-qd ( root, item-bbox, item-ref) else erreur ( 'add-qd', message );
end;

action put-qd ( root; item-bbox; item-ref );
  { cette action place un élément dans le quadtree d'une collection d'éléménts / cellule }
   entrée, sortie : root : type root  { nœud racine & rectangle enveloppe & identification du quadtree }
  entrée : item-bbox : rectangle enveloppe de l'élément, item-ref : référence / adresse réelle de l'élément
begin
   if root^.rac = nil then
      creer-nœud ( root^.rac, qtree, nil);   { nil : pas de père }
    if not inclus ( item-bbox, root^.bbox ) then agrandir-qd (root^.rac, root^.bbox, item-bbox);
   add-Item (nil, root^.rac, root^.bbox, item-bbox, item-ref, root^.thresh );
end;

action add-Item ( father, var racine, rect, item-bbox, item-ref,thresh)
{ cette action place un élément dans le sous-quadtree d'une collection d'éléménts / cellule }
   entrée : racine : pointeur racine du quadtree, father : pointeur père, rect: rectangle associé au nœud,
     item-bbox : rectangle enveloppe, item-ref : référence de l'élément, thresh : résolution du quadtree
 begin
   if racine = nil then creer-nœud ( racine, qtree, father);
   if not minimum ( rect, thesh) then begin
      partager-rectangle ( rect, rect1, rect2, rect3, rect4);
      if inclus ( item-bbox, rect1) then add-item (racine, racine^.ul, rect1, item-bbox, item-sref, thresh) else
      if inclus ( item-bbox, rect2) then add-item (racine, racine^.ul, rect2, item-bbox, item-sref, thresh) else
      if inclus ( item-bbox, rect3) then add-item (racine, racine^.ul, rect3, item-bbox, item-sref, thresh) else
      if inclus ( item-bbox, rect4) then add-item (racine, racine^.ul, rect4, item-bbox, item-sref, thresh) else
      add-nœud-qd ( racine, rect, item-bbox, item-sref,thresh); end
  else add-nœud-qd ( racine, rect, item-bbox, item-sref,thresh);
 end;
```

```
action agrandir-qd (var rac, var rect, item-bbox);
   { cette action agrandit un quadtree au-dessus de sa racine lorsque l'élément à insérer est en dehors du
    rectangle enveloppe de la collection déja traitée }
    entrée, sortie : rac : racine du quadtree, item-bbox : rectangle enveloppe du quadtree
    entrée : item-bbox : rectangle enveloppe de l'élément.
begin
    agrandir-rectangle ( rect, rect1, rect2, rect3, rec4);
    ancrac:=rac; créer-nœud ( rac,qtree,nil); anc-rac^.father := RAC;
    if inclus-région ( item-bbox, rect1) then  begin rac^.ul := ancrac; rect <-rect1 end else
    if inclus-région ( item-bbox, rect2) then  begin rac^.ur := ancrac; rect <-rect2 end else
    if inclus-région ( item-bbox, rect3) then  begin rac^.ll := ancrac; rect <-rect3 end else
    if inclus-région ( item-bbox, rect4) then  begin rac^.lr := ancrac; rect <-rect4 end else

    if inclus-zone1 ( item-bbox, rect1) then
       begin rac^.ul := ancrac; agrandir-qd ( rac, rect1) end else
    if inclus-zone2 ( item-bbox, rect2) then
       begin rac^.ur:= ancrac; agrandir-qd ( rac, rect2) end else
    if inclus-zone3 ( item-bbox, rect3) then
       begin rac^.ll:= ancrac; agrandir-qd ( rac, rect3) end else
    if inclus-zone4 ( item-bbox, rect4) then
       begin rac^.lr:= ancrac; agrandir-qd ( rac, rect4) end;
end;

action add-nœud-qd ( rac, rect, item-bbox, item-sref,thresh);
   { cette action ajoute l'élément dans un nœud quaternaire }
 begin
  projeter (rect,seg1, seg2);
   if vertical ( item-bbox ) then
       add-xbtree ( rac, rac^.x-median,seg1, item-bbox, item-sref,thresh)
  else add-ybtree ( rac, rac^.x-median, seg2, item-bbox, item-sref, thresh);
 end;

action add-xbtree ( father, rac, seg, item-bbox, item-sref, thresh);
   { cette action met l'élément dans l'arbre binaire adaptatif associé à la médiane X-médian }
begin
  if rac = nil then creer-nœud ( rac, btree, father);
  if not minimum_ ( seg, thresh) then
     partager -segment (seg, seg-left, seg-right);
    if inclus_ ( item-bbox^.projX, seg-left) then
      add-xbtree ( rac, rac^.left, seg-left, item-bbox, item-sref, thresh)
    else if inclus ( item-bbox^.projY, seg-right) then
      add-xbtree ( rac, rac^.left, seg-left, item-bbox, item-sref, thresh)
    else add-nœud-xbtree ( rac, item-bbox, item-sref)
  else add-nœud-xbtree ( rac, item-bbox, item-sref)
end;
```

3.2. Algorithmes de suppression d'un élément (/ item) dans un quadtree, principe

```
action supr-qd ( qd-ident, item-bbox, item-ref);                    { Interface }
  { cette action enlève un élément de la collection }
    entrée : qd-ident, item-bbox, item-ref
begin
  root := qd-root (qd-ident);
 if root<> nil then if root^.rac <> nil then remove-qd ( root, root^.rac,root^.bbox, item-bbox, item-ref)
 else erreur ('supr-qd',message);
end;

action remove-qd ( root, racine, rect, item-bbox, item-ref);
   { cette action enlève un élément d'un quadtree / sous-quadtree }
    entreé, sortie, : root : quadtree root & bounding-box & treshold & ident.
     entrée : ITEM_BBOX : item's bounding-box, ITEM_REF : item's reference *)
    begin
       si racine > nil then begin
       if not minimum (rect, root^.thresh) then begin
              partager-rectangle ( rect, rect1, rect2, rect3, rect4);
          if inclus ( item-bbox, rect1) then
              remove-qd ( root, racine^.ul, rect1, item-bbox, item-sref)
           else if inclus ( item-bbox, rect2) then
              remove-qd ( root, racine^.ur, rect2, item-bbox, item-ref)
           else if inclus ( item-bbox, rect3) then
              remove-qd ( root, racine^.ll, rect3, item-bbox, item-sref)
           else if inclus ( item-bbox, rect4) then
              remove-qd ( root, racine^.lr, rect4, item-bbox, item-sref)
           else remove-nœud-qd ( root, racine, rect4, item-bbox, item-sref)
       end;
       else remove-nœud-qd ( root, racine, rect4, item-bbox, item-sref)
    end;

action remove-nœud-qd ( root, racine, rect, item-bbox, item-sref)
  { cette action enlève lélément item de l'un des 2 arbres binaires associés et supprime éventuellement
aussi le nœud s'il est vide et récursivement son père s'il est vide.}
```

3.3. Suppression de quadtree

```
action delete-qd ( qd-ident);                                       { Interface }
   { suppression de quadtree }
begin
   root := qd-root ( qd-ident );
   if root <> nil then begin
      supri-qd ( root^.rac); root^.rac := nil;          { suppression duquadtree }
      supr-index (qd-ident);                            { suppression du quadtree dans l'index }
   end;
end;

action supri-qd ( racine );
begin
  if racine <> nil then begin
       supri-qd ( racine^.ul );   supri-qd ( racine^.ur );
       supri-qd (  racine^.ll );    supri-qd ( racine^.lr );
       supri-bin ( racine^.x-median ); supri-bin ( racine^.y-median );
       dispose-  ( racine, qtree);
end; end;
action supri-bin ( rac );
begin
```

```
  if rac <> nil then begin
      supri-bin (rac ^.left); supri-bin (rac^.right);
      supri-list (rac^.first);
       dispose- ( rac, btree);
  end;
end;

action supri-list ( r );
begin
   while r <> nil do begin
        suiv-r := r^.next;     dispose_ ( r );     r : = suiv-r;
   end;
end;
```

3.4. Algorithmes de recherche geometrique

```
action select-qd ( qd-ident, region, var liste)                    { Interface }
   { cette action est une recherche géométrique des éléments couvrant (au moins partiellement)
   région dans la collection d'éléments / cellule de quadtree qd-ident }
   entrée : qd-ident : identification du quadtree, région : région de recherche,
   sortie : liste : liste des éléments trouvés.
begin
   root := qd-root ( qd-ident ); if root<> nil then erreur ('select-qd', message);
   liste := nil; window ( root^.rac, root^.bbox, region, liste);
end;

action window  ( rac, rect, region, var liste);
begin
 if rac <> nil then                                  { visite au sous-arbre (rac) }
  if inclus ( rect, region ) then put-out-qd ( rac, liste) else
   if overlap ( rect, region) then begin
      axes-medians ( rect, xmedian, ymedian);
      vis-x-tree ( rac^.x-median, xmedian, region, liste);
      vis-y-tree ( rac^.y-median, ymedian, region, liste);
           partager-rectangle ( rect, rect1, rect2, rect3, rect4);
           window ( rac^.ul, rect1, region, liste );
           window ( rac^.ur, rect2, region, liste );
           window ( rac^.ll, rect3, region, liste );
           window ( rac^.lr, rect4, region, liste );
   end;
end;

action vis-x-tree ( rac, seg, region, var liste);
  { visite à l'arbre binaire associé à une mediane de partage d'un quadrant }
begin
  if rac <> nil then
   if inclus- ( seg, region) then put-out-btree ( rac, liste) else begin
      visit-nœud-xtree ( rac, region, liste);
      share- ( seg, seg-left, seg-right);
      if overlap ( se-left, region) then vis-x-tree ( rac^.left, seg-left, region, liste);
      if overlap ( seg-right, region) then vis-x-tree ( rac^.right, seg-right, region, liste);
   end;
end;
```

4. QUADTREE ET HIERARCHIE DE RECTANGLES (ou d'objets géometriques)

```
type matrix_topo = begin
            A, B, C, D, E, F : real;          { transformation générale du plan }
end;

procedure h-select ( symbol : pstruct; region : tyrect; sel : integer);
      { selection / autre traitement de tous les objets inclus hiérarchiquement dans region ;
      sel : traitement selectionné; symbol : symbole de plus haut niveau }
begin
      tronquer ( region ); select-qd ( symbol^.qd, region, liste );
      premier (liste, element-de-symbol);
      while ( element-de-symbol<> nil) do begin
            case typ ( element-de-symbol ) of
                  primaire : display ( symbol, element-de-symbol, matrice-identite, sel);
                  construit : begin
                        t-a := element-de-symbol^.topo^;        { transformation du modèle psref }
                        h-window ( symbol, element-de-symbol^.psref,t-a, region);
                  end;
            avancer (liste, element-de-symbol);
      end;
end;


procedure h-window ( top-symbol, symbol: pstruct; t-a : matrix_topo; region : tyrect);
      { t-a : transformation absolue de symbol dans top-symbol }
begin
        save-region := region;
      reverse-t-a := reverse (t-a);          modifi-rectangle ( region, reverse-t-a);
      tronquer ( region ); select-qd ( symbol^.qd, region, liste);
      premier ( liste, element-de-symbol);
      while ( liste <> nil) do begin
            case typ (element-de-symbol) of
                  primaire : display ( symbol, element-de-symbol, t-a, sel );
                  construit : begin
                        region := save-region;                  { repèrage absolu }
                        t-a:= produit ( element-de-symbol^.topo^, t-a );
                        h-window (top-symbol, element-de-symbol^.psref, t-a, region);
                  end;
            avancer ( liste, element-de-symbol);
      end;
end;
```

Présentation générale des algorithmes, remarque générale

*Ces algorithmes, écrits dans un langage algorithmique de type pascal, ont tous subi dans leur implémentation de nombreuses **optimisations** qui ne sont pas indiquées ici. L'analyse algorithmique est donnée ici dans sa forme originale.*

ANNEXE 2

L'ANALYSE HIERARCHIQUE DES VLSI

ALGORITHMES

1. STRUCTURE DE DONNEES POUR L'ANALYSE HIERARCHIQUE DE LA DESCRIPTION GEOMETRIQUE D'UN CIRCUIT

```
type biblio = record
            nom: string8;              { nom }
            model : table_model;       { table des modèles , accès par hash-coding }
            inter : table_interaction;  { table des interactions entre modèles}
            elem : set_pelement;       { adresses des éléments de tous les modèles }
end;
type matrix-topo = record                                   { transformation générale}
            A, B, C, D, E, F : real;
end;
type raccess = record                                       { accès aux MOS }
            source1, source2, gate : ^element;
end;
type elem = record
        nom, dad : integer;            { index de l'élément et de son père dans le tableau elem }
        color: layer;                  { layer1..layer2}
        father : ^struct;              { père hiérarchique }
        dimens : ^elect_param;         { paramètres electriques }
        bbox : tyrect;                 { X1,Y1,X2,Y2}
        error, halo: boolean;          { ensemble des erreurs } { vrai si elem couvre le halo associé}
        typ : fi_type;                 { type de l'élément }
        case type of
        sref, pin : ( psref: integer; topo : ^matrix_topo);  { référence et transformation de modèle }
        nmos, pmos : (access : ^raccess);                    { accès du transistor }
        port, contact, wire : (tens, typort : byte);         { vdd / vss, i/o/a}
end;
type interaction = record
        A, B : integer;                { références des 2 modèles }
        T_bRa,                         { position relative : modèle B / modèle A }
        T_aRb : matrix_topo;           { position relative : modèle A / modèle B}
        next : ^interaction;
end;
type struct = record
        nom : integer; unité: real;                          { nom et unité du modèle }
        bord : pelement; bbox : tyrect;                      { paramètres géométriques}
        fi : fi_type;                                        { type du modèle}
        ensemble-éléments: set-of-elements
        qd, qd-dual, qd-err : integer;                       { quadtrees associés }
        rewrit, extract, checked, compression: boolean;  { flags }
        l-rewrit, l-union, l-eqivport, l-violation: ^pairwise;{ paires d'équivalences, violations ...}
        l-err : ^object;                                     { DRC / ERC violations }
        l-subsymbols : ^l_pstruct;                           { modèles utilisés}
        next : ^struct;                                      { hash-coding : liste des collisions }
        graphe : ^graphe;          { listes nœuds externes & des nœuds internes & des transistors }
end;
type set-of-element = intervalle;   { intervalle dans le tableau des éléments }
```

Définition de la pile des ancêtres

