Proof of correctness of the Mazoyer's solution of the firing squad problem in Coq

Jean Duprat

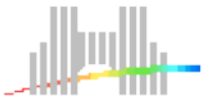
Laboratoire de l'Informatique du Parallélisme
Ecole Normale Supérieure de Lyon
Unité mixte de Recherche CNRS-INRIA-ENS LYON n° 5668

Proof of correctness of the Mazoyer's solution of the firing squad problem in Coq.

Jean DUPRAT

Mars 2002

Research Report N° 2002-14



Ecole Normale Supérieure de Lyon
46, Allée d'Italie, 69364 Lyon Cedex 07, France
Telephone : +33(0)4 72 72 80 37
Telecopieur : +33(0)4 72 72 80 80
Adresse electronique : lip@ens-lyon.fr

Proof of correctness of the Mazoyer's solution of the firing squad problem in Coq.

Jean Duprat
Mars 2002.

Abstract.

The firing squad synchronization is a cellular automata problem introduced by Moore, in 1987, J.Mazoyer gave a six state solution to this problem. The proof of correctness of this solution uses discrete geometrical considerations, but is quite hard to verify due to the multiplication of cases and indexes. To be more confident in the proof, a proof assistant developed by the Inria, Coq, has been used. This report exposes the development in Coq of the proof. A large use of inductive structures, a natural way in Coq, offers a clearer vision of the Mazoyer's solution. The full development of the proof is available in the contributions of the Coq software.

Keywords. :

Cellular automata, proof assistant, firing squad, Coq.

Résumé.

La synchronisation d'une ligne de fusiliers est un problème d'automates cellulaires posé par Moore, en 1987, J.Mazoyer donna une solution avec six états à ce problème. La preuve de correction de cette solution utilise des considérations de geometrie discrète, mais elle est assez difficile à vérifier à cause de la multiplication des cas et des indices. Pour être plus sur de cette preuve, un assistant à la preuve développé par l'Inria, Coq, a été utilisé. Ce rapport expose le développement en Coq de la preuve. Un grand usage des structures inductives, une voie naturelle en Coq, offre une vision plus claire de la solution de J.Mazoyer. Le développement complet de la preuve est disponible dans les contributions du logiciel Coq.

Mots clés. :

Automate cellulaire, assistant de preuve, ligne des fusiliers, Coq.

Introduction.

Nous assistons depuis quelques annees à l'apparition de plusieurs "theorems prover". En dehors des cas d'ecole que l'on trouve dans les "tutoriaux" de ces logiciels, la question de l'usage de ces logiciels par la communaute scientifique se pose.

Nous allons utiliser l'assistant de preuve Coq pour reecrire la preuve de correction d'une solution du probleme du "firing squad". Cette preuve, qui avait ete entierement ecrite "à la main" et qui etait reconnue comme assez complexe, a ete choisie parce qu'elle est entierement constructive.

Les conclusions tirees sont de deux ordres :

- quelles pistes suggerer aux concepteurs du logiciel pour en rendre l'usage plus courant ?
- quel interet l'utilisateur peut-il avoir à ecrire sa preuve avec Coq plutot que manuellement ?

Le problème.

Il s'agit d'un probleme d'automate cellulaire pose par Moore en 1962 [MOO]. On considere une ligne de N automates evoluant dans un temps discret. Chaque automate calcule son etat suivant à partir de son etat courant et de l'etat courant de ses deux voisins, les memes regles (particularisees toutefois pour les deux extremites) etant utilisees par tous les automates. On suppose que l'etat initial est un etat dit quiescent (blanc sur la figure) pour tous les automates sauf une extremite qui se trouve dans un etat different dit general (rouge). Le but est que tous les automates de la ligne passent au meme instant et pour la premiere fois dans un etat feu (noir).

Le temps minimum est 2N-1. Plusieurs solutions ont ete proposees et nous nous interesserons à celle de Jacques Mazoyer qui reussit à resoudre ce probleme avec six etats seulement [MAZ]. C'est la preuve de correction de la solution decrite dans cette publication qui a ete reprise en Coq. Les systemes d'indices ont ete modifies pour favoriser l'addition par rapport à la soustraction et pour commencer les enumerations à zero. Par contre les differents objets definis dans la preuve de J.Mazoyer se retrouvent ci-dessous avec le meme nom pour faciliter une lecture en regard de la preuve manuscrite et de la preuve Coq.

La figure ci-contre represente l'evolution dans le temps d'une ligne de 29 cellules.

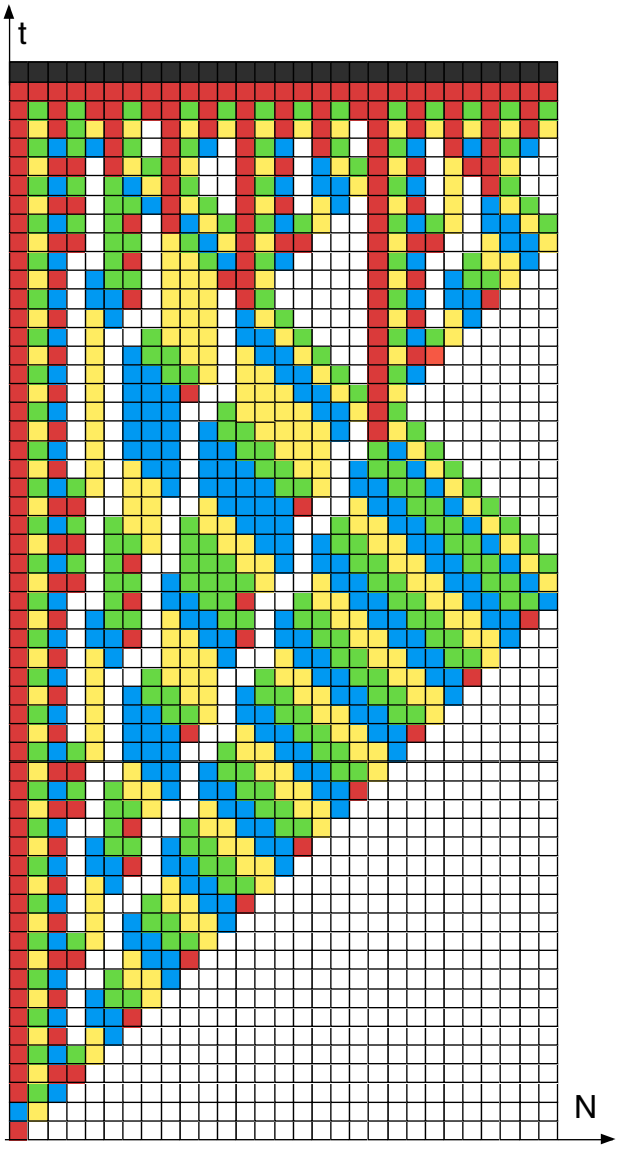


Figure 1 : solution du firing squad.

L'assistant de preuves Coq.

Base sur le calcul des constructions de T.Coquand, l'assistant de preuves Coq est le fruit du travail d'une equipe de l'INRIA autour de T.Coquand, G.Huet et C.Paulin [CO&]. Il est disponible gratuitement et evolue regulierement. La version utilisee ici etait la version 5.10 pour MacIntosh. La preuve est disponible dans les contributions.

Coq permet de definir des objets à partir des constructeurs de base et des definitions precedentes. Il permet d'etablir des lemmes et des theoremes en construisant l'arbre de preuves à partir de tactiques. En ce sens, il s'agit bien d'un assistant de preuves, car l'utilisateur guide le logiciel. Un demonstrateur ferait la recherche automatiquement. Ce point est l'objet d'un commentaire en conclusion 2.