```
type ancetre = enregistrement   {liste des ancêtres d'un objet empilés de bas en haut}
      sref : ^struct; { sous-cellule ancêtre}
      r_topo,         { transformation de la sous-cellule par rapport à son père / structure appelante}
      a_topo : matrix_topo;          { transformation absolue de la sous-cellule}
      prev : ^ancestre;                    { ancêtre précédent }
    fin;
type pancestor= ^ancêtre;   type ppile = ^ancêtre; type pélément = ^element; type pstruct = ^struct;
type matrix_topo = transformation;  type tyrect = record x1, y1, x2, y2 end;
```

2. ANALYSE DE LA HIERARCHIE

2.1. Extraction incrementale des équivalences et noeuds electriques d'une cellule, principe de l'algorithme

```
procedure construire-noeuds (symbol : pstruct);
début
   élément-de-symbol := premier-élément (symbol);
   tant que élément-de-symbol <> nil faire début
      cas typ (élément-de-symbol) est
         primaire : début
            select-qd (symbol, élément^.bbox, liste);
            tant que liste <> nil faire début
               cas typ (élément) est
                  primaire : si non vérifié-entre ( élémenta , liste^.elem )
                     alors début
                     basic-équivalence ( élémenta, liste^.elem, union, erreur);
                     si union alors union-find ( élémenta, liste^.elem, union, find);   { -> find }
                     si erreur ou find alors
                        reporter-dans(symbol,élémenta,élémentb,erreur,find);
                     fin;
                  construit :;
               fin;
               avancer (liste); fin;
         construit : réaliser-équivalences-pins   { equiv. port -> équiv. pin }
      fin;
   élément-de-symbol := élément-suivant (symbol, élément-de-symbol);
   comprimer-les-composantes-connexes;  { compression des arbres de fischer-galler }
   faire-remonter-les-ports-&-mettre-dans-listes-les-racines-des-c-connexes;
   { les ports sont remontés comme racines des arbres de F.G. et représentent les nœuds externes }
   { nœuds externes et nœud internes - i.e. leur racine- sont mis dans 2 listes de nœuds de symbol }
fin;
```

2.2. Analyse incrementale, principe de l'algorithme

```
procedure analyser (nom-symbol : entier; symbol : pstruct; sel : integer);
   { analyse incrémentale de symbole, sel : sélection du traitement }
début
   lire-symbol (nom-symbol, symbol);   { base de données -> bibliothéque , rien si déjà lu }
   faire-le-traitement (symbol);
   reporter-les-résultats-et-erreurs (symbol, nom-symbol);   { bibliothéque -> base de données }
fin;
```

2.3. Analyse hiérarchique, principe

```
procedure analyser-hiérarchie (symbol : pstruct; sel : integer);
{ analyse hiérarchique de symbole, sel : sélection du traitement }
début
   lire-symbol (nom-symbol, symbol);
      { base de données -> bibliothéque }
   si typ (symbol) = construit alors si non vérifié (symbol, sel ) alors
      pour chaque sous-symbole faire analyser-hiérarchie (sous-symbol,sel)
   analyser (nom-symbol, symbol, sel);
   marquer-symbole-analysé (symbol, sel, date);
fin;
```

2.4. Analyse de la connectivité électrique localement à un symbole

```
action  union-find (élément1, élément2 : pélément; union : boolean; var find : boolean); {chap.2}
début
  x := élément1^.nom;   y:= élément2^.nom;
  I  := X; tant que b.elem [I]^.dad > 0 faire  I := b.elem [I].dad;
  J := Y; tant que b.elem [j]^.dad > 0 faire  j:= b.elem [j].dad;
                                                    { compression des chemins }
  tant que  b.elem  [X]^.dad > 0 faire  début  T := X; X := b.elem [X]^.dad; b.elem [T]^.dad :=  I   fin;
  tant que  b.elem  [Y]^.dad > 0 faire  début  T := Y; Y := b.elem [Y]^.dad; b.elem [T]^.dad := J   fin;
  si  union  et  (I <> J) alors
      si  b.elem [J]^.dad < b.elem [I]^.dad alors              { balance }
            début b.elem [J]^.dad :=  b.elem [J]^.dad + b.elem [I].dad;  dad [I] := J fin
      sinon   début b.elem  [I]^.dad :=  b.elem [J]^.dad + b.elem  [I].dad; dad [J] := I  fin;
  find := ( I <> J );
fin;
```

2.5. Analyse hierarchique de la connectivite electrique, recherche d'un équivalent électrique global à travers la hiérarchie de conception

```
fonction représentant-nœud ( top-symbol : pstruct; objet: pelement;
      var  équivalent : pelement; var pile-rem : pancêtre) : name_elem;
  entrée :  top-symbol : symbole de plus haut niveau;
          objet, pile-rem <- objet dont on cherche l'équivalent électrique; pile-rem est initialisé par la
          pile des ancêtres de objet.
  sortie : équivalent, pile-rem -> l'équivalent électrique de plus haut niveau dans la hiérarchie, i.e. un
          élément et sa pile d'ancêtres.

début
  cas typ  (objet)  est
    primaire: begin
    équivalent :=équivalent-électrique-local  (objet);  { racine de l' arbre de fischer-galler de objet}
    { recherche de l'équivalent électrique global }
    si (objet^.father<> top-symbol) and (équivalent^.typ = port) alors      {sinon return}
    début          { transformation port -> pin }
      bbox:= objet^.bbox;                                  { bbox : rectangle enveloppe de objet }
      modifi-rectangle (bbox, pile-rem^.r_topo); pile-rem:= pile-rem^.prev;  troncature (bbox);
      access-qd (pile-rem^sref^.qd, bbox, liste); { ->la liste des objets de sref de rect. env. bbox }
      chercher-pin (liste, pin);
      si pin <> nil  alors
          représentant-nœud:= représentant-nœud (top-symbol , pin,  équivalent, pile-rem );
    fin;
  fin;
fin;
```

2.6. Vérification incrementale d'une cellule, principe de l'algorithme de DRC

```
procedure vérifier-symbol (symbol : pstruct);
  {analyse/ verification du dernier niveau de symbol}
début
   initialier-pile ( pile-a, symbol); pile-b := pile-a; précédent-contexte <-- (pile-a, pile-b);
   élément-de-symbol := premier-élément (symbol);
 tant que élément-de-symbol <> nil faire début
      empiler (pile-a, élément-de-symbol); { empiler est effectif si l'élément est de type construit }
      cas typ ( élément-de-symbol) est
         primaire: début {recherche des interactions des éléments primaires avec les autres éléments}
             select-qd (symbol^.qd, région (element-de-symbol) , liste ); {elem.prim. & constr.)
             tant que liste <> nil faire début
                 empiler ( pile-b,liste^. elem ); { empiler est effectif si l'élément est de type construit }
                 cas typ (liste^.elem) est
                 primaire : si non ok-géom (symbol, élément-de-symbol, liste^.elem,
                                            région (element-de-symbol),  liste^.elem^.bbox) alors
                       si non vérifié-entre (symbol, élément-de-symbol, liste^.elem,
                       pile-a, pile-b, T_aRb, précédent-contexte )   { -> précédent-contexte }
                       alors basic-drc (symbol, élément-de-symbol,liste^.elem,pile-a,pile-b);
                 construit : si non vérifié-entre (symbol, élément-de-symbol, liste^.elem,
                       pile-a, pile-b, T_aRb, précédent-contexte )   { -> précédent-contexte }
                       alors vérifier-interaction (symbol, élément-de-symbol, liste^.elem,
                       pile-a, pile-b, T_aRb);
                 fin;
                 dépiler (pile-b, liste^.elem);
                 avancer (liste);
             fin;
         fin
         construit : début {recherche des interactions entre éléments construits cherchés dans qdual}
             select-qd (symbol^.qdual, région (élément-de-symbol) , liste ); {elem. construits)
             tant que liste <> nil faire début
                 empiler ( pile-b,liste^. elem );
                 si non vérifié-entre (symbol, liste^.elem, élément-de-symbol,
                       pile-b, pile-a, T_bRa , contexte-appel, précédent-contexte )
                 alors vérifier-interaction (symbol, liste^.elem, élément-de-symbol,
                       pile-b, pile-a, T_bRa , précédent-contexte );   { -> précédent-contexte }
                 dépiler (pile-b, liste^.elem);
                 avancer (liste);
             fin;
         fin;
      fin;
      dépiler (pile-a, élément-de-symbol);  { dépiler est effectif si l'élément est de type construit }
      élément-de-symbol :=element-suivant (symbol, élément-de-symboi);
 fin;
   marquer-symboi-vérifié (symbol);
   libérer-pile (pile-a, symbol);
fin;
```

```
procedure vérifier-interaction ( symbol: pstruct; élémenta,élémentb : pelement;
      pile-a, pile-b : pancestor; T_aRb : matrix_topo);
  {vérification des interactions entre symboles et sous-symboles}
début
   ebbox-a := région (élémenta); modifi-rectangle (ebbox-a, T_aRb); troncature (ebbox-a);
   symbol-de-élémentb := pile-b^.sref;
   select-qd ( symbol-de-élémentb, ebboxa, liste); { en sortie : la liste des éléments voisins de a}
  précédent-contexte <-- (pile-a, pile-b);
  tant que liste <> nil faire début
    empiler (pile-b, liste^.elem);          { empiler est effectif si l'élément est de type construit }
      cas typ (liste^.elem) est
      primaire :
         cas typ (élémenta) est
           primaire: si non ok-géom(symbol, élémenta,liste^.elem,ebboxa,liste^.elem^.bbox)
                 alors basic-drc (symbol, élémenta, liste^.elem, pile-a, pile-b)
            construit: si non vérifié-entre ( symbol, liste^.elem, elementa,
                      pile-b, pile-a, T_bRa, précédent-contexte ).   { -> précédent-contexte }
                 alors vérifier-interaction (symbol, liste^.elem,élémenta,
                      pile-b, pile-a, T_bRa);
         fin;
      construit : si non vérifié-entre (symbol, élémenta, liste^.elem,
                   pile-a, pile-b, T_aRb, précédent-contexte)        { -> précédent-contexte }
                 alors vérifier-interaction (symbol, liste^.elem, élémenta,
                   pile-a, pile-b, T_aRb)
      fin;
      dépiler (pile-b, liste^.elem);      { dépiler est effectif si l'élément est de type construit }
      avancer (liste)
  fin;
fin;

fonction vérifié-entre (symbol : pstruct; elementa, elementb : pelement;
   pile-a, pile-b : pancestre; var T_aRb : matrix_topo;
    var précédent-contexte : pcontexte ) : booleen;
   { cette fonction est vraie si elementa et elementb ont déjà été vérifiés dans cette position
   relative; cette fonction calcule et maintient la transformation T_aRb qui définit la
   position relative de A dans le repère lié à B; l'appel de cette fonction assure ainsi le marquage
   de la vérification des 2 éléments construits; pour les autres types d'interactions certaines
   redondances de vérification sont aussi évitées.
   Cette fonction vérifie que le contexte d'appel (pile-a, pile-b) a changé afin d'optimiser le nombre
   de transformations effectivement calculées; ainsi elle met à jour précédent contexte }

fonction ok-géom ( symbol : pstruct; elementa, elementb : pelement; bboxa, bboxb : tyrect)
   { cette fonction est vraie si les gardes d'espacement entre les rectangles bboxa et bboxb sont
   respectées, instances des rectangles enveloppes de elementa et elementb calculés dans un même
   repère, indépendemment de la connectique (niveau 1 d'élimination ) }
```

3. SHELL D'UNE CELLULE

```
procedure shell ( symbol : pstruct);
  { cette action sélectionne et affiche la partie externe d'une cellule composée de l'ensemble de ses
  nœuds externes }
début
  initialier-pile (pile, symbol);
  élément-de-symbol := premier-élément (symbol);
  tant que élément-de-symbol <> nil faire début
      empiler (pile, élément-de-symbol); { empiler est effectif si l'élément est de type construit }
      cas typ ( élément-de-symbol) est
         primaire : si nœud-est-externe (symbol, élément-de-symbol, pile)
                        alors display-élément (symbol, élément-de-symbol, pile);
         construit : shell-sous-symbol (symbol, élément-de-symbol^.sref, pile);
      fin;
      dépiler (pile, élément-de-symbol);
      élément-de-symbol := élément-suivant (symbol, élément-de-symbol );
  fin;
  libérer-pile (pile, symbol);
fin;

procedure shell-sous-symbol ( top-symbol, symbol : pstruct; pile : pancestor);
  { sélection des parties externes de top-symbol cachées dans ses sous-cellules }
début
  élément-de-symbol := premier-élément (symbol);
  tant que élément-de-symbol <> nil faire début
      empiler (pile, élément-de-symbol); { empiler est effectif si l'élément est de type construit }
      cas typ ( élément-de-symbol) est
         primaire : si nœud-est-externe (top-symbol, élément-de-symbol, pile)
                        alors display-élément (top-symbol, élément-de-symbol, pile);
         construit : shell-sous-symbol (top-symbol, élément-de-symbol^.sref, pile);
      fin;
      dépiler (pile, élément-de-symbol);
      élément-de-symbol := élément-suivant (symbol, élément-de-symbol );
  fin;

fonction nœud-est-externe ( symbol : pstruct; élément : pelement; pile: pancestor) : booleen;
  { cette fonction est vraie si le nœud électrique de élément est externe dans symbol }
début
  nœud-de-élément
                :=représentant-nœud (symbol, élément, équivalent-de-élément,pile-équivalent);
  nœud-est-externe := (équivalent-de-élément^.typ = port) et (pile-équivalent^.sref= symbol);
fin;
```

4. ALGORITHMES DE MISE A PLAT DU GHAPHE ELECTRIQUE

Le graphe electrique de symbol est mis à plat à partir de sa représentation compacte et hiérarchique;

entrée : représentation compacte et hiérarchique du graphe électrique
sortie : grpahe électrique complètement à plat.

principe