Outre son moteur, Coq offre des bibliotheques evitant d'avoir à reconstruire les bases des mathematiques : arithmetique, logique, theorie des ensembles, …. La seule bibliotheque utilisee pour cette preuve est la bibliotheque "arith" qui contient les definitions et proprietes usuelles des operations sur les naturels.

Enfin une preuve se structure en sections, permettant à la fois une structuration logique et une meilleure efficacite. Il est possible d'avoir des variables et des lemmes locaux aux sections qui ne sont pas visibles de l'exterieur. De plus, sauf directive contraire, les theoremes seront perçus de l'exterieur uniquement par leur enonce sans leur terme de preuve, ce qui allege l'environnement.

La preuve.

Cette preuve est bâtie sur les figures geometriques planes obtenues par le developpement de l'automate dans l'espace (une dimension) et le temps (la seconde dimension).

Elle est structuree en 12 sections; le graphe des dependances de ces sections, les unes par rapport aux autres, est dessine ci-contre. Ces sections traduisent une conception descendante de construction de la preuve, des structures les plus elementaires aux structures les plus complexes et des theoremes les plus generaux aux theoremes les plus specifiques au probleme.

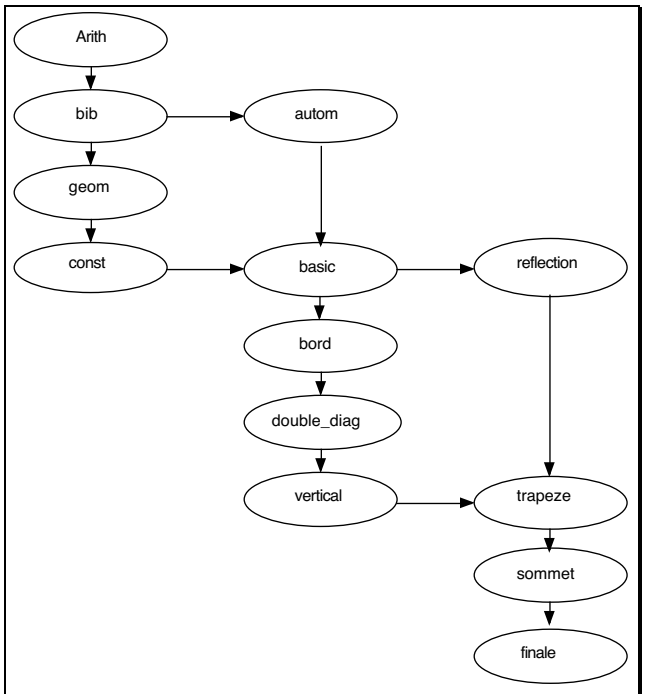


Figure 2 : graphe de dependances.

La bibliothèque.

Cette section contient tout ce qui peut être considere comme n'etant pas specifique au probleme mais comme faisant partie du bagage mathematique de base.

Ces proprietes peuvent devenir inutiles dans les versions suivantes du logiciel. (Il arrive meme que ces proprietes soient deja presentes dans d'autres contributions mais elles ont alors ete rangees ici sous un nom coherent avec l'ensemble des noms choisis par l'auteur. Le problème du nom des proprietes est d'ailleurs tres lie à ce que l'auteur considere comme mnemotechnique ce qui n'est pas uniforme).

Je conseillerai volontiers à celui qui debute un travail important en Coq de se faire sa bibliotheque, cela permet d'alleger les demonstrations, d'eviter que des petits lemmes triviaux encombrent les sections et se dupliquent dans plusieurs sections, enfin cela facilite le travail de developpement.

Dans cette bibliotheque, on trouvera successivement :

- des règles de démonstration de propriétés dépendant les unes des autres. Par exemple :

```
Theorem Rec4 : (A,B,C,D,E:Prop)
   (A->B->C->D->E)->(A->B->(B->C)->(C->D)->E).

Intuition.

Save.
```

- les entiers naturels de un à neuf, leur ajout ou retranchement. Par exemple :

```
Definition trois := (S (S (S O))).

Lemma plus_trois :
   (n:nat) (plus n trois)=(S (S (S n))).

Intros; Unfold trois; Repeat Rewrite <- plus_n_Sm; Auto.

Save.
```

- la multiplication par deux (double) et trois (triple) avec quelques propriétés telles que :

```
Definition double := [p:nat] (plus p p).

Definition triple := [p:nat] (plus (plus p p) p).

Lemma le_double_triple :
   (n:nat) (le (double n) (triple n)).

(Intro; Unfold double triple; Apply le_plus_l).

Save.
```

- la division par deux et la parité.

```
Mutual Inductive even : nat -> Prop :=
   even_O : (even O)
   | even_S : (n:nat) (odd n) -> (even (S n))
with odd : nat -> Prop :=
odd_S : (n : nat) (even n) -> (odd (S n)).
```

- la division par trois et les restes modulo 3 avec plusieurs propriétés relatives à la division entière par trois.

```
Inductive div3 [a:nat] : Set  :=
reste_0 : (q:nat) (a=(triple q))->(div3 a)
| reste_1 : (q:nat) (a=(S (triple q)))->(div3 a)
| reste_2 : (q:nat) (a=(S (S (triple q))))->(div3 a).


Theorem quotient3 : (n:nat) (div3 n).

Induction n.
(Apply reste_0 with q:=O; Auto).
(Clear  n; Intros; Elim H; Intros).
(Apply reste_1 with q:=q; Auto).
(Apply reste_2 with q:=q; Auto).
(Apply reste_0 with q:=(S q); Auto).
(Rewrite -> triple_S; Rewrite -> e; Auto).

Defined.

Lemma plus_deuxtiers_untiers :
(n:nat) (Omod3 n) ->
   (plus (double (tiers n)) (tiers n))=n.

(Intros; Unfold double ; Apply triple_tiers; Auto).

Save.
```

- quelques bricoles inclassables (pourquoi y en a-t-il toujours ?).

Remarque :

Dans Coq, les naturels sont bâtis à l'aide des deux constructeurs O (zéro) et S (successeur). Les opérations "plus, minus, le, lt, ge et gt" sont definies dans la bibliotheque, ces operations utilisant une notation prefixe. Dans la suite, pour faciliter le travail de lecteurs pour lesquels ces notations sont peu familieres, toutes les expressions entieres seront ecrites entre parentheses avec la notation infixe usuelle et les constantes sous leur ecriture chiffree.

Géométrie.

Comme le montre la figure de l'évolution temporelle, on peut ramener le problème à un problème de géométrie discrete à deux dimensions l'espace et le temps. On repere donc chaque case par son couple de coordonnees entieres (t,x) le temps et l'abscisse.

On va étudier les proprietés verifiées sur toutes les cases de figures geométriques.

```
Definition Local_Prop := nat->nat->Prop.
```

Ainsi, on définit une ligne horizontale (temps constant) par :

```
Inductive Horizontale
   [t, x, long :nat;
   P : Local_Prop]  : Prop :=
make_horizontale :
   ((dx :nat) (dx≤long) ->
   (P t (x+dx))) ->
(Horizontale t x long P).
```

P	P	P	P	P	P	P	P

Figure 3 : Horizontale.

(les paramètres définissent une ligne de (long+1) cases des coordonnées (t,x) aux coordonnees (t,x+long)).