```
procedure graph-symbol ( symbol : pstruct);
  { cette procédure met à plat la représentation compacte et hiérarchique du réseau des noeuds
  électriques de symbol }
début
  initialier-pile (pile, symbol);
  élément-construit-de-symbol := premier-élément-construit (symbol);
  tant que élément-construit-de-symbol <> nil faire début
       empiler (pile, élément-de-symbol);  { empiler est effectif si l'élément est de type construit }
        graph-sous-symbol (top-symbol, élément-construit-de-symbol.^sref, pile);
       dépiler (pile, élément-de-symbol);
       élément-construit-de-symbol := construit-suivant (symbol, élément--construit-de-symbol );
  fin;
  nœud-de-symbol := premier-nœud (symbol);        { parcourir listes des nœuds internes et externes }
  tant que nœud-de-symbol <> nil faire début
       créer-nœud (nœud-de-symbol, pile);     { créer une instance du nœud électrique }
       nœud-de-symbol := nœud-suivant ( symbol, nœud-de-symbol );
  fin
  libérer-pile (pile, symbol);
fin;

procedure graph-sous-symbol ( top-symbol, symbol: pstruct; pile:pancestor);
       { mise à plat d'un sous-symbole : tous les nœuds internes sont instantiés }
début
  élément-construit-de-symbol := premier-élément-construit (symbol);
  tant que élément-construit-de-symbol <> nil faire début
       empiler (pile, élément-de-symbol);  { empiler est effectif si l'élément est de type construit }
        graph-sous-symbol (top-symbol, élément-construit-de-symbol^.sref, pile);
       dépiler (pile, élément-de-symbol);
       élément-construit-de-symbol := construit-suivant (symbol, élément--construit-de-symbol );
  fin;
  nœud-interne-de-symbol := premier-nœud-interne (symbol);  { listedes nœuds internes }
  tant que nœud-interne-de-symbol <> nil faire début
       créer-nœud (nœud-de-symbol, pile);          { créer une instance du nœud électrique }
       nœud-interne-de-symbol := nœud-interne-suivant ( symbol, nœud-interne-de-symbol );
  fin
fin;
```

5. CONSTRUCTION DYNAMIQUE DU HALO D'UNE CELLULE
(D.R.C. de hiérarchies cohérentes)

```
procedure Halo ( symbol : pstruct);
  { cette action construit de manière dynamique le Halo d'une cellule }
début
  initialier-pile (pile, symbol);
  élément-de-symbol := premier-élément (symbol);
  tant que élément-de-symbol <> nil faire début
      empiler (pile, élément-de-symbol); { empiler est effectif si l'élément est de type construit }
      cas typ ( élément-de-symbol) est
         primaire : si élément-couvre-halo (symbol, élément-de-symbol, pile)
                       alors mettre-élément -dans-halo (symbol, élément-de-symbol, pile);
         construit : si élément-couvre-halo (symbol, élément-de-symbol, pile)
                       alors halo-sous-symbol (symbol, élément-de-symbol^.sref, pile);
      fin;
      dépiler (pile, élément-de-symbol);
      élément-de-symbol := élément-suivant (symbol, élément-de-symbol );
  fin;
  libérer-pile (pile, symbol);
fin;

procedure halo-sous-symbol ( top-symbol, symbol : pstruct; pile : pancestor);
  { sélection des parties externes de top-symbol cachées dans ses sous-cellules }
début
  élément-de-symbol := premier-élément (symbol);
  tant que élément-de-symbol <> nil faire début
      empiler (pile, élément-de-symbol); { empiler est effectif si l'élément est de type construit }
      si élément-appartient-à-halo ( symbol, élément-de-symbol) alors
         cas typ ( élément-de-symbol) est
            primaire : si élément-couvre- halo (top-symbol, élément-de-symbol, pile)
                  alors mettre-élément -dans-halo (top-symbol, élément-de-symbol, pile);
            construit : si élément-couvre-halo (top-symbol, élément-de-symbol, pile)
                  alors halo-sous-symbol (top-symbol, élément-de-symbol^.sref, pile);
         fin;
      fin;
      dépiler (pile, élément-de-symbol);
      élément-de-symbol := élément-suivant (symbol, élément-de-symbol );
  fin;

fonction élément-appartient-à-halo ( symbol, élément-de-symbol) : booleen;
  { cette fonction est vraie si élément couvre le halo du symbol qui l'a inclus}
début
  ebbox-de-élément := région ( élément ); { la région d'interaction de élément est calculée selon la garde
                                            maximale de l'élément de couleur élément^.color }
  si intersection (ebbox-de-élément, top-symbol^.bord)
  alors début
            élément-appartient-à-halo := vrai;
            élément^.halo := vrai;      { pour marquage dans la cellule et la base de données , cf. note 2}
        fin
  sinon début
            élément-appartient-à-halo := faux
            élément^.halo := faux;      { pour marquage dans la cellule et la base de données , cf. note 2}
  fin;
fin;