On a besoin de savoir construire une horizontale en mettant bout à bout deux horizontales :

```
Lemma hh_hor : (t,x,cote,cote':nat) (P:Local_Prop)
(Horizontale t x cote P) ->
(Horizontale t (x+cote+1) cote' P) ->
(Horizontale t x (cote+cote'+1) P).
```

ce qui se démontre en utilisant le fait que x≤cote+1+cote' se décompose en deux cas :

- x≤cote
- cote+1≤x≤cote+1+cote'.

Il est pratique de disposer de la démonstration directe pour des petites horizontales de taille 1, 2, 3 ou 4, par exemple :

```
Lemma hor_trois : (t, x :nat) (P : Local_Prop)
(P t x) -> (P t (x+1)) -> (P t (x+2)) -> (P t (x+3)) ->
(Horizontale t x (3) P).
```

Des lignes horizontales dont la, respectivement les deux, premiere(s) case(s) verifie une propriété différente seront egalement utilisees.

```
Inductive Horizontale_t0
[t, x, long :nat; P0, P :
   Local_Prop]  : Prop :=
make_horizontale_t0 :
(P0 t x) ->
(Horizontale t (x+1) long P) ->
(Horizontale_t0 t x long P0 P).
```

P0	P	P	P	P	P	P	P

Figure 4 : Horizontale_t0.

```
Inductive Horizontale_t1
[t, x, long :nat;
P0,P1, P : Local_Prop]  : Prop
:= make_horizontale_t1 :
(P0 t x) ->
(P1 t (x+1)) ->
(Horizontale t (x+2) long P) ->
(Horizontale_t1 t x long P0 P1 P).
```

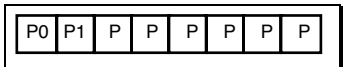


Figure 5 : Horizontale_t1.

De même, on définit une ligne verticale :

```
Inductive Verticale
[t, x, haut :nat; P : Local_Prop]  : Prop
:= make_verticale :
((dt :nat) (dt≤haut) -> (P (t+dt) x)) ->
(Verticale t x haut P).
```

Il est utile de montrer qu'un segment inclu dans une verticale de propriété P est une verticale de propriété P :

```
Lemma inclus_vert :
   (t, t', x, haut, haut' :nat)
   (P : Local_Prop)
(t≤t') ->
((t'+haut')≤(t+haut)) ->
(Verticale t x haut P) ->
(Verticale t' x haut' P).
```

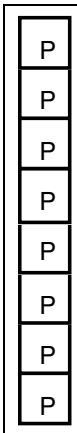


Figure 6 : Verticale.

On construira une verticale par aboutement de deux verticales :

```
Lemma vv_vert :
   (t, x, haut, haut' :nat) (P : Local_Prop)
(Verticale t x haut P) ->
(Verticale (t+haut+1) x haut' P) ->
(Verticale t x (haut+haut'+1) P).
```

mais il arrivera également que l'on construise une verticale de 2p cases par aboutement de p verticales de 2 cases :

```
Lemma rec_vert : (t, x, haut :nat) (P : Local_Prop)
((dt :nat) (dt≤haut) ->
(P (t+2.dt) x) /\ (P (t+2.dt+1) x)) ->
(Verticale t x (2.haut+1) P).
```

Enfin les petites verticales de hauteur 1, 2 ou 3 auront une définition directe telle que :

```
Lemma vert_deux : (t, x :nat) (P : Local_Prop)
(P t x) -> (P (t+1) x) -> (P (t+2) x) ->
(Verticale t x (2) P).
```

On définit le triangle rectangle isocèle :

```
Inductive Triangle_inf
[t, x, cote :nat; P : Local_Prop]
: Prop := make_triangle_inf :
((dt, dx :nat)
   (dx≤cote) -> (dt≤dx) ->
   (P (t+dt) (x+dx))) ->
(Triangle_inf t x cote P).
```

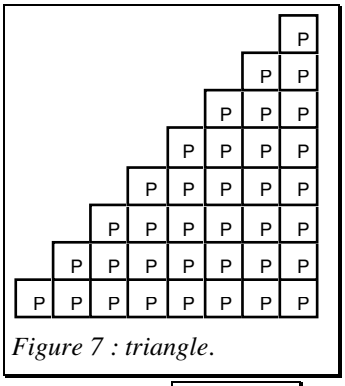


Figure 7 : triangle.

ainsi que la méthode permettant de l'obtenir par sa base et la règle de construction qui rajoute une case de propriete P au-dessus de 2 cases de propriete P.

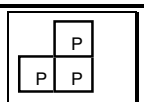


Figure 8 : règle.

```
Lemma rec_triangle_inf :
(t, x, cote :nat) (P : Local_Prop)
(Horizontale t x cote P)->
((t',x':nat) (P t' x')->(P t' (x'+1))->
                        (P (t'+1) (x'+1)))->
(Triangle_inf t x cote P).
```

Enfin on se servira de diagonales caracterisées par leurs extrémités :

```
Inductive Diag
   [t, x, cote :nat;
   P, Q, R : Local_Prop]  :
Prop := make_diag :
(1<cote) ->
(P t (x+cote)) ->
((dt,dx:nat)
   (0<dt) -> (0<dx) ->
   ((dt+dx)=cote)->
   (Q (t+dt) (x+dx))) ->
(R (t+cote) x) ->
(Diag t x cote P Q R).
```

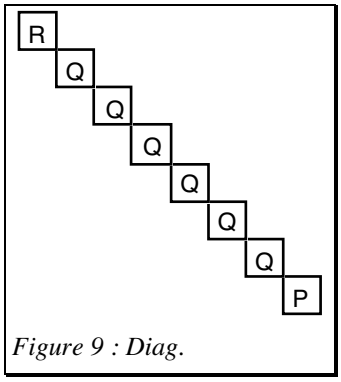


Figure 9 : Diag.

```
Inductive Diag'
   [t, x, cote :nat;
   P, Q', Q, R : Local_Prop]  :
Prop := make_diag' :
(2<cote) ->
(P t (x+cote)) ->
((dx:nat)
   ((dx+1)=cote)->
   (Q' (t+1) (x+dx)))->
((dt,dx:nat) (1<dt) ->
   (0<dx) -> ((dt+dx)=cote)->
   (Q (t+dt) (x+dx))) ->
(R (t+cote) x) ->
(Diag' t  x cote P Q' Q R).
```

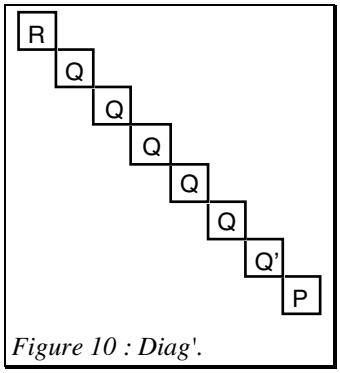


Figure 10 : Diag'.

```
Inductive Semi_Diag
   [t, x, cote :nat;
   P, Q : Local_Prop]  : Prop := make_semidiag :
(0<cote) ->
(P t (x+cote)) ->
((dt,dx:nat) (0<dt) ->
((dt+dx)=cote)->
   (Q (t+dt) (x+dx))) ->
(Semi_Diag t x cote P Q).
```

Il est intéressant de se donner une méthode de preuve d'une diagonale (ou d'une diag' ou d'une semi-diag) en procedant de bas en haut et de droite à gauche. La recurrence est particuliere puisqu'elle concerne l'interieur de la diagonale.

```
Lemma Rec_Diag :
(t, x, cote :nat) (P, Q, R : Local_Prop)
(1<cote) ->
(P t (x+cote))->
((dx:nat) ((dx+2)=cote)->(P t (x+cote))->
   (Q (t+1) (x+dx+1)))->
((dt,dx:nat) (0<dt)-> (0<dx)->((dt+dx+2)=cote)->
   (Q (t+dt) (x+dx+2))->(Q (t+dt+1) (x+dx+1)))->
((dt:nat) ((dt+2)=cote)->(Q (t+dt) (x+2))->
   (Q (t+dt+1) (x+1)))->
((dt:nat) ((dt+1)=cote)->(Q (t+dt) (x+1))->
   (R (t+cote) x))->
(Diag t x cote P Q R).
```

L'énoncé de ce lemme met en évidence l'interêt du système de coordonnées adopté, avec les coordonnées absolues (t,x) de la case inferieure gauche du rectangle (origine) englobant la figure, et les coordonnees relatives (dt,dx) de chaque case de la figure par rapport à cette case origine. Ainsi, tout s'exprime en termes d'addition et d'inegalites.

Les constructions.

Du fait de la règle de calcul de l'état suivant d'une cellule à partir de l'état courant de la cellule et de ses deux voisines, le calcul des etats d'une diagonale va se faire à partir des deux diagonales precedentes. Cette section traite toutes les variantes de ces constructions utiles par la suite (selon qu'il s'agit de "diag", de "diag' " ou de "semi-diag", que l'empilement a ou non un decalage dans l'espace).

Cette section commence par lister toutes les formes de calcul sur les coordonnees que peut prendre un pas elementaire. En voici les deux premiers de la serie de neuf existants dans cette section :

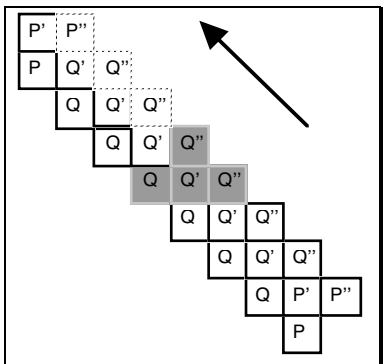


Figure 11 : principe de construction.

```
Lemma Pas_hh: (t,x,dt,dx:nat)
(loi P Q R T)->
(P ((t+(dt+2)) (x+dx))->
(Q ((t+1)+(dt+1)) (x+(dx+1))->
(R ((t+2)+dt) (x+(dx+2)))->
(T ((t+2)+(dt+1)) (x+(dx+1))).

(Intros t x dt dx; Repeat Rewrite <- plus_n_Sm; Simpl;
Intros; Auto).

Save.
```

```
Lemma Pas_hd : (t,x,dt,dx:nat)
(loi P Q R T)->
(P (t+(dt+1)) (x+dx))->
(Q ((t+1) dt) (x+(dx+1)))->
(R ((t+1)+dt) ((x+1)+(dx+1)))->
(T ((t+1)+(dt+1)) ((x+1)+dx)).

(Intros t x dt dx; Repeat Rewrite <- plus_n_Sm; Simpl;
Intros; Auto).

Save.
```

Le premier correspond en empilement vertical, les trois diagonales ont comme origine dans le temps t, t+1 et t+2. Le second correspond à la figure avec un decalage spatial de la troisieme diagonale, les coordonnees des origines des diagonales sont successivement (t,x) (t+1,x) (t+1,x+1). Le but de tous ces lemmes, dont la preuve est tres facile, est de simplifier les unifications dans la suite en faisant toujours le bon calcul sur les coordonnees.

La deuxième partie de cette section réalise les empilements. On a successivement

• 3 diagonales "diag" alignées verticalement,

```
Lemma DDD :
(Diag t x cote P Q P)->
(Diag (t+1) x cote P' Q' P')->
(P'' (t+2) (x+cote))->
(Diag (t+2) x cote P'' Q'' P'').
```

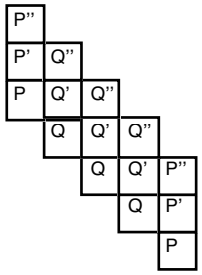


Figure 12 : DDD.

• 1 diagonale "diag' " et 2 diagonales "diag" alignées verticalement,

```
Lemma D'DD :
(Diag'  t x cote P R Q P)->
(Diag (t+1) x cote P' Q' P')->
(P'' (t+2) (x+cote))->
(Diag (t+2) x cote P'' Q'' P'').
```

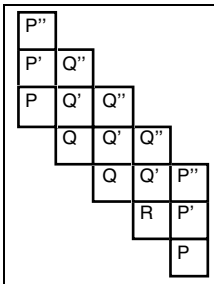


Figure 13 : DD'D.

• 1diagonale "diag' ", 1 diagonale "diag" et 1 diagonale "diag' " alignées verticalement,

```
Lemma D'DD' :
(Diag' t x cote P R Q P)->
(Diag (t+1) x cote P' Q' P')->
(P'' (t+2) (x+cote))->
(Diag' (t+2) x cote P'' R'' Q'' P'').
```

• 1diagonale "diag", 1 diagonale "diag' " et 1 diagonale "diag" alignées verticalement,

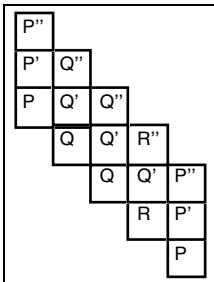


Figure 14 : D'DD'.

```
Lemma DD'D :
(Diag  t x cote P Q P)->
(Diag' (t+1) x cote P' R' Q' P')->
(P'' (t+2) (x+cote))->
(Diag (t+2) x cote P'' Q'' P'').
```

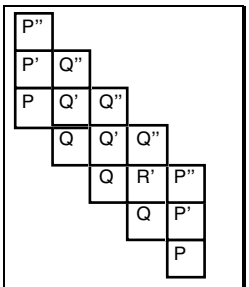


Figure 15 : DD'D.

• 2 diagonales "diag" alignées verticalement et 1 diagonale "diag' " décalée,

```
Lemma DD_D':
(2<cote)->
(Diag t x cote P Q P)->
(Diag (t+1) x cote P' Q' P')->
(P'' (t+1) ((x+1)+cote))->
(Diag' (t+1) (x+1) cote
                    P'' R'' Q'' P'').
```

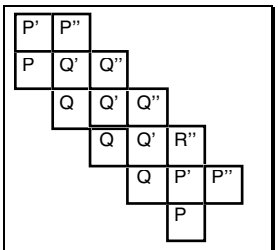


Figure 16 : DD_D'.

• 1diagonale "diag" et, décalées, 1 diagonale "diag' " et 1 diagonale "diag" alignées verticalement,

```
Lemma D_D'D :
(Diag t x cote P Q P)->
(Diag' t (x+1) cote P' R' Q' P')->
(P'' (t+1) ((x+1)+cote))->
(Diag (t+1) (x+1) cote P'' Q'' P'').
```

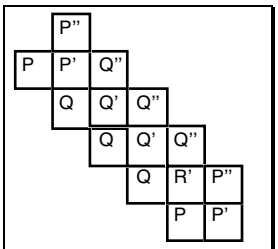


Figure 17 : D_D'D.

• 2diagonales "diag" alignées verticalement et 1 diagonale "diag" décalée et allongée,

```
Lemma DD_D$ :
(Diag t x cote P Q P)->
(Diag (t+1) x cote P' Q' P')->
(P'' (t+1) (x+(cote+1)))->
(Diag (t+1) x (cote+1) P'' Q'' P'').
```

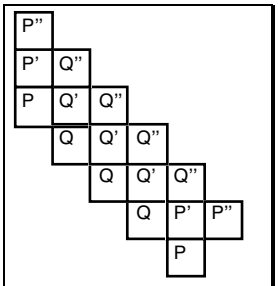


Figure 18 : DD_D$.