{ pour marquage dans la cellule et la base de données , cf. note 2}
```

fonction **élément-couvre-halo (top- symbol** : pstruct; **élément** : pelement; pile: pancestor) :
booleen; (1)
{ cette fonction est vraie si ***élément*** *de contexte hiérarchique* ***pile*** *couvre le* ***halo*** *de* ***top-symbol*** *}*
début
ebbox-de-élément := **région** (élément); *{ la région d'interaction de élément est calculée selon la* ***garde maximale*** *de l'****élément*** *de couleur élément^.color }*
modifi-rectangle (ebbox -de-élément , pile^.a-topo);
si **intersection** (ebbox-de-élément, top-symbol^.**bord**) alors **élément-couvre-halo** := vrai
sinon **élément-couvre-halo** := faux
fin;

({ pour marquage dans la cellule et la base de données , cf. note 2}

(1) - note 1 : Il y a **intersection** *de l'instance de* ebbox-de-element *avec le bord de* **top-symbol** *si et seulement si il y a* **intersection** *de cette région de recherche avec l'un des éléments (segments) qui forment le* **bord** de **top-symbol.**

- note 2 : Un ***quad-tree*** *spécifique associé au* ***bord*** *de* ***symbol*** *peut être éventuellement utilisé pour filtrer les éléments primaires ou construits qui appartiennent au halo de top-symbol. Cette information est alors éventuellement conservée avec top-symbol dans la base de donnée, afin de ne pas avoir à être recalculée. Ce qui optimise alors la construction dynamique du halo de top-symbol.*

*- **note 3 :** Cette technique de reconstruction des halos est un simple litrage hiérarchique pour lequel il est nécessaire de recalculer les équivalences électriques. Une information incomplète de connectivité électrique provoque alors la détection de fausses erreurs. Afin d'éviter cet inconvénient , il est necessaire de propager également les équivalences électriques des éléments terminaux des sous-cellules imbriquées, et d'éviter de revérifier des éléments primaires inclus dans une même sous-cellule.*

*- **note 4** : cette technique beaucoup est plus complexe et moins sûre que la technique de construction incrémentale des cellules halos (cf. note 3). La sauvegarde des halos rend certainement les algorithmes plus performants avec un certain coût à payer en mémoire secondaire.*

ANNEXE 3

QUADTREES ET BASES DE DONNEES GEOGRAPHIQUES

QUADTREES ET BASES DE DONNEES GEOGRAPHIQUES

La recherche **géométrique des sous-régions couvrant (au moins partiellement) un domaine donné** est aussi l'un des problèmes des applications des bases de données géographiques. Une structure basée sur l'allocation de clefs analytiques calculées par une décomposition quaternaire d'une région est proposée par [ABE 83], [ABE 84] *(figure 1)*. Ces **clefs analytiques** données aux différents enregistrements permettent leur gestion par le **système général de base de données** et la **recherche géométrique des enregistrements à extraire de la base de donnée et à examiner** [ABE 83]. Le problème géométrique résolu par une telle méthode paraît semblable aux problèmes géométriques des conceptions VLSI. Cependant le contexte du problème est très différent puisque les **bases de données géographiques sont des bases qui évoluent lentement**. Une très grande région est formée de nombreuses sous-régions comme un puzzle est constitué de nombreuses pièces. Ainsi cette méthode est surtout intéressante pour les cas où l'ensemble des enregistrements est suffisamment important pour nécessiter l'usage de mémoires secondaires (disques).

Les clefs numériques assignées à chaque enregistrement ne sont pas nécessairement uniques et dépendent de la position du rectangle / sous-région dans la région. La structure est conceptuellement une liste qui est accessible dans l'ordre séquentiel des clefs et aussi aléatoirement en utilisant la clef comme index. Pour ces différents types d' accès , certaines propriétés sont demandées à la clef.

Le principe de l'algorithme de recherche géométrique - La **décomposition quaternaire de la région englobante est celle des quadtrees adaptatifs de rectangles** et le calcul de la clef de localisation d'un rectangle / sous-région correspond au calcul de la position qu'aurait le rectangle dans ce type de quadtree. Ainsi la même clef de localisation est associée à tous les enregistrements bloqués dans un même nœud quaternaire. Lors d 'une recherche géométrique des sous-régions ou enregistrements couvrant (au moins partiellement) un domaine, toutes les sous-régions appartenant aux nœuds quaternaires dont la région englobante a une intersection non vide avec le domaine, doivent être examinés. Leur examen demande alors leur accès dans la base de données géographique directement au moyen de leur clef de localisation.

La clef de localisation - La **clef de localisation caractérise le nœud quaternaire auquel appartient le rectangle / sous-région**; elle traduit le chemin qu'il faut suivre à partir de la racine de l'arbre quaternaire associé à la région pour atteindre ce nœud c'est-à-dire la **liste ordonnée de ses ancêtres**. Mais la clef de localisation doit posséder aussi certaines proprétés qui permettent sa gestion par le système général de base de données (B-arbres, [COM 79]).

Des clefs de localisation / régionnement par décomposition quaternaire d'une région ont été proposées par un certain nombre d'auteurs. [COO 79] adopte les indices d'une matrice de Morton [MOR 66], [COM 81]; [WEB 79] suggère une forme similaire basée sur le "Sussenguth tree". [KLI 79], [JON 81], [GAR 82] développent aussi une clef dérivée de la liste ordonnée des ancêtres d'un nœud. La plupart des schémas ainsi présentés privilégient l'énumération par niveau. [ABE 83] construit une clef dont l'ensemble des valeurs privilégie les énumérations et traversées en préordre.

Les clefs de localisation / régionnement sont donc définies comme des index numériques pour les quadrants formés par la décomposition quaternaire d'une région. L'ensemble des valeurs obtenues pour une résolution quaternaire de 3 est fournie par le tableau ci-dessous. Si les fils d'un nœud quaternaire sont numérotés 1, 2, 3, 4 par rapport à leur père et si les valeurs manquantes (blanc) sont remplacées par des zéros alors la **clef de régionnement d'un nœud / quadrant est bien la liste ordonnée de ses ancêtres éventuellement complétée par des zéros. L'ensemble des clefs s'exprime ainsi en base 5.**

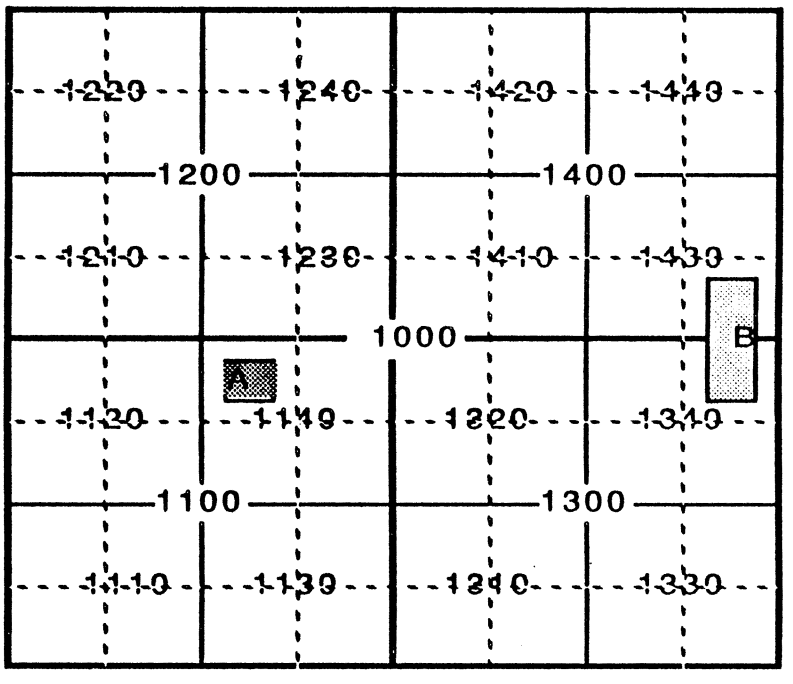


Figure 1 - Trois niveaux de décomposition et leur clef de localisation en base 5. Les clefs analytiques de localisation de sous-régions d'enveloppes rectangulaires A et B sont respectivement 1142 et 1000.

L'ensemble des clefs ainsi calculées pour chaque quadrant a la propriété de privilégier l'énumération séquentielle en préordre des différents nœuds ou sous-quadrants du quadtree c'est-à-dire que les sous-quadrants fils sont examinés juste après leur père. L'enregistrement d'une sous-région **est "insérée"** dans ce **quadtree adaptatif de rectangles sans pointeurs** par le calcul de sa clef analytique de régionnement (et l'insertion dans le B-tree correspondant); sa clef est alors la clef du plus petit quadrant qui l'englobe totalement. Le **quadtree adaptatif de rectangles n'est plus alors qu'un "directory"** pour les algorithmes d'insertion et de **recherche géométrique**.

De toutes les méthodes basées sur des clefs analytiques de régionnement, cette méthode est la seule à traiter le problème assez large de la couverture d'un domaine de recherche par un ensemble de sous-régions au moyen d'un quadtree adaptatif de rectangles. Il est à craindre que le nombre d'éléments examinés par une recherche soit important par rapport au nomdre d'éléments trouvés (§4.). Un **"raffinement" de la clef** de régionnement par **l'implémentation de régionnements binaires** dans les divers sous-quadrants améliorerait certainement aussi les recherches de sous-régions.

ANNEXE 4

LA METHODOLOGIE STICK

1. LE SYMBOLISME STICK

Le terme **"Stick"** est un terme générique donné à un style de description symbolique du layout , produite au moyen un éditeur graphique interactif. Ce symbolisme permet, avec un jeu minimum de règles, de réaliser rapidement le dessin de circuits complexes, tout en gardant une bonne densité d'intégration.

Ce symbolisme permet de manipuler uniquement le squelette des transistors et celui des interconnexions entre ces transistors. Le symbolisme Stick utilisé au *CNET Grenoble par le système de CAO Cassiopée* est généré sur une grille de pas fixe, pour une technologie donnée [LEC 84]. Deux versions ont été développées, l'une pour la technologie NMOS, l'autre pour la technologie CMOS. Ce sytème de conception demande donc de la part des utilisateurs une connaissance des règles technologiques utilisées. Une vérification des règles de connectique est réalisée sur le layout symbolique ainsi édité, ainsi qu'une vérification des règles de dessin. Un simulateur électrique permet de contrôler le comportement électrique lié à cette implantation. Finalement une génération des masques est effectuée très simplement à partir de la description symbolique.

Dans ce symbolisme quatre types de symboles sont utilisés : les transistors, les connexions, les divers types de contacts et les caissons. Dans la technologie CMOS actuellement utilisée, les transistors sont de longueur minimale et de largeur paramétrable. Il est possible de construire des transistors coudés.

2. LES REGLES DE DESSIN DU LAYOUT

2.1. La formulation des règles symboliques

Les règles de dessin ont des valeurs paramétrables selon la technologie utilisée. Elles sont numérotée et un fichier technlogique produit la valeur associée à chaque règle. Le principe de la classification des règles est le suivant :

1. Règles entre connexions identiques
2. Règles entre connexions différentes
3. Règles entre connexions et croisement de connexions
4. Règles entre contacts équipotentiels
5. Règles entre contacts non équipotentiels.
6. Règles entre contact et connexion
7. Règles entre contact et croisement de connexion
8. Règles entre transistors de même type
9. Règles entre transistors de type différents
10. Règles entre transistors et connexions
11. Règles entre transistor et contacts
12. Règles concernant les caissons
13. Autres règles

Remarque 1 : Le **transistor est un symbole orienté**, c'est-à-dire qu'il faut distinguer les **règles dans le sens du transistor**, et celles dans **le sens perpendiculaire au transistor**.

Remarque 2 : Il est nécessaire d'ajouter des **règles entre symbole et caisson**.

Ces règles ont la particularité de pouvoir être **différentes** suivant que l'on se trouve à l'**intérieur** et à **l'extérieur** du caisson.

Ainsi, par exemple les règles entre **transistors de même type** - **.8** - génèrent les 6 cas suivants :

8.1. distance entre axes de transistors connectés par la diffusion et se faisant face
8.2. distance entre axes de transistors connectés par la diffusion mais décalés ou inégaux
8.3. distance entre axes de transistors non connectés par de la diffusion
8.4. distance entre bouts de transistors connectés par le polysilicon
8.5. distance entre bouts de transistors non connectés par le polysilicon
8.6. distance entre axe de transistor et bout de transistor

Les règles concernant les **caissons** sont :

12.1. distance entre 2 caissons
12.2. distance entre caisson et diffusion interne
12.3. distance entre caisson et diffusion externe
12.4. distance entre caisson et contact métal1/ diffusion interne
12.5. distance entre caisson et contact métal2/ diffusion externe
12.6. distance entre caisson et prise caisson
12.7. distance entre caisson et prise substrat
12.8. distance entre caisson et axe de transistor interne
12.9. distance entre caisson et axe de transistor externe
12.10. distance entre caisson et bout de transistor interne
12.11. distance entre caisson et bout de transistor externe

ANNEXE 5

RESULTATS EXPERIMENTAUX

1. EVALUATION EXPERIMENTALE DES DIVERS QUADTREES

Le comportement des quadtrees a été étudié sur les description symboliques et aussi sur les descriptions logiques des circuits réels pour de petites et moyennes cellules composées d'éléments primaires (rectangles / segments / polygones) et d'éléments structurés représentés par leur placement (vue externe des sous-cellules ou instances de modèles) [BER 85]. Ces cellules provenaient du circuit *Microprocesseur de Traitement du Signal* réalisé au *Centre National d'Etudes des Télécommunications de Grenoble.*

Différents seuils de partage quaternaire et binaire furent choisis en mode relatif c'est-à-dire que les résolutions quaternaires et binaires (Q-depth et B-depth) paramétraient la décomposition des données. Les différents mixed-quadtrees ainsi crées peuvent ainsi correspondre à des structures de données très différentes. Les cas suivants apparaissent :

1. Les **résolutions quaternaires et binaires sont nulles** : le mixed-quadtree est alors une simple **liste d'éléments.**
2. La **résolution binaire est nulle** : le quadtree est un simple **quadtree adaptatif.**
3. La **résolution quaternaire est nulle** : le quadtree est un **arbre binaire adaptatif.**
4. Aucune des deux résolutions n'est nulle : le quadtree est alors un mixed-quadtree adaptatif (mixed-quadtrie). **Un mixed-quadtree de résolution quaternaire Q-depth et de résolution binaire B-depth est repésenté par la notation (Q-depth, B-depth).**

Ces seuils différents de résolution quaternaire et binaire nous ont permis d'étudier le comportement de ces divers structures et de les comparer entre elles *(figures 2 et 3).* Les algorithmes sont écrits en Pascal et ont été implémentés sur un VAX 11 / 782. Les quadtrees ainsi implémentés et évalués utilisent un seul niveau de mémoire, la mémoire interne. Les **résolutions maximales** de partage des données **correspondent** toujours pour les différents exemples traités à la réalisation de **perfect trees**; les "perfect trees" sont des arbres adaptatifs qui n'ont qu'un élément (en moyenne) par nœud terminal.

Un **espace mémoire de O (K)** est en moyenne utilisé **par chaque élément** rangé dans le quadtree avec **K constante multiplicative** pour la mémoire nécessaire à une collection de N éléments. Cette constante multiplicative dépend des résolutions. Dans les cas qui ont été étudiés et pour tous les types de **mixed-quadtrees** créés, cette **constante multiplicative de l'espace des données n'a jamais dépassé 2.5** et un **facteur multiplicatif de 1.5** correspond généralement à de **bonnes performances** des arbres ainsi construits. L'évolution comparative de l'espace mémoire ajouté a été étudié en facteur des résolutions quaternaires et binaires *(figure 5);* elle montre l'intérêt des partitions binaires moins coûteuses que les partitions quaternaires et aussi efficaces pour des recherches géométriques de faible surface *(figure 1).* Les mixed-quadtrees adaptatifs réalisent ainsi une structure plus efficace que les simples quadtrees adaptatifs pour différentes aires de recherche à égalité de mémoire ajoutée.

Ainsi les mixed-quadtrees sont des structures moins coûteuses en mémoire que les 4-D trees (qui on un facteur multiplicatif 4); le coût ajouté par les mixed-quadtrees aux simples listes d'éléments paraît peu supérieur au coût ajouté par les "multiples storage quad trees" (avec K= 1.25) [BRO 86]; les "multiples storage quad trees" doivent stocker les éléments de manière multiple dans les différents sous-arbres.

Les mixed-quadtrees paraissent aussi intéressants que les 4-D trees [ROS 85] pour la recherche des éléments couvrant (au moins partiellement) une petite région avec un ratio du nombre d'éléments examinés au nombre d'éléments trouvés variant en moyenne de 2 pour les moyennes fenêtres à 8 pour les petites fenêtres (point search) pour des cellules d'environ 80 transistors (700 rectangles) *(figure 1 ,qd (4,6)).*

De faibles seuils de partage (grande résolution Q-depth ou B-depth) sont meilleurs pour de petites fenêtres de recherche et de larges seuils de partage sont meilleurs pour de grandes fenêtres *(figure 4).* Les quadtrees adaptatifs sont meilleurs pour les grandes fenêtres et les arbre binaires adaptatifs sont meilleurs pour les petites fenêtres et les recherches sous des points. Ainsi les **mixed-quadtrees réalisés avec un large seuil de partition quaternaire et un faible seuil de partition binaire réalisent un bon compromis** entre ces deux structures *(fig.4 & fig.5).* De l'ordre de 2 ou 3 secondes pour environ 700 éléments les temps de construction des quadtrees et mixed-quadtrees

sont raisonnables *(figure 1)*. Ils sont souvent négligables devant la durée totale du process qui les a créés *(chap.5)*.

Une question essentielle est souvent posée par les utilisateurs potentiels d'une telle méthode de recherche : **Quelle est la taille minimale des cellules pour laquelle l'utilisation de quadtrees est intéressante ?** La réponse dépend beaucoup de la nature des applications susceptibles de bénéficier de cette méthode. Pour **une application de DRC conventionnel**, sur des cellules sans hiérarchie, l'expérience a montré, et le calcul a confirmé, qu'**à partir de 10 transistors** (et donc moins d'une centaine de rectangles) la méthode de recherche géométrique devenait intéressante et **qu'au delà les gains devenaient très vite substantiels**. Pour des applications telles que **les ERC conventionnels** la méthode géométrique est profitable pour de très petites cellules de **seulement quelques transistors;** cette application bénéficie alors d'une recherche rapide des quelques éléments situés sous une très petite fenêtre de recherche par rapport à la surface globale de la cellule et les gains réalisés sont alors substantiels (chap. 5).

Les diverses applications qui utilisent l'interface d'utilisation des mixed-quadtrees trouvent généralement leurs **meilleurs seuils de partage ou résolutions quaternaire et binaire** par **optimisation du temps global d'un process** sur une cellule type ou sur un ensemble de cellules (conceptions) très diverses.

Les **applications hiérarchiques** utilisent la hiérarchie des mixed-quadtrees induite par la hiérarchie de conception pour des **recherches géométriques** simples ou hiérarchiques (range searching); mais elles les utilisent aussi comme des index géométriques qui permettent **l'accès géométrique** aux éléments des instances de cellule par **recherche géométrique dans une région** (range searching) sur leur quadtree associé ou par **recherche géométrique exacte** (exact match / query) [KNU 73]. Cette double fonction de recherche et d'accès géométriques assurent l'intérêt et l'efficacité globale de cette méthode pour des applications hiérarchiques. Si **les divers quadtrees de la forêt hiérarchique ainsi induite par une conception VLSI** paraissent individuellement peu équilibrés ils **sont en fait équilibrés "de manière cachée"** dans la profondeur de la hiérarchie de conception puisque la distribution des rectangles / atomes de géométrie est de manière cachée assez uniforme [BEN 80]. Il est ainsi facile de les optimiser globalement par un **paramètrage absolu et global des seuils de partition quaternaire et binaires.**

Cet accès géométrique a été conçu et **testé sur de petites et moyennes conceptions VLSI sans hiérarchie et sur des conceptions hiérarchiques**. Il n'a donc pas été évalué pour des conceptions complètement instantiées c'est-à-dire pour N grand (nombre de rectangles / d'atomes). Cependant cette méthode doit être aussi efficace pour de grandes valeurs de N. Afin de s'assurer globalement de l'efficacité de cette méthode sur de plus grandes cellules, **N cellules sans hiérarchie** d'environ 80 transistors ont été **juxtaposées et expansées**; le **DRC conventionnel** sur la cellule sans hiérarchie ainsi obtenue s'est réalisé en un **temps de $N^{1.1}$ fois le temps obtenu sur une seule cellule**. Ce résultat laisse espérer un bon comportement de la méthode aux grandes valeurs de N et pour le même type de distribution des données.

ratio A0 / A	éléments de A	éléments examinés / éléments trouvés		
en % d'éléments	nombre	(0,0)	(4,0)	(4,6)
100 %	691	1	1	1
42 %	294	2.3	1.65	1.26
10 %	71	9.7	3.34	2.15
3.47 %	24	28.82	6.37	2.37
1.01 %	7	99.00	18.45	3.72
0.29 %	2	345.82	64.10	9.51
0.14 %	2	345.82	64.10	8.00
0.00 %	1	691.	128.00	16.00
	0			
BUILD		1 s	2 s	3 s

Figure 1 **Ratio du nombre d'éléments examinés par le nombre d'éléments trouvés** dans des régions d'aire A ; A0 est l'enveloppe de la cellule;
(p,q) : quadtree de résolution quaternaire p et de résolution binaire q.
BUILD : temps de construction du quadtree (p,q).
Remarque : Le graphique correspond aux moyennes de 30 mesures réalisées sur des fenêtres placées aléatoirement; les petites fenêtres qui tombaient dans un espace complétement vide de la cellule et correspondaient à un temps de recherche presque nul ont été éliminées de la moyenne et regénérées (effets de bords).

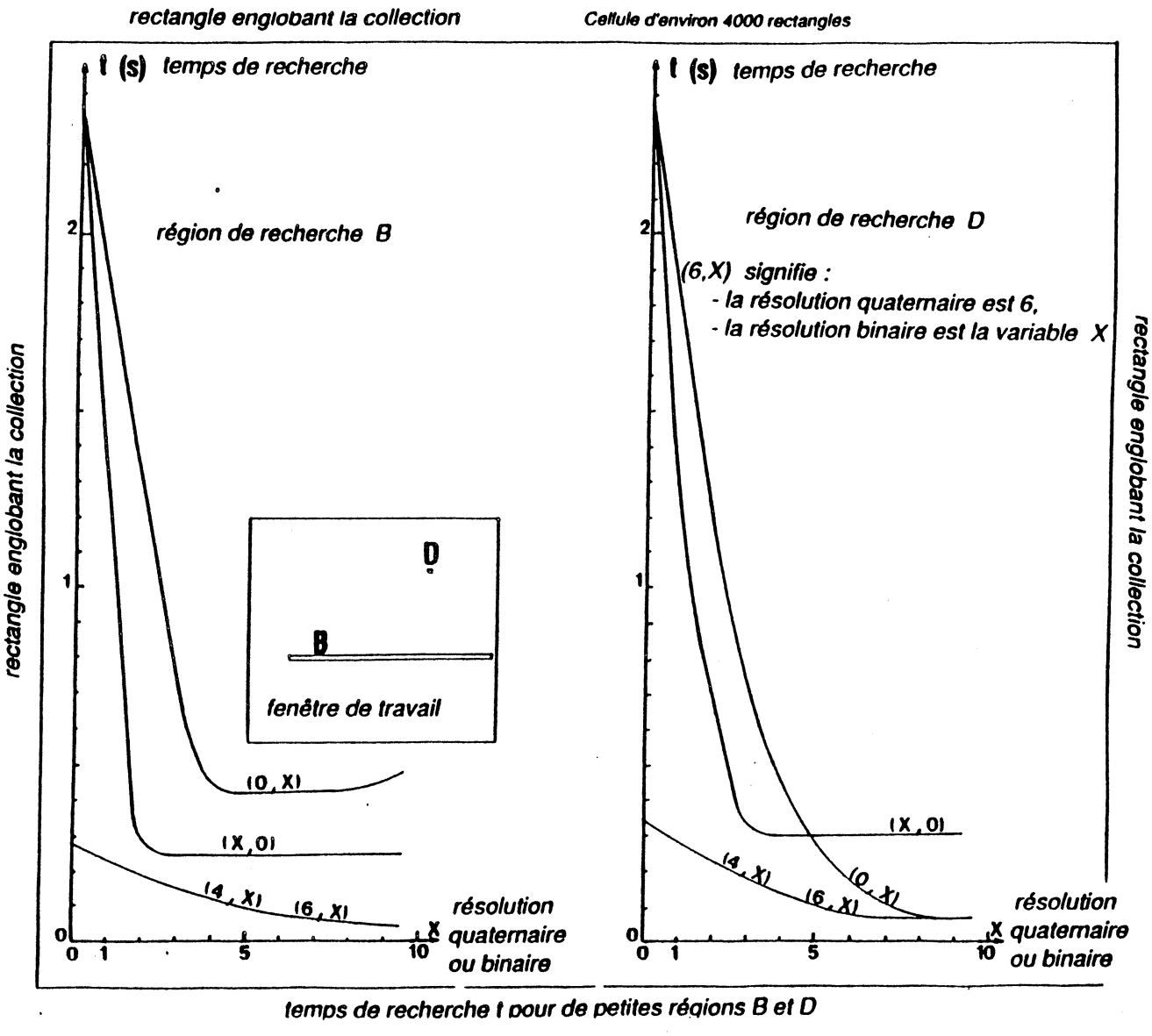


Figure 2 : ***Performance des recherches géométriques*** *sur une conception correspondant à environ 4000 rectangles, pour de petites régions de sélections relativement à la taille du rectangle englobant la cellule et* ***en fonction des résolutions quaternaire et binaire des quadtrees****, à gauche pour la fenêtre de dimension B; à droite pour la fenêtre de dimension D;* ***pour (6, X) lire la résolution quaternaire est 6, la résolution binaire est X.***

Remarque : Le graphique correspond à la moyenne de 30 mesures, estimée à moins de 5 % dans l'intervalle de confiance à 0.95.

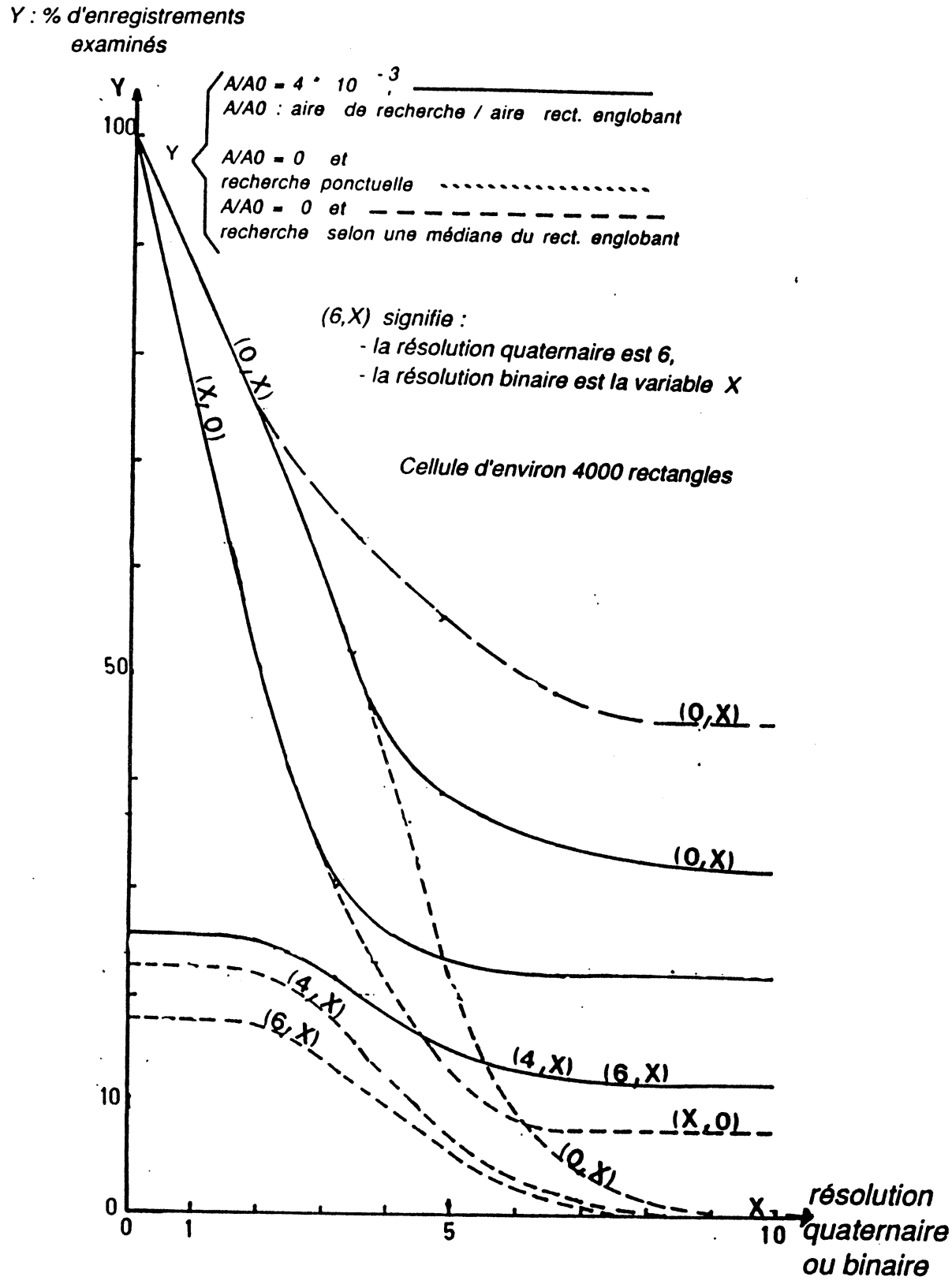


Figure 3 - **Pourcentage d'éléments examinés par diverses recherches géométriques** *sur une cellule RAM (4000 rectangles) du circuit µpts.*

Y = (nombre d'éléments examinés/ nombre d'éléments trouvés) * 100;

Les fenêtres de sélection d'aire A sont données en fonction de leur dimension relativement au rectangle enveloppe de la cellule $A = A0 * 2^{-p} * 2^{-q}$, ***les régions d'aire non nulle sont en trait plein;*** *les régions associées à des segments sont en pointillé long* ***(p=10, q=0); les régions associées à des pics (point searchs) sont en pointillés courts (p=10, q=10)***

pour (6, X) lire la résolution quaternaire est 6, la résolution binaire est X.

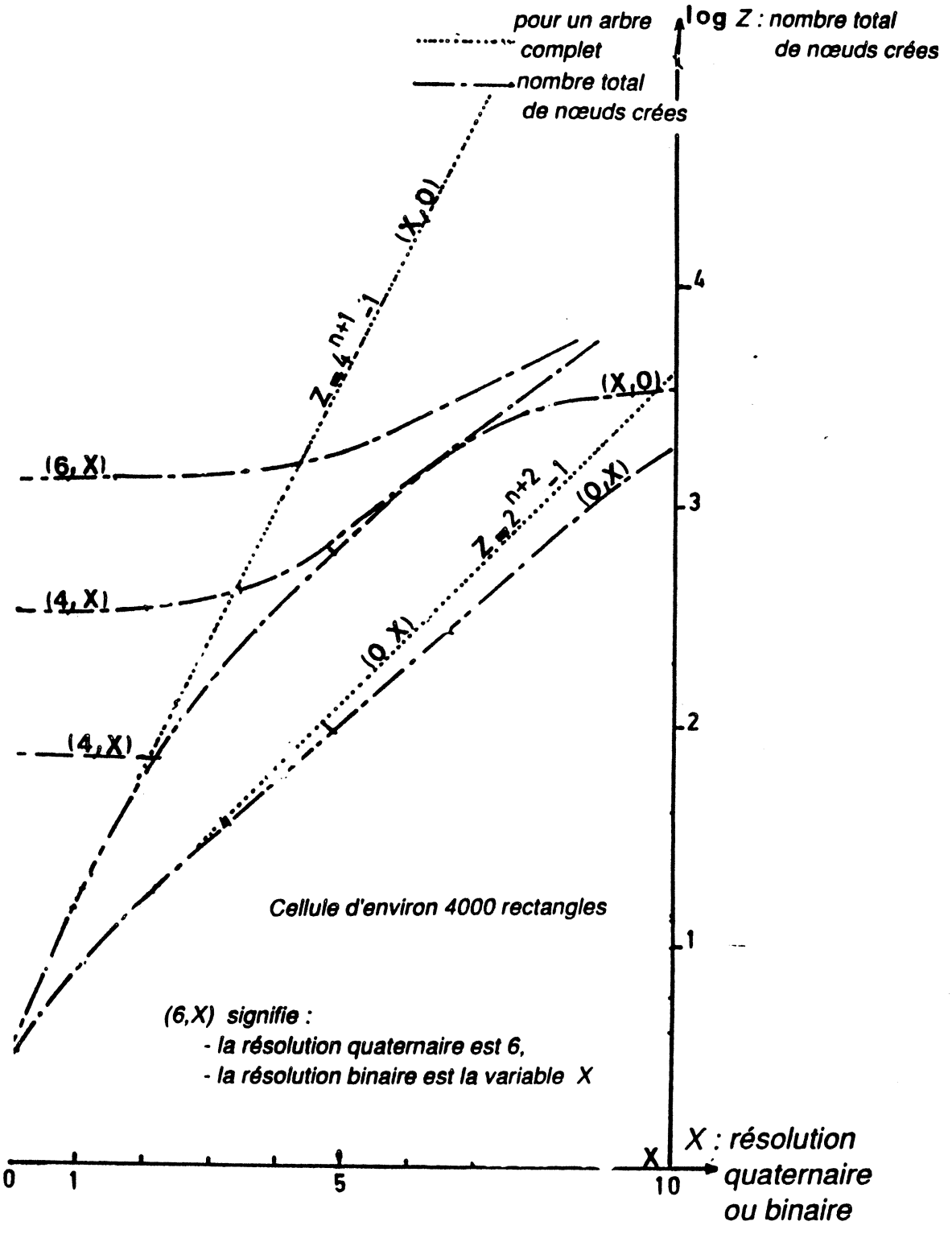