• 1 diagonale "diag" et, décalées, 2 diagonales "diag" alignées et allongées,

```
Lemma D_DD$ :
(Diag t x cote P Q P)->
(Diag t x (cote+1) P' Q' P')->
(P'' (t+1) (x+(cote+1)))->
(Diag (t+1) x (cote+1) P'' Q'' P'').
```

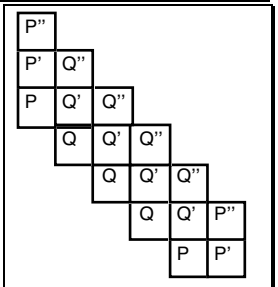


Figure 19 : D_DD$.

• 2 diagonales "diag" alignées verticalement et 1 diagonale "diag" décalée,

```
Lemma DD_D :
(2<cote)->
(Diag t x cote P Q P)->
(Diag (t+1) x cote P' Q' P')->
(P'' (t+1) ((x+1)+cote))->
(Diag (t+1) (x+1) cote P'' Q'' P'').
```

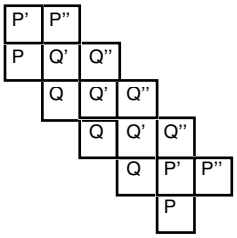


Figure 20 : DD_D.

• 1diagonale "diag' " et 1 diagonale "diag" alignées verticalement, et 1 diagonale "diag" decalee,

```
Lemma D'D_D :
(Diag' t x cote P R Q P)->
(Diag (t+1) x cote P' Q' P')->
(P'' (t+1) ((x+1)+cote))->
(Diag (t+1) (x+1) cote P'' Q'' P'').
```

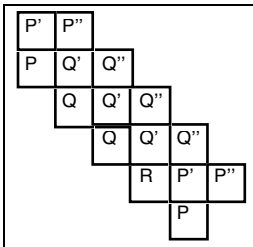


Figure 21 : D'D_D.

• 1 diagonale "diag" et, décalées, 2 diagonales "diag" alignées verticalement,

```
Lemma D_DD :
(Diag t x cote P Q P)->
(Diag t (x+1) cote P' Q' P')->
(P'' (t+1) ((x+1)+cote))->
(Diag (t+1) (x+1) cote P'' Q'' P'').
```

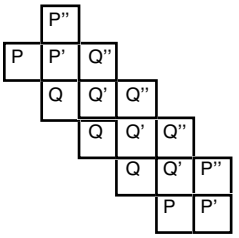


Figure 22 : D_DD.

• 2diagonales "diag" alignées verticalement et 1 diagonale "diag" décalée et raccourcie,

```
Lemma DD$_D :
(1<cote)->
(Diag t x (cote+1) P Q P)->
(Diag (t+1) x (cote+1) P' Q' P')->
(P'' (t+2) (x+cote+))->
(Diag (t+1) (x+1) cote P'' Q'' R'').
```

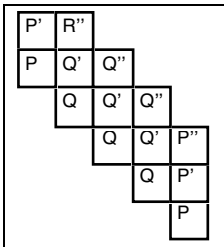


Figure 23 : DD$_D.

• 2diagonales "diag" alignées verticalement et 1 diagonale "semi-diag" décalée et raccourcie,

```
Lemma DD_d :
(0<cote)->
(Diag t x (cote+2) P Q R)->
(Diag (t+1) (x+1) (cote+1) P' Q' R')->
(P'' (t+2) (x+cote+2))->
(Semi_Diag (t+2) (x+2) cote P'' Q'').
```

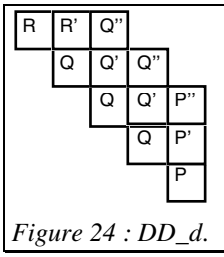


Figure 24 : DD_d.

• 1 diagonale "diag" et 1 "semi-diag" alignées verticalement et 1 diagonale "semi-diag" decalee et raccourcie,

```
Lemma Dd_d :
(0<cote)->
(Diag t x (cote+2) P Q R)->
(Semi_Diag (t+1) (x+1) (cote+1) P' Q')->
(P'' (t+2) (x+cote+2))->
(Semi_Diag (t+2) (x+2) cote P'' Q'').
```

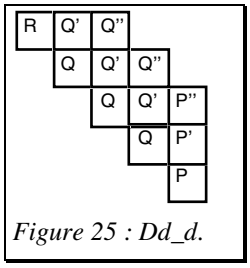


Figure 25 : Dd_d.

• 3 diagonales "semi-diag" décalées et raccourcies

```
Lemma dd_d :
(0<cote)->
(Semi_Diag t x (cote+2) P Q)->
(Semi_Diag (t+1) (x+1) (cote+1) P' Q')->
(P'' (t+2) (x+cote+2))->
(Semi_Diag (t+2) (x+2) cote P'' Q'').
```

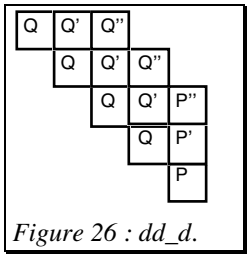


Figure 26 : dd_d.

Pour être démontrés, tous ces lemmes ont besoin d'hypothèses supplémentaires sur les lois régissant les transitions. Afin de simplifier l'écriture et la lisibilité, ces hypothèses ont été regroupées en tête de section (et non répétées dans l'énoncé de chaque théorème).

```
Variables
  P, Q, R,  P', Q', R',  P'', Q'', R''  : Local_Prop.
Hypothesis PPQP : (loi P P' Q'' P'').
Hypothesis PQPQ : (loi P Q' P'' Q'').
Hypothesis PQQP : (loi P Q' Q'' P'').
Hypothesis PQQQ : (loi P Q' Q'' Q'').
Hypothesis PQRQ : (loi P Q' R'' Q'').
Hypothesis XPQP : (loi_droite P' Q'' P'').
Hypothesis QPPQ : (loi Q P' P'' Q'').
Hypothesis QPPR : (loi Q P' P'' R'').
Hypothesis QQPQ : (loi Q Q' P'' Q'').
Hypothesis QQPR : (loi Q Q' P'' R'').
Hypothesis QQQQ : (loi Q Q' Q'' Q'').
Hypothesis QQRQ : (loi Q Q' R'' Q'').
Hypothesis QRPQ : (loi Q R' P'' Q'').
Hypothesis RPPQ : (loi R P' P'' Q'').
Hypothesis PQQR : (loi P Q' Q'' R'').
```

Coq garantit, une fois la section fermée, l'énoncé minimal de chaque lemme n'ayant que les hypothèses nécessaires à sa propre preuve.

Cette section contient essentiellement la fonction de transition.

On définit d'abord l'ensemble des six états :

```
Inductive Set Couleur :=
   A : Couleur | B : Couleur | C : Couleur | L : Couleur
   | G : Couleur | F : Couleur.
```

La fonction de transition, d'un ensemble fini (Couleur X Couleur X Couleur) dans un ensemble fini (Couleur) est décrite en extension par des tables. La premiere caracterise l'etat suivant en fonction de l'etat courant :

```
Definition Transition := [c0, c1, c2 : Couleur]
<Couleur> Case c1 of
   (*A*) (Transition_A c0 c2)
   (*B*) (Transition_B c0 c2)
   (*C*) (Transition_C c0 c2)
   (*L*) (Transition_L c0 c2)
   (*G*) (Transition_G c0 c2)
   (*F*) F
end.
```

On note qu'à l'exception de l'état final (stable), les autres états dépendent tous des voisins, ce qui donne cinq nouvelles tables :

```
Definition Transition_A := [c0, c2 : Couleur] <Couleur>
Case c0 of
   (*A*) (Transition_A_A c2)
   (*B*) (Transition_B_A c2)
   (*C*) A
   (*L*) (Transition_L_A c2)
   (*G*) C
   (*F*) F
end.

Definition Transition_B := [c0, c2 : Couleur] <Couleur>
Case c0 of
   (*A*) (Transition_A_B c2)
   (*B*) (Transition_B_B c2)
   (*C*) (Transition_C_B c2)
   (*L*) (Transition_L_B c2)
   (*G*) (Transition_G_B c2)
   (*F*) F
end.

Definition Transition_C := [c0, c2 : Couleur] <Couleur>
Case c0 of
   (*A*) B
   (*B*) (Transition_B_C c2)
   (*C*) (Transition_C_C c2)
   (*L*) (Transition_L_C c2)
   (*G*) B
   (*F*) F
end.
```

```
Definition Transition_L := [c0, c2 : Couleur] <Couleur>
Case c0 of
   (*A*) (Transition_A_L c2)
   (*B*) L
   (*C*) (Transition_C_L c2)
   (*L*) L
   (*G*) (Transition_G_L c2)
   (*F*) L
end.

Definition Transition_G := [c0, c2 : Couleur] <Couleur>
Case c2 of
   (*A*) G
   (*B*) G
   (*C*) G
   (*L*) (Transition__G_L c0)
   (*G*) (Transition__G_G c0)
   (*F*) G
end.
```

On note que dans certains cas, la donnée de l'état courant et de celle d'un voisin suffit à déterminer l'état suivant et que dans d'autres cas, il faut faire appel à une troisieme table comme celles-ci (pour l'etat courant A) :

```
Definition Transition_A_A := [c2 : Couleur] <Couleur>
Case c2 of
   (*A*) A
   (*B*) B
   (*C*) C
   (*L*) A
   (*G*) B
   (*F*) F
end.

Definition Transition_B_A := [c2 : Couleur] <Couleur>
Case c2 of
   (*A*) F
   (*B*) G
   (*C*) C
   (*L*) G
   (*G*) C
   (*F*) F
end.

Definition Transition_L_A := [c2 : Couleur] <Couleur>
Case c2 of
   (*A*) A
   (*B*) L
   (*C*) G
   (*L*) A
   (*G*) F
   (*F*) F
end.
```

On remarque également que la détermination se fait toujours en premier sur le voisin de gauche sauf pour l'etat general qui privilegie l'etat de droite. Ce choix a ete fait au vu des tables de transitions pour alleger l'ecriture des tables. Il a eu une consequence benefique dans la suite car il s'est trouve parfois d'avoir à demontrer qu'un etat suivant etait obtenu dans une configuration ou l'etat courant et l'etat du seul voisin (de droite pour un etat G, de gauche pour un etat L) etaient connus. La preuve, quel que soit l'etat de l'autre voisin etait obtenue sans etude de cas et de ce fait plus rapidement.

Il faut noter ici une différence avec la preuve papier de J.Mazoyer. Sur le papier, la définition de la fonction de transition est partielle, de nombreuses configurations, reputees non accessibles, sont laissees indefinies. Coq etant base sur le calcul des constructions n'admet pas de fonctions partielles. Il aurait bien sur ete possible de simuler un etat inaccessible par une septieme valeur d'etat et la preuve que cette valeur n'est jamais

prise. Toutefois, la preuve aurait été considérablement alourdie. De plus, l'intérêt d'une telle manipulation dans le cas présent était faible, ce que nous voulions montrer était la correction d'une solution. C'est la raison pour laquelle il a été choisi de remplir les cases indéfinies des tables de la solution de J.Mazoyer par les états qui rendaient l'écriture de la fonction de transition la plus facile. Les puristes ne pourront nier que la solution ainsi décrite est correcte mais ils pourront toujours objecter qu'en remplissant ainsi les vides, nous avons peut-être par un hasard aussi fortuit qu'heureux rempli une case utile avec la bonne valeur…

La section contient également la fonction calculant toute transition.

```
Parameter N : nat.

Axiom necessaire : (2<N).

Fixpoint Etat [t : nat] :  nat -> Couleur :=
[x : nat] <Couleur> Case t of
(*O*)<Couleur> Case x of
   (*O*) G
   (*Sx'*)[x' : nat]
         (Ifdec (eq_nat_dec (N+1) (x'+1)) G
         (Ifdec (eq_nat_dec (N+2) (x'+1)) C L))
   end
(*St'*)[t' : nat] <Couleur> Case x of
   (*O*)(Transition L (Etat t' O) (Etat t' 1))
   (*Sx'*)[x' : nat]
         (Transition (Etat t' x') (Etat t' (x'+1))
                          (Etat t' (x'+2)))
   end
end.
```

Le premier cas (t=0) correspond à l'état initial de l'automate et le second (t=t'+1) à l'état courant. Le constructeur Ifdec fonctionne à la manière d'un "si alors sinon" qui se détermine sur la valeur de décision de l'égalité des naturels "eq_nat_dec".

La base (t=0) est donc une ligne infinie d'automates à l'état quiescent L à l'exception des cases d'abscisses 0, N+1 et N+2 respectivement à l'état G, G et C.

```
Lemma G0O : (Etat O O)=G.

Lemma G0N : (Etat O (N+1))=G.

Lemma C0N1 : (Etat O (N+2))=C.

Lemma base_L :
   (x:nat) (0<x) -> (x<(N+1)) -> (Etat O x)=L.

Lemma base$_L : (x:nat) ((N+2)<x) -> (Etat O x)=L.
```

Plusieurs choix ont été faits ici.

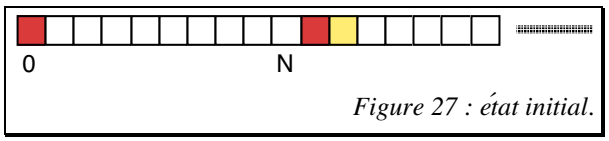


Figure 27 : état initial.

• Le premier est de ne borner ni le temps ni l'espace. Cela permet d'utiliser les naturels pour travailler et non les naturels d'un intervalle sur lesquels il aurait fallu montrer les propriétés usuelles. Les fonctions de Coq ne pouvant être partielles, il fallait donc définir un comportement pour toutes les cellules (fictives) au-delà de la nième (et dernière cellule de la ligne des fusiliers).

• Plutôt que d'avoir un état X (comme le propose J.Mazoyer) pour caractériser l'extérieur, on a cherché à faire en sorte que le cas N s'insère dans la récursivité. Les autres rectangles "similaires" (au sens de la récursivité) et plus petits apparaissant dans le rectangle sont caractérisés par des frontières verticales formées d'états différents de G à gauche et égaux à G à droite. Il a donc été décidé de faire comme si un état d'abscisse -1 était à L et de faire en sorte que les règles de calcul génèrent un état G pour l'automate d'abscisse N+1. C'est la raison d'être de l'état C en N+2 à l'origine (d'autres façons d'imposer cette verticale d'état G auraient pu être choisie).

• Enfin la condition N>2 s'est imposée pour être toujours dans le cas général. Les démonstrations pour les valeurs 0, 1 et 2 auraient pu être faites de manière à avoir une preuve pour tout N, elles auraient surchargé la preuve sans ajouter quoi que ce soit de vraiment intéressant.