Figure 4 - ***Espace mémoire utilisé en fonction des valeurs des résolutions quaternaire et binaire des divers quadtrees réalisés*** *sur une cellule RAM (4000 rectangles).*
Z = noeud créés (binaires et quaternaires); (en log_{10} Y)
pour (6, X) lire la résolution quaternaire est 6, la résolution binaire est X

Les lignes pointées en (a) représentent la mémoire théorique d'un arbre complet de même paramètres. ***L'espace mémoire Z est en pointillés longs.***

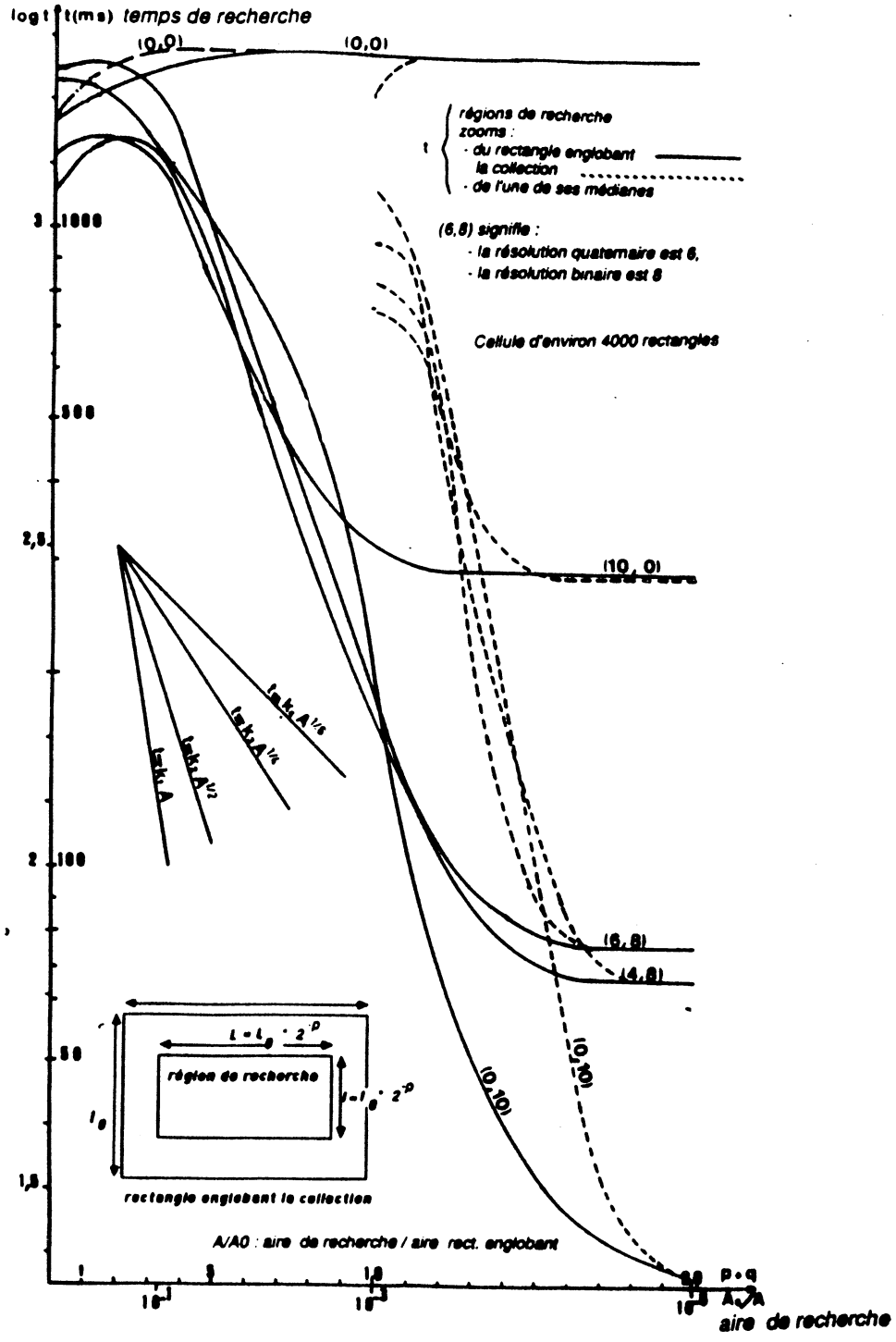


Figure 5: ***Temps de recherche t, en fonction des aires des régions de recherche pour divers quadtrees*** *créés sur la cellule RAM ('4000 rectangles);*

A0 : aire du rectangle enveloppe de la cellule; $A0 = L * \ell$

A : aire de la région de sélection donnée comme suit $L = L0 * 2^{-p}$, $\ell = \ell 0 * 2^{-q}$ *avec p , q : (0..10); ainsi p+q est proportionnel à* $\log (A/A0)$ *marqué sur l'axe horizontal.*

Les recherches géométriques d'aire presque nulle (segments) sont en pointillé long , les *recherches géométriques d'aire non nulle sont des* ***"zooms" du rectangle enveloppe de la cellule , représentés en trait plein.***

(4,8) signifie que la résolution quaternaire est 4 et la résolution binaire est 8.

Remarque : Le graphique correspond aux moyennes de 50 mesures par fenêtre; ces moyennes sont estimées à moins de 5 % dans l'intervalle de confiance à 0.95.

Le premier extrémum de chaque courbe est un effet de bord. Le deuxième extrémum correspond à la saturation du quadtree.

2. L'ANALYSE DU LAYOUT, RESULTATS EXPERIMENTAUX

2.1. L'amélioration de l'analyse conventionnelle

Quelques cellules d'environ **80 transistors** réalisées dans un **symbolisme Stick** *(annexe 4)* ont été utilisées pour mettre au point les quadtrees et optimiser les **algorithmes conventionnels** de **vérification des régles de dessin (D.R.C.)** et **d'extraction des nœuds électriques (E.N.E.)**. Les résultats présentés en *figure 6* sont les différents temps de traitement obtenus pour ces outils conventionnels avec des quadtrees différemment paramétrés . **Qd (p, q)** signifie **quadtree de résolution quaternaire p et de résolution binaire q** *(chapitre 4)*. Les meilleurs temps obtenus pour les **D.R.C.** correspondaient pour les **quadtrees (3,3)** à l'égalité des dimensions des régions de sélection du D.R.C. et de partition des quadtrees. Les **mixed-quadtrees** qui optimisent les **D.R.C.** produisent aussi de bons résultats pour les **E.N.E.**

taille de la cellule	***QD (p,q)***	***temps du D.R.C.***	***temps de l' E.N.E.***
78 transistors (691 rectangles)	(0,0)	298 s	350 s
	(3,0)	101 s	120 s
	(3,3)	60 s	30 s
	(0,5)	61 s	30 s
	(2,4)	59 s	29 s

Figure 6 - ***Optimisation d'algorithmes conventionnels de Vérification des Régles de Dessin (D.R.C.) et d'Extraction des Nœuds Electriques - (E.N.E.)*** -

Remarque : Le nombre de rectangles élémentaires dans une région de sélection de cet outil conventionnel de DRC est de 27 à 5 éléments près, pour ce D.R.C. ; les régions de sélection sont définies par la plus grande garde à vérifier pour l'ensemble du layout , ce qui pour le D.R.C. n'est pas optimal.

Les cellules précédemment utilisées pour trouver les meilleurs quadtrees, ont été ensuite **juxtaposées** et **mises à plat** afin d'étudier le **comportement des quadtrees** pour de plus **grosses cellules**. Lorsque **N cellules** sont ainsi juxtaposées, le **temps du D.RC. conventionnel** obtenu pour ce layout est **d'environ** $N^{1.1}$, ce qui correspond à un très bon comportement. La *figure 7* donne, en colonnes 2 et 3, les temps du D.R.C. et de l'E.N.E. pour les cellules de base de 82 transistors et pour des **cellules cellules formées de 5 cellules de base** - d'environ 400 transistors. Les *colonnes 5 et 6* permettent la **comparaison avec** les outils D.R.C. et E.R.C. avant leur optimisation; ces temps ont été obtenus en utilisant l'**outil conventionnel avant son optimisation** par recherche géométrique. Le gain réalisé par l'utilisation des quadtrees pour ces outils est exprimé dans la *figure 8*. La *colonne 4* de la *figure 7* donne les **temps de création** des quadtrees qui optimisent les D.R.C. et E.N.E. ; il est à remarquer qu'ils constituent un faible pourcentage - **< 8 %** - du temps de traitement.

Les **mixed-quadtrees** implémentés sont efficaces pour ce **D.R.C conventionnel** à partir de **10 transistors** et à partir de **quelques transistors** pour l'**E.N.E.** Pour de plus **grosses cellules** les **gains** deviennent vite **très intéressants** si l'on examine alors les gains obtenus par rapport à l'utilisation du même outil avant son amélioration *(figure 8)*. L'optimisation des cellules de 80 et de 400 transistors a montré que des mixed-quadtrees réalisaient un très bon compromis entre le temps du traitement et la mémoire qu'ils utilisent; ces mixed-quadtrees correspondent, pour ces cellules, à un **partage quaternaire faible** - entre 400 et 800 rectangles dans un nœud terminal - et un **partage binaire beaucoup plus fin** - peu rectangles dans un nœud terminal , de cette partition secondaire.

taille de la cellule	temps du D.R.C.	temps de l' E.N.E.	temps de création	temps DRC avant optimisation	temps ERC avant optimisation
78 transistors	59 s	29 s	2.5 s	298 s	350 s
390 transistors	362 s	210 s	12 s	7450 s	8750 s
10 transistors	4.6 s	3 s	0.3 s	4.6 s	5.4 s

figure 7 - **Temps de vérification des Régles de Dessin (D.R.C.) et d'Extraction des Nœuds Electriques (E.R.C.) en fonction de la taille des cellules** -

taille de la cellule	Gain D.R.C.	Gain E.N.E
10 transistors	0 %	45 %
78 transistors	80 %	92 %
390 transistors	95 %	98 %

figure 8 - **Gains obtenus par l'amélioration des algorithmes conventionnels de D.R.C. et d'E.N.E.** -

Remarque : L'extraction des nœuds électriques pour cet outil conventionnel n'utilisait pas la méthode **Union-find** *et la fusion des composantes connexes utilisait un algorithme en N^2. La méthode* **Union-find** *doit assurer des performances bien meilleures aux algorithmes d'analyse hiérarchiques actuellement développés.*

2.2. L'analyse hiérarchique et l'extension des outils conventionnels

Après cette première phase d'amélioration des outils conventionnels d'analyse, la technique des halos fut essayé dans le but d'améliorer encore la vérification de layouts hiérarchiques *sans recouvrement de sous-cellules (chap. 2)*. Après la vérification d'une cellule, son halo était alors construit par filtrage des éléments primaires inclus dans son halo, afin de remplacer la cellule dans les niveaux supérieurs de la hiérarchie *(fig. 4)*. Un algorithme de construction dynamique du halo d'une cellule, qui évite la sauvegarde de cellules halos, est donné dans *l'annexe 2.*

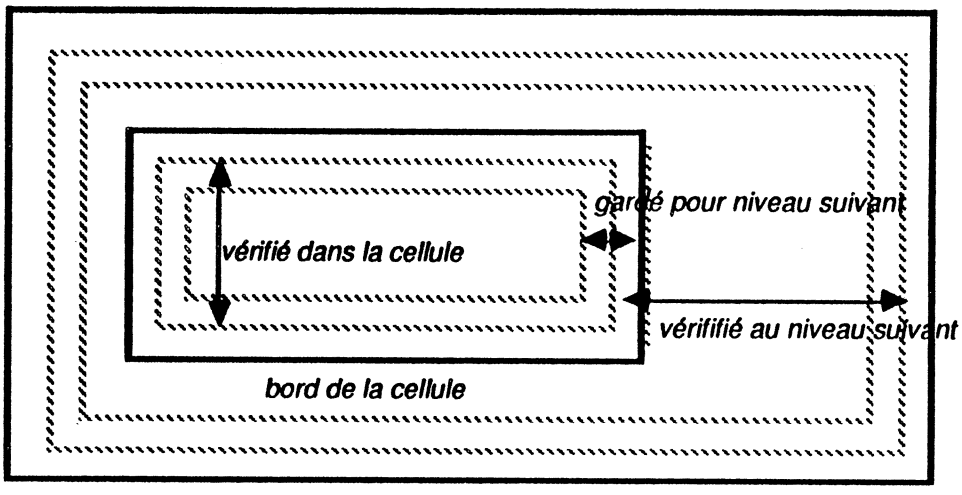


figure 9 - ***La technique des halos*** - *La valeur maximum (x2) de toutes les règles de dessin définit la taille du halo d'une cellule. Ces halos sont gardés et seront instanciés dans les divers contextes qui vont les utiliser. La connectivité électrique est propagée si les éléments qui constituent les halos sont aussi porteurs des équivalences électriques. La réalisation de leurs équivalences dans les divers contextes qui les utilisent propage alors la connectivité électrique...Ainsi l'analyse d'une cellule est réalisée sur la partie non hiérarchique du layout de la cellule augmentée du layout instancié des halos de ses sous-cellules. La nécessité d'utiliser une hiérarchie sans recouvrements ou avec des recouvrements limités par l'interface que constituent les halos de sous-cellules est un inconvénient de cette méthode; seuls les layouts décrits avec cette restriction hiérarchique peuvent être analysés. Mais l'inconvénient majeur de cette méthode est la taille des cellules halos qui rendent l'analyse hiérarchique peu intéressante sur un certain nombre de conceptions - en particulier pour l'analyse des tableaux de cellules. La mémoire secondaire utilisée pour conserver ces halos est non négligeable.*

La *figure 10* présente la taille des halos qu'il était nécessaire d'extraire pour diverses tailles de cellules afin d'assurer un D.R.C. hiérarchique selon la méthode présentée sur la *figure 9*. La *colonne 1* indique la **taille des cellules**. La *colonne 2* indique la **taille des halos** correspondants en pourcentage par rapport à la taille des cellules, et en conséquence le **supplément de mémoire** utilisé. La *colonne 3* indique le **gain partiel** réalisé pour le D.R.C. d'une cellule qui remplacerait cette instance de cellule par son halo *(cf. figure 7)*.

L'extension du ***D.R.C. Stick conventionnel*** par la *technique des halos* ne peut être envisagée qu'avec des halos larges d'au moins de 2 fois la règle maximale des règles de dessin. Or la règle de garde maximale d'un ensemble de règles Stick est de 6.5 λ – avec λ = 3.00 microns; cette garde maximale est donc pour des cellules de surface 100 λ * 100 λ et d'environ 80 transistors relativement élévée. Les valeurs des contraintes maximales observées sur des ensembles de règles de dessin des layouts physiques paraissent généralement bien inférieures à cette valeur.

La prise en compte de 2 fois la règle maximale des règles de dessin est une des causes de ce manque d'efficacité pour ce D.R.C. symbolique. Il était difficile cependant d'utiliser autrement ce D.R.C. conventionnel qui n'effectue pas entre deux éléments des vérifications purement binaires et doit éventuellement analyser leur voisinage.

Cette méthode nous a donc paru peu efficace pour améliorer ***le D.R.C. Stick conventionnel*** pour des circuits constitués essentiellement de petites cellules. En effet, pour des cellules constituées par des tableaux de petites cellules, le gain peut être nul et même globalement négatif. Une solution proposée à ce problème est le groupage d'un certain nombre de cellules [WAG 84]. La reconnaissance de la régularité et l'étude des interactions relatives entre sous-cellules sont alors aussi utiles *(Chap. 2)*.

REFERENCES BIBIOGRAPHIQUES

REFERENCES

[ABE 83] D. J. ABEL, J.L. SMITH "A Data Structure and Algorithm Based on a Linear Key for a Rectangle Retrieval Problem", Computer Vision, Graphic and Image Processing, no. 24, 1-13, 1983.

[ABE 84] D. J. ABEL, "A B-TREE Structure for Large Quadtrees", Computer Vision, Graphic and Image Processing, no. 27, 19-31, 1984.

[AHO 74] A.V. AHO, J.E. HOPCROFT AND J.D. ULMAN "The Design and Analysis of Computer Algorithms", Addison-Wesley, reading, Mass. 74.

[AHO 83] A.V. AHO, J. HOPCROFT, J. ULLMAN "Data Structures and Algorithms", reading, Addison-Wesley, 1983, . pp. 181-193.

[ANN 84] J. ANNEVELINK, P. DEWILDE, T.G.R. van LEUKEN "Hierarchical Verification of VLSI Artwork", ISCAS 1984, Canada.

[ARN 82] M. ARNOLD, H. OUSTERHOUT "LYRA : A New Approach to Geometric Layout Rule Checking", 19 th Design Automation Conference, 1982.

[ASA 86] T. ASANO, M. SATO, T. OHTSUKI "Computational Geometry Algorithms", Layout Design and Verification, T. Ohtsuki (editor), Elsevier Science Publishers B.V. (North-Holland), 1986.

[BAK 80] C. BACKER, "Artwork Analysis Tools for VLSI Circuits", T.R. MIT:LCS/ TR-239, Massachusettes Institute of Technology, Cambridge, Massachussetts 02139, May 1980.