Cette section commence par définir comme propriété locale, le fait pour une case d'être dans un état donné.

```
Definition A_Etat    :=[t,x:nat] (Etat t x)=A.
Definition B_Etat    :=[t,x:nat] (Etat t x)=B.
Definition C_Etat    :=[t,x:nat] (Etat t x)=C.
Definition G_Etat    :=[t,x:nat] (Etat t x)=G.
Definition L_Etat    :=[t,x:nat] (Etat t x)=L.
Definition F_Etat    :=[t,x:nat] (Etat t x)=F.
```

Ceci permet de définir les structures appelées A_basic, B_basic et C_basic dans l'article de J.Mazoyer.

```
Inductive A_basic [t,x,cote:nat] :
Prop := make_A_basic :
(2<cote)->
(Diag t x cote L_Etat A_Etat L_Etat)->
(Diag (t+1) x cote L_Etat A_Etat
                          L_Etat)->
(A_basic t x cote).
```

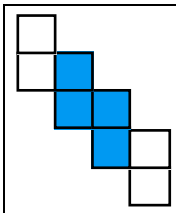


Figure 28 : A_basic.

```
Inductive B_basic [t,x, cote:nat] :
Prop := make_B_basic :
(2<cote)->
(Diag' t x cote L_Etat G_Etat B_Etat
                          L_Etat)->
(Diag (t+1) x cote L_Etat B_Etat
                          L_Etat)->
(B_basic t x cote).
```

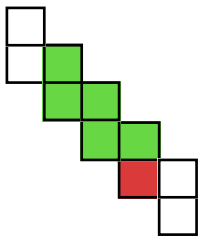


Figure 29 : B_basic.

```
Inductive C_basic [t,x, cote:nat] :
Prop := make_C_basic :
(1<cote)->
(Diag t x cote L_Etat C_Etat L_Etat)->
(Diag (t+1) x cote L_Etat C_Etat
                          L_Etat)->
(C_basic t x cote).
```

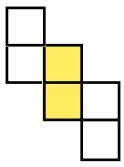


Figure 30 : C_basic.

Les figures 28 et 30 représentant les structures A_basic et C_basic sont de longueur minimum, bien entendu, ces structures peuvent être beaucoup plus longues. Le fait de choisir les longueurs minimum différentes a permis de simplifier la démonstration en gérant ces longueurs au plus juste. Il faut noter le confort d'utilisation de Coq pour gérer ce genre de paramètres : lorsque la démonstration bute sur une condition de longueur indémontrable, cela signifie qu'il est nécessaire de rajouter une hypothèse dans le lemme en cours.

Le propre de ces figures de bases est de s'empiler de façon régulière :

• soit 2 figures identiques alignées verticalement :

```
Lemma A_A :
(A_basic t x cote)->
(L_Etat (t+2) (x+cote))->
(L_Etat (t+3) (x+cote))->
(A_basic (t+2) x cote).
```

ce lemme se démontre en appliquant deux fois le lemme DDD;

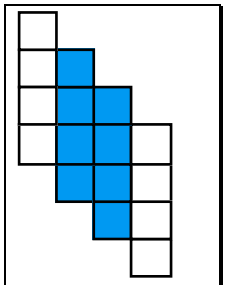


Figure 31 : A_A.

```
Lemma B_B :
(B_basic t x cote)->
(L_Etat (t+2) (x+cote))->
(L_Etat (t+3) (x+cote))->
(B_basic (t+2) x cote).
```

celui-ci se démontre en appliquant d'abord D'DD' puis DD'D;

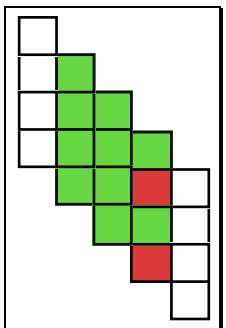


Figure 32 : B_B.

```
Lemma C_C :
(C_basic t x cote)->
(L_Etat (t+2) (x+cote))->
(L_Etat (t+3) (x+cote))->
(C_basic (t+2) x cote).
```

qui se démontre comme A_A;

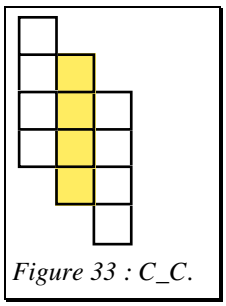


Figure 33 : C_C.

• soit 2 figures différentes décalées de même longueur :

```
Lemma A_B :
(A_basic t x cote)->
(L_Etat  (t+1) (x+cote+1))->
(L_Etat (t+2) (x+cote+1))->
(B_basic (t+1) (x+1) cote).
```

qui demande un appel à DD_D', puis un appel à D_D'D;

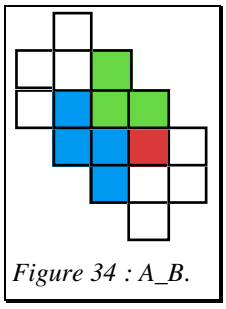


Figure 34 : A_B.

```
Lemma B_C :
(B_basic t x cote)->
(L_Etat  (t+1) (x+cote+1))->
(L_Etat (t+2) (x+cote+1))->
(C_basic (t+1) (x+1) cote).
```

alors que ce lemme demande un appel à D'D_D puis un à D_DD;

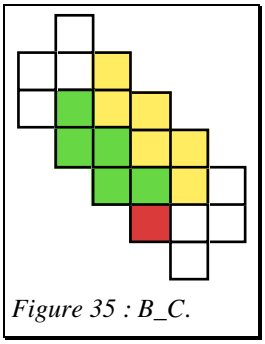


Figure 35 : B_C.

• soit enfin 2 figures différentes décalées et de longueur différentes :

```
Lemma C_A :
(C_basic t x cote)->
(L_Etat (t+1) (x+cote+1))->
(L_Etat (t+2) (x+cote+1))->
(A_basic (t+1) x (cote+1)).
```

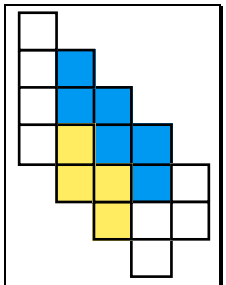


Figure 36 : C_A.

qui utilise successivement le lemme DD_D$ et le lemme D_DD$.

Toutes les démonstrations suivent le même schéma, en faisant appel aux lemmes de la section des constructions, puis en vérifiant que les hypothèses d'application de ces lemmes instancies par les valeurs d'etats correspondantes sont bien verifiees par la fonction de transition decrite dans les tables de la section automate.

Cette section s'achève par une série de petits lemmes caractérisants des cas particuliers utilisés dans la suite.

```
Lemma GA_G : (t,x:nat)
   (G_Etat t x) -> (A_Etat t (x+1)) -> (G_Etat (t+1) x).

Lemma GB_G : (t,x:nat)
   (G_Etat t x) -> (B_Etat t (x+1)) -> (G_Etat (t+1) x).

Lemma GC_G : (t,x:nat)
   (G_Etat t x) -> (C_Etat t (x+1)) -> (G_Etat (t+1) x).

Lemma GA_$C : (t,x:nat)
(G_Etat t x) -> (A_Etat t (x+1)) ->
                                    (C_Etat (t+1) (x+1)).

Lemma GBA_$C : (t,x:nat)
(G_Etat t x) -> (B_Etat t (x+1)) -> (A_Etat t (x+2)) ->
   (C_Etat (t+1) (x+1)).

Lemma GBG_$G : (t,x:nat)
(G_Etat t x) -> (B_Etat t (x+1)) -> (G_Etat t (x+2)) ->
   (G_Etat (t+1) (x+1)).

Lemma GBC_$B : (t,x:nat)
(G_Etat t x) -> (B_Etat t (x+1)) -> (C_Etat t (x+2)) ->
   (B_Etat (t+1) (x+1)).

Lemma GC_$B : (t,x:nat)
(G_Etat t x) -> (C_Etat t (x+1)) ->
                                    (B_Etat (t+1) (x+1)).
```

Le bord gauche.