[BAL 82] M. BALES "Layout Rule Spacing of Symbolic Integrated Circuit Artwork", Memorandum UCB/ER1 M82/72, May 82.

[BAI 77] H.S. BAIRD "Fast Algorithms for LSI Artwork Analysis, 14 th Design Automation Conference, 1977.

[BAR 80] E.E. BARTON, I. BUCHANAN, The Polygon Package - Computer -Aided Design 12, 1980, 3-11.

[BAS 83] J. BASTIAN, M. ELLEMAN, P.J. FOWLER "Symbolic Parasitic Circuit extractor for Circuit Simulation", 20th Design Automation Confererence, 1983.

[BEN 75] J.L. BENTLEY "Multidimensional Binary Search Used for Associative Seaching", Comm. ACM 18, Sept., 1975.

[BEN 75 & STA] J.L. BENTLEY, D.F. STANAT "Analysis of Range Searches in Quad trees, Inf. Proc. Let. 3, July 1975.

[BEN 78] J.L. BENTLEY "Algorithms for Reporting and Counting Geometric Intersections", T.R. CMU-CS-78-135, 1978.

[BEN 79] J.L. BENTLEY, J.H. FRIEDMAN "Data Structures for Range Searching", ACM Computer Surveys, Vol. 11, no 4, 1979

[BEN 79 & FRI] J.L. BENTLEY, J.H. FRIEDMAN "Mutidimensional Binary Search Trees in Database Applications", IEEE Transaction on Software Engineering, Vol. SE-5, no. 4, 1979.

[BEN 80] J.L. BENTLEY, D. HAKEN, R. HON "Statistics on VLSI Designs", Carnegie-Mellon Univ. , T.R. CMU-CS-80-111, April 1980, Dep. of Computer Science.

[BEN 80 & HAK] J.L. BENTLEY, D. HAKEN, R. HON "Fast Algorithms for VLSI Task", 1980, IEEE Compcon Spring'80, 1980.

[BEN 80 & WOO] J.L. BENTLEY, D. WOOD "An Optimal Worst Case Algorithm for Reporting Intersections of Rectangles", IEEE Transactions of Rectangles", IEEE Transaction on Computer, 1980.

[BEN 81 & OTT] J.L. BENTLEY, T. OTTMANN "The Complexity of Manipulating Hierarchically Defined Sets of Rectangles", T.R. CMU-CS-81-109, Carnegie-Mellon Univ., 1981.

[BEN 82] Bening L.C., Lane T.A., Alexander C.R. ans Smith J.E. , "Developpements in Logic Network Path Delay Analysis, Proc. 19th. Design Automayion Conférence, Las Vegas, Nevada, June 1982, pp.605-615.

[BER 85] J. BERGER "Quad-tree Hierarchy for Circuit Data Retrieval in Structured Design", pp. 650-653, IEEE International Conference on Computer Design, Port Chester, NY, 1985.

[BER 86] J. BERGER "Incremental Design Verifications Based Upon a Geometrical Access for Circuit Data Retrieval", pp. 394-402, ICTC 86, Integrated Circuit Technology, Conference Limerick, Ireland, C; Burkley, E. McQUADE, A. RAHMAN editors, 1986.

[BER 86] J. BERGER, G. MAZARE "Hierarchical Design Verifications based Upon a Topological Access to Circuit Data", pp. 19-21, Twelve European Solid-State Circuit Conference, Delft, The Netherland, 1986.

[BER 86] J. BERGER, G. MAZARE "A Range Searching Sub-System Used to Perform Efficient VLSI Design Checks", pp. 408-411, IEEE International Conference on Computer-Aided Design, Santa Clara, California, 1986.

[BEY 82] A.M. BEYLS, B. HENNION, J. LECOURVOISIER, G. MAZARE, A. PUISSOCHET "A Design Methodology based upon Symbolic Layout and Integrated CAD tools", 19th Design Automation Conference, 1982.

[BRO 86] R.L. BROWN "Multiple Storage Quad Tree : A Simple Faster Alternative to Bisector List Quad Trees", IEEE Transactions On Computer-Aided Design, Vol. CAD-5, No. 3, July 1986.

[CHAI 81] S. CHAIKEN, D. KLEITMAN, M. SAKS, and J. Shearer, "Covering Regions by Rectangles", SIAM J. Computing, vol. 2, pp. 225-231, 1973.

[CHA 80] S. CHAO, Y. HUANG, L.M. YAM "A Hierarchical Approach for Layout Versus Circuit Consistency Check", 17 th Design Automation Conference, 1980.

[CHA 84] P.T. CHAPMAN, K. CLARK "The Scan Line Approach to Design Rules Checking : Computational Experiences", 21 th. Design Automation Conference, 1984.

[CHA 79] B.M. CHAZELLE and D.P. DOBKIN, "Decomposing a Polygon into its Conex Parts, "proc. 11th Annu. ACM Symp. on Theory of Computing, pp. 38-48, 1979.

[CHE 86] J. CHERRY, H. SHROBE, N. MAYLE, H. MINSKY, K. RETI, N. WESTE "NS : An Integrated Symbolic Design System", VLSI'85, & Advance in CAD for VLSI, E. Hörbst, North-Holland, IFIP 86.

[CHO 85] Y. E. CHO "A Subjective review of Compaction", 22th Design Automation Conference, 1985.

[COM 81] M.A. COMEAU "A Coordinate Reference System for Spatial Data Processing, CLDS Technical Bulletin, no. 3, Canada Land Data Systems Division, Ottawa, Ontario, 1981.

[COM 79] D. COMER "The Ubiquitous B-Tree", Computing Surveys 11, 1979.

[COM 81] M.A. COMEAU, "A Coordinate Reference system for spatial data processing, CLDS Technical Bulletin 3, Lands Directorate, Environment Canada, 1981.

[COO 79] G. COOK "The Structural and algorithmic basis of a geographic data base" Cook/ 1-Cook/30, in International Advanced Studies Symposium on Topological Data Structures For Geographic Information Systems, Dedham, Mass. , Oct. 16-2&, 1977 (G. Dutton, Ed.), Harvard University Laboratory for Computer Graphics and Spatial Analysis, 1978 / 1979.

[COR 81] L.V. CORBIN, "Custom VLSI Electrical Rule Checking on an Intelligent Terminal", 18th. Design Automation Conference, 1981.

[DRA 85] M. DRANEY, "An Interactive Design Rule Checker", VLSI Design, Feb. 1985.

[DREY 87] G. DREYFUS "Chaos et CAO ou la Methode du Recuit Simule", AFCET / Interfaces, no 53, 1987.

[DUN 78] A.E. DUNLOP "SLIP : Symbolic Layout of Integrated Circuits with Compaction", CAD, vol. 10, no. 6, nov. 1978.

[DUN 80] A.E. DUNLOP "SLIM - The Translation of Symbolic Layout into Mask Data", 17th Design Automation Conference, 1980.

[EDE 80] H. EDELSBRUNER "Dynamic Data Structures for Orthogonal Intersections Queries", Rep. F59, tech. Univ. Graz, Institute Für Informationsverarbeitung 1980.

[EDE 82] H. EDELSBRUNER "Intersection Problems in Computational Geometry", Ph. D. Thesis, Rep. 93, IIG, Technische Univ. Graz, Austria, 1982.

[ENT 85] G. ENTENMAN, S. W. DANIEL "A Fully Automatic Hierarchical Compactor", 22th. Design Automation Conference, 1985.

[FIN 74 & BEN] R. FINKEL, J.L. BENTLEY "Quadtrees : A Data Structure for Retrieval on Composite Keys", Acta Inf. 4, 1974.

[FIS 72] M. J. FISCHER, "Efficiency of Equivalence Algorithms", in Complexity of Computer Computations (R.E. Miller and J. W. Thatcher, eds.) pp. 153-168.

[FRA 84] D.S. FRANZBLAU and D. J. KLEITMEN, "An Algorithm for Constructing Regions with Rectangles : Independence and minimum generating sets for Collections of Intervels", Proc. 16th Annu. ACM Symp. on Theory of Computing, pp 167-174, 1984.

[FRI 77 & BEN] J.H. FRIEDMAN, J.L. BENTLEY, R.A. FINKEL "An Algorithm for Finding Best Matches in Logarithmic Expected Time", ACM Trans. Math. Soft. 3, Sept. 1977.

[FOK 83] J. T. FOKKEMA, T.G.R. VAN LEUKEN "An Efficient Datastructure and Algorithm for VLSI Artwork Verification", IEEE International Conference on Computer Design,1983.

[GAL 64] B. GALLER, M. FISCHER "An Improved Equivalence Algorithm", com. ACM, May 64.

[GAL 83] L. GALLOT "Techniques Informatiques dans la Verification des Gardes des Circuits Integres", These de 3ieme Cycle, univ. Paris Sud, 1983.

[GAR 82] I. GARGANTINI, 1982, An Effective Way to Represent Quadtrees. Comm. ACM 25, 12 (Dec), 905-910.

[GRE 86] J. W. GREENE, "Layout-to-Layout Compaction for Technology Conversion", VLSI Systems Design, nov. 86.

[GUP & HON] A. GUPTA, R.W. HON, "HEXT, A Hierarchical Circuit Extractor", "Journal of VLSI and Computer systems 1(1), pp. 23-39, Computer Science Press (1983).

[HED 85] T. HEDGE, W. DAWSON, Y.E. CHO "Bitmap Graph Build Algorithm for compaction", ICCAD 1985.

[HEI 82] C. HEINTZ "Un extracteur de Circuits Intègres", These de 3ieme cycle, U. Paris Sud, Centre d'Orsay, 1983.

[HIT 82] Hitchcok R.B. "Timing Verification and th timing Analysis Program, Proc. 19th Design Automation Conférence, Las Vegas, Nevada, June 1982, pp. 594-604.

[HON 83] R. W. HON "The Hierarchical Analysis of VLSI Designs", Ph.D. Thesis, Columbia University, Departement of Computer Science, NY., NY. 10027.

[HSU 79] M. HSUEH "Symbolic Layout and Compaction of Integrated Circuits", Memo. UCS/ER1 N79/80, univ. of California, Berkeley.

[JOH 82] S. JOHNSON, "Hierarchical Design Validation Based on Rectangles", 1982 Conference on Advanced Research on VLSI, M.I.T.

[JON 81] L. Jones and S.S. Iynegar, "Represenyation of a region as a forest of quadtrees, Proc. PRIP'81 Conference, Dallas, Texas, 1981, 57-59.

[JUL 86] C. JULLIEN, A. LEBLOND, J. LECOURVOISIER "A Database Interface for an Integrated CAD System", 23rd Design Automation Conference, 1986.

[KED 82] G. KEDEM "The Quad-tree : A Data Structure for Hierarchical On-line Algorithms", 19th. Design Automation Conference, 1982.

[KED 83] G. KEDEM, H. WANATABE "Optimisation Techniques for IC Layout and Compaction", 20th. Design Automation Conference, 1983.

[KLI 79] A. Klinger and M.L. Rhodes, Organisation and access of image data by areas, IEEE Trans. Pattern Anal. Mach. Intell. PAMI-1, 1979, 50-60.

[KNU 73] D. E. KNUTH "The Art of Computer Programming : Sorting and Searching", reading, Addison-Wesley, 1973.

[KOR 84] J. L. KORS, M. ISRAEL "An Interactive Graph Extractor", 21 th Design Automation Conference, 1984.

[LAU 81] U. LAUTHER "An N logN Algorithm for Boolean Mask Operations", 18th. Design Automation Conference, 1981.

[LIA 83] Y. LIAO "An Algorithm to Compact a VLSI Symbolic Layout with Mixed Constraints", IEEE transactions on Computer-aided Design, vol. cad-2, Ap. 83.

[LEC 84] J. LECOURVOISIER "Cassiopée : Un Systéme Intégré pour la CAO des VLSI", L'Echo des Recherches, no. 118, 4éme trimestre 1984. Centre National des Télécommunications.

[LEU 86] T.G.R. VAN LEUKEN, J.T. FOKKEMA, J. LIEDORP "State Driven Hierachical Design Rule Verification", IEEE,may 5-7,1986 International Symposium on Circuits and Systems, San José, California, pp. 556-559.

[LEU 86] T.G.R. Van Leuken, "An Hierarchical and Technology Independent Design Rule Checker", T.G.R. Van Leuken, J. Lieorp. The Integrated Circuit Design Book, P. DEWILDE Editor, Delft University of Technology, 2.1-2.50.

[LEE 80] J. VAN LEEUWEN and D. WOOD "Dynamization of Decomposition of Decomposable Searching Problems", inf. proc. let 10, 1980.

[LUE 78] G. S. LUEKER "A Data Structure for Orthogonal Range Queries", Proc. of the 19th Annual IEEE Symposium on Foundations of Computer Science, pp-28-34.

[LIN 76] B.W. LINDSAY, B.T. PREAS "Design Rule Checking and Analysis of IC Mask Designs", 13th Design Automation Conference, June 76.

[LOS 79] P. LOSLEBEN, K. THOMSON "Topological Analysis for VLSI Circuits", 16 th Design Automation Conference,1979.

[MAT 84] T. MATSUYAMA, L. V. HAO, M. NAGAO "A File Organization for geographic information systems base on spatial proximity. Comput. Vision Gr. Image Process, 26, 3 (June), 303-318.

[McC 81] E. M. MCCreight, "Priority Search Trees", Tech. Rep XEROX PARC CSL-81-5, 1981.

[MCCA 84] S. MCCABE "A Program for Hierarchical Verification of VLSI Layouts", VLSI Design, Aug. 84.

[McCO 84] S.P. McCORMICK "A Circuit Extractor For IC Designs", 21th. Design Automation Conference, 1984.

[McCO 84] S.P. McCORMICK "Automated Circuit Extraction from Mask Descriptions of MOS Networks", Massachusetts Institute of Technology, Feb. 84, Master of Science.

[MCGR & WHI 80] J. MCGRATH, T. WHITNEY "Design Integrity and Immunity Checking", 1980.

[MEA 80] C. MEAD, L. CONWAY "Introduction to VLSI Systems", Addison-Wesley, reading, 1980.

[MOR 66] G. M. MORTON, "A Computer Oriented Geodetic Data base and a New Technique in file Sequencing", IBM Canada Ltd, 1966.

[NAN 86] S.K. NANDY, L.V. RAMAKRISSHNAN "Dual Quadtree Representation for VLSI Designs", 23 rd 21th. Design Automation Conference, 1986.

[NAN 86] S.K. NANDY, L.M. PATNAIK, "Linear Time Geometrical Design Rule Checker based on Quadtree Representation of VLSI Mask Layouts", CAD 1986.

[NEW 82 & FITZ] M. NEWELL, D. FITZPATRICK "Exploitation of Hierarchy in Analysis of Integrated Circuit Artwork", IEEE Transaction on CAD of Integrated Circuit and Systems, vol. CAD-1 no. 4, oct. 82.

[NIE 82] J. NIEVERGELT, F. PREPARATA "Plane Sweep Algorithms for Intersecting Geometric Figures, Com. ACM 25 (10), 1982.

[NIE 84] J. NIEVERGELT, H. HINTERBERGER "The Grid File : An Adaptable, Symetric Multikey File Structure, Institut fur Informatik, SEVCIK, ACM Transaction on Database Systems, Vol. 9, no. 1, March 84.

[NEL & SHA] B.J. NELSON, M.A. SHAND, "An Integrated, Technology Independent, High Performance Artwork Analyser for VLSI Circuit Design", Journal of VLSI and Computer Systems, Computer Science Press (Third Quarter 1985).

[NEW 78 &SPRO] W. M. NEWMAN, R. SPROUL "Principles of Interactive Computer Graphics", reading, McGRAW-HILL, International Student Edition, 1978.

[OHT 82] T. OHTSUKI, "Minimum Dissection of Rectilinear Regions", Proc. IEEE Internat.

Symp. on Circuits and Systems, Rome, pp. 225-231, 1973.

[OUS 81] J. K. OUSTERHOUT, "Cæsar : An Interactive Editor for VLSI Layouts", VLSI Design, vol. 2, no. 4, fourth quarter 1981, pp; 34-38.

[OUS 83] J. K. OUSTERHOUT, "Cristal : A Timing Verifier for NMOS VLSI Circuits", Third Caltetch Conférence on Very Large Scale Integration, March 1983, pp. 57-69.

[OUS 84 & HAM] J.K. OUSTERHOUT, G. HAMAKI, R. MAYO "Magic : A VLSI Layout System", 21 th Design Automation Conference, 1984.

[OUS 84] J. OUSTERHOUT, "Corner Stitching, A Data Structuring Technique for VLSI Layout Tools", IEEE Trans. on CAD, Vol. CAD-3, Jan. 1984.

[OUS 84b] J.K. OUSTERHOUT, "Switch-Level Delay Models for Digital MOS VLSI", Proc. 21 th Design Automation Conference, 1984.

[OVE 82] M.H. OVERMARS, J. Van LEEUWEN "Dynamic Multi-dimensional Data Structures based on quad and K-d Trees", Acta Inf. 17/3, 267-285.

[PAV 82] T. PAVLIDIS "Algoritms for Graphic and Image Processing", reading, Springer-Verlag.

[PRE 85] F. PREPARATA, M. SHAMOS "Computational Geometry", reading, Springer-Verlag, 1985.

[RIV 80] R.L. RIVEST, "A Description of a single-Chip Implémentation of the RSA Cipher", Lambda, forth quater 1980, pp.14-18.

[ROG 85] C.D. ROGERS, J.B. ROSENBERG, S.W. DANIEL "MCNC's Vertically Integrated Symbolic Design System", 22 th Design Automation Conference, 1985.

[ROS 85] J.B. ROSENBERG "Geographical Data Structures Compared, A Study ofData Structures Supporting Regions Queries", Transaction on Computer Aided Design, Vol. CAD 4, no. 1, Jan. 1985.

[ROS 83] J. ROSENBERG, N. WESTE "ABCD - A Better Circuit Description", MCNC tech. rep. no. 4983-0, MCNC of North Carolina, RTP NC277709 (Feb. 83).

[ROS 82] A. ROSENFELD, A.C. KAK "Digital Picture Processing", 2nd Edition Academic Press, 1982

[ROSET 82] A. ROSET "Une Méthode de Conception Symbolique pour le NMOS", T.R. NT/CNS/CCI/05, CNET-GRENOBLE. Private Communication, juillet 1982.

[SAM 80 & ROS] H. SAMET, A. RSENFIELD, 1980,. "Quad-trees Structures for Image Processing". In proc; of th 5th International Conference on Pattern Recognition (Miami beach, Fla, Dec). IEEE, New York, pp. 815-818.

[SAM 84] H. SAMET "The Quadtree and Related Hierarchical Data Structure", Computing Survey, Vol. 16, no. 2, June 84.

[SCH 81] L.K. SCHEFFER "A Methodology for Improved Verification of VLSI Designs without Loss of Area", proc. Caltech Conf. on VLSI, 1981.

[SCH 84] L. K. SCHEFFER, "The Use of the Strict Hierarchy for Verification of Integrated Circuits". Ph.D. 1984, Stanford University.

[SCH 85] L.K. SCHEFFER "Hierarchical Analysis of IC Artwork with User-Defined Rules", 22 th Design Automation Conference, 1985.

[SCH 86] L.K. SCHEFFER, R. SOETTARMAN "Hierarchical Analysis of IC Artwork with User-Defined Rules", IEEE Design and Test, 1986.

[SCHI 85] M. SCHIELE "Automatic Design Rule Adaptation of Leaf Cell Layouts", The VLSI Journal 3, 1985, 93-112.

[SCHL 83] M. SCHLAG, Y.Z. LIAO, C.K. WONG "An Algorithm for Optimal Two-Dimensional Compaction of VLSI Layouts, Integration, The VLSI journal 1, 1983.

[SCO 84] W. SCOTT, J. OUSTERHOUT "Plowing : Interactive Stretching and Compaction in Magic", 21 th Design Automation Conference, 1984.

[SCO 85] W. SCOTT, J. OUSTERHOUT "Magic Circuit Extractor", 22 th Design Automation Conference, 1985.

[SAH 80] S. SHANI, A. BHATT "The Complexity of design Automation Problems", Proc. 17th Design Automation Problems", Proc. 17th Design Automation Conference, 1980.

[SED 83] SR. SEDGEWICK, 1983, "Algorithms", reading, Addison-Wesley Series in Compuer

Science.

[SHAN 86] M.A. SHAND, "Hierarchical VLSI Artwork Analysis", VLSI'85, E. Hörbst Editor, Elsevier Science Publishers B.V. (North Holland), IFIP 86.

[SHI 86] H. SHIN, A. SANGIOVANNI-VICENTELLI, C. SEQUIN "Two-dimensional Compaction by Zone Refining", 23 rd Design Automation Conference, 1986.

[SIA 87] P. SIARRY, L. BERGONZI, G. DREYFUS "Thermodynamic Optimisation of Block Placement", IEEE Transaction on Computer-Aided Design, Vol. CAD-6, no. 2, March 87.

[SMI 85] P. SMITH; S. DANIEL "The VIVID System Approach to Technology Independence : The Master Technology File System", 22 nd DAC, 1985.

[STE 84] S. STEVENS, S. MCCABE "IDS - A System for fast Hierarchical Design of Handgrafted VLSI Circuits". ICCAD 84.

[TAR 75] R. E. TARJAN, "On the Efficiency of a good but no linear set merging algorithm" J. ACM 22:2, pp. 215-225.

[TAR 83] G.M. TAROLLI, W. HERMAN "Hierarchical Circuit Extraction with Detailed Capacitance", 20 th Design Automation Conference, 1983.

[TAY 84] G.S. TAYLOR, J. OUSTERHOUT "Magic's Incremental Design Rule Checker", 21 st Design Automation Conference,1984.

[TRIM 81] S. TRIMBERGER, J. ROWSON, C. LANG, J. GRAY, "A Structured Design Methodology and Associated Software Tools". IEEE Transactions on Circuits and Systems, Vol. CAS-28, No. 7, July .1981.

[TRO 81] H. TROPF, H. HERZOG "Multidimensional Range Search in Dynamically Balanced Trees", Angewandte Informatik 2/81.

[WAG 84] T. WAGNER "Hierarchical Layout Verifications", 21 st Design Automation Conference, 1984.

[WAN 84] H. WANATABE "IC Layout Generation Using Mathematical Optimization", 1984.

[WEB 79] W. Weber, "Threes types of map data structures, their ANDs and NOTs and a possible OR, Weber/1-Weber/1è, inInternational Advanced Studies Symposium on Topological Data Structures for Geographic Information Systems, Dedham, Mass., Oct. 16-2&, 1977, (G. Dutton, Ed), Harvard University La boratory for Computer Graphics and Spatial Analysis, 1978 / 1979.

[WEI 82] D. WEISE "Hierachical Analysis Tools in an Interactive Environment", MIT 1982.

[WEI 82] D. WEISE "Exploiting Hierarchy in the Analysis of VLSI systems", Master of Science, Massachusetts Institute of Technology, 1982.

[WES 81] N. WESTE "Virtual Grid Symbolic Layout", 18 th Design Automation Conference, 1981.

[WES 86] N. WESTE "Principles of CMOS VLSI Design, A System Perspective", reading, Serie Editor (north Holland) T. OHTSUKI, 1985, IFIP 86.

[WHI 81] T. WHITNEY "A Hierarchical Design Rule Checking Algorithm", Lambda, first quarter 1981.

[WHI 81] T. WHITNEY "A Hierarchical Design Rule Checker", T.R. 4320, California Institute of Technology, may 1981.

[WIL 78] P. WILCOX, H. ROMBEEK, D. M. CAUGHEY, "Design Verifications Bsed on One Dimensional Scans, Proc. 15th Design Automation Conférence, Las Vegas, Nevada, Junr 1978, pp. 285-289.

[WIL 78] Dan E. WILLARD "Balanced Forests of K-D Trees as Dynamic Data Structures", TR-23-78, Harvard University, 1978.

[WIL 78] D. E. WILLARD "Predicate-oriented database search algorithms, Harvard University, Cambridge, MA, Aiken Computation Laboratory, Ph D. Thesis, Report TR-20-78, 1978.

[WOL 83] W. WOLF "Two Dimensional Compaction Strategies", Ph.D. Thesis, dep. of Electrical Enginneering, Stanford University, March 1984.

[WON 85] Y. WONG "Hierarchical Circuit Verification", 22 nd Design Automation Conference, 1985.

[YAM 84] M. YAMASHITA, T. IBARAKI, and N. HONDA, "The minimum Cover Problem of a Rectilinear Region by Rectangles. Institute of Elecytonics and Communication Engineering of Japan, Report AL-84-16, pp. 13-24, 1984.

[YOS 77] YOSHIDA K., MITSUHASKI T., NAKADA Y., CHIBA T., OGITA K. and NAKATSUKA S., A Layout Checking System For Large Scale Integrated Circuits, Proc. 14th. Design Automation Conference, 1987.

TABLE DES MATIERES

UN SOUS-SYSTEME DE RECHERCHE GEOMETRIQUE ET D'EQUIVALENCE POUR LA CAO DE CIRCUITS INTEGRES VLSI

ABSTRACT

As the complexity of VLSI integrated circuit designs increases, the task of checking the mask descriptions for correctness becomes very time consuming. The wide acceptance of hierarchical mask descriptions (layouts) allows the development of faster hierarchical methods.

This thesis explores a new method for geometrical range searching based on quadtree hierarchy, and the hierarchical use of the Union-find equivalence algorithm, for searching and updating electrical equivalences. An improved version of the adaptative quadtree, the Mixed-Quadtree, has been developed and evaluated to optimize conventionnal tools such as Design Rule Checkers and Electrical Extractors, and supply fast editing of VLSI designs. This method gives even better results. Therefore, geometrical and electrical equivalence are searched by means of a range searching sub-system which is completely tested and evaluated. Hierarchical and geometrical algorithms for editing and verifying the VLSI layouts are also presented.

The hierarchical and geometrical methods which have been explored can be applied to a wide range of artwork analysis tasks such as circuit extraction, design rule checking, editing and compaction for real or symbolic mask descriptions. These methods are completely dynamic, and can therefore be used in an interactive mode, for on-line algorithms and general editing functions. Therefore these methods allow to edit flexible layouts. Hierarchical Design Rule Checker and Electrical Extractor are developped in order to evaluate performances of such algorithms for hierarchical analysis. Furthermore an overview of the problems which can be solved faster or in an interactive mode are also dicussed in the spectrum of CAD tools for VLSI designs.

INPG département des études doctorales

AUTORISATION de SOUTENANCE

VU les dispositions de l'article 15 Titre III de l'arrêté du 5 juillet 1984 relatif aux études doctorales

VU les rapports de présentation de Messieurs
. P. DE WILDE, Professeur
. R. GERBER, Professeur

Madame TOUSSAN Josette épouse BERGER

est autorisé(e) à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, spécialité "Informatique"

Fait à Grenoble, le 7 mars 1988

Georges LESPINARD
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,