Cette section s'occupe plus particulièrement du bord gauche de la figure. On observe la répétition de motifs se terminant par des états G, formant ainsi (à l'exception de la case t=2) une verticale d'états G (rouge). Ces figures ont des noms tires du papier de J.Mazoyer d'ou le decalage d'une unite entre le nom et la longueur.

```
Inductive un_end [t,x:nat] :Prop :=
make_un_end :
(G_Etat t x)->
(G_Etat (t+1) x)->
(un_end t x).
```

Figure 37 : un_end.

```
Inductive deux_end [t,x:nat] :Prop :=
make_deux_end :
(C_Etat t (x+1))->
(B_Etat (t+1) (x+1))->
(un_end (t+1) x)->
(deux_end t x).
```

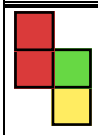


Figure 38 : deux_end.

```
Inductive trois_end [t,x:nat] :Prop :=
make_trois_end :
(A_Etat t (x+2))->
(G_Etat (t+1) (x+2))->
(deux_end (t+1) x)->
(trois_end t x).
```

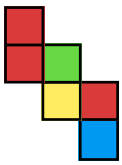


Figure 39 : trois_end.

```
Inductive quatre_end [t,x:nat] : Prop :=
make_quatre_end :
(L_Etat t (x+3))->
(L_Etat (t+1) (x+3))->
(trois_end (t+1) x)->
(quatre_end t x).
```

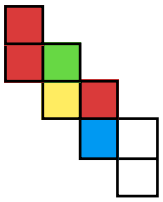


Figure 40 : quatre_end.

```
Inductive cinq_end [t,x:nat] : Prop :=
make_cinq_end :
(L_Etat t (x+4))->
(L_Etat (t+1) (x+4))->
(G_Etat (t+1) (x+3))->
(B_Etat (t+2) (x+3))->
(trois_end (t+2) x)->
(cinq_end t x).
```

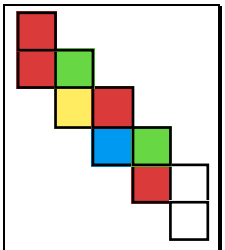


Figure 41 : cinq_end.

Remarque:

Grâce au fait que l'on a pu autoriser l'existence de figures C_basic de longueur 2, il n'est plus necessaire de definir la figure six_end de J.Mazoyer; celle-ci est l'aboutement d'une figure quatre_end et d'une figure C_basic de longueur 2.

Par construction, toutes les figures n_end se terminent à gauche par deux cases dans l'état G. Cette propriete etant utilisee par la suite, elle fait l'objet de petits lemmes triviaux :

```
Lemma un_GG : (t,x:nat)
(un_end t x)-> (G_Etat t x) /\ (G_Etat (t+1) x).

Lemma deux_GG : (t,x:nat)
(deux_end t x)-> (G_Etat (t+1) x) /\ (G_Etat (t+2) x).

Lemma trois_GG : (t,x:nat)
(trois_end t x)-> (G_Etat (t+2) x) /\ (G_Etat (t+3) x).

Lemma quatre_GG : (t,x:nat)
(quatre_end t x)-> (G_Etat (t+3) x) /\ (G_Etat (t+4) x).
```

```
Lemma cinq_GG : (t,x:nat)
(cinq_end t x)-> (G_Etat (t+4) x) /\ (G_Etat (t+5) x).
```

Les lemmes suivants définissent l'empilement vertical des structures n_end :

```
Lemma un_deux : (t,x:nat)
(un_end t x)->
(C_Etat (t+1) (x+1))->
(B_Etat (t+2) (x+1))->
(deux_end (t+1) x).
```

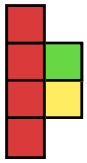


Figure 42 : un_deux.

```
Lemma deux_trois : (t,x:nat)
(deux_end t x)->
(A_Etat (t+1) (x+2))->
(G_Etat (t+2) (x+2))->
(trois_end (t+1) x).
```

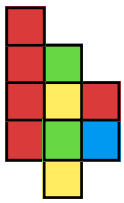


Figure 43 : deux_trois.

```
Lemma deux_quatre : (t,x:nat)
(deux_end t x)->
(L_Etat t (x+2))->
(L_Etat t (x+3))->
(L_Etat (t+1) (x+3))->
(quatre_end t x).
```

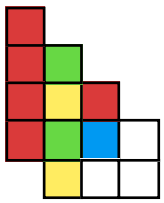


Figure 44 : deux_quatre.

```
Lemma trois_quatre : (t,x:nat)
(trois_end t x)->
(L_Etat (t+1) (x+3))->
(L_Etat (t+2) (x+3))->
(trois_end (t+2) x).
```

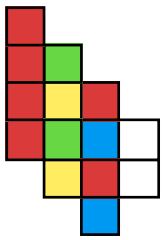


Figure 45 : trois_quatre.

```
Lemma trois_cinq : (t,x:nat)
(trois_end t x)->
(G_Etat (t+1) (x+3))->
(B_Etat (t+2) (x+3))->
(trois_end (t+2) x).
```

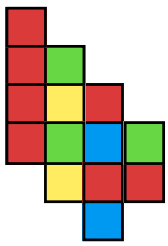


Figure 46 : trois_cinq.

Cela permet de constater que les structures n_end ont un fonctionnement semblable à celui des x_basic en ce sens que :

- deux structures identiques s'empilent alignées si deux cellules quiescentes sont presentes sur le bord droit :

```
Lemma quatre_quatre : (t,x:nat)
(quatre_end t x)->
(L_Etat (t+2) (x+3))->
(L_Etat (t+3) (x+3))->
(quatre_end (t+2) x).
```

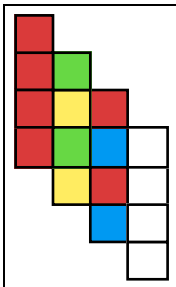


Figure 47 : quatre_quatre.

```
Lemma cinq_cinq : (t,x:nat)
(cinq_end t x)->
(L_Etat (t+2) (x+4))->
(L_Etat (t+3) (x+4))->
(cinq_end (t+2) x).
```

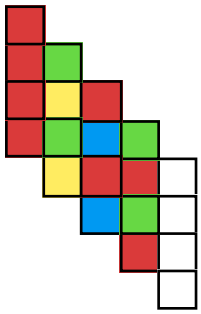


Figure 48 : cinq_cinq.

et deux cellules différentes s'empilent décalées si les deux cellules quiescentes sont decalees :

```
Lemma quatre_cinq : (t,x:nat)
(quatre_end t x)->
(L_Etat (t+1) (x+4))->
(L_Etat (t+2) (x+4))->
(cinq_end (t+1) x).
```

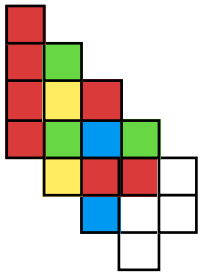


Figure 49 : quatre_cinq.

```
Lemma cinq_quatre : (t,x:nat)
(cinq_end t x)->
(L_Etat (t+1) (x+5))->
(L_Etat (t+2) (x+5))->
((C_basic (t+1) (x+3) 2) /\
         (quatre_end (t+3) x)).
```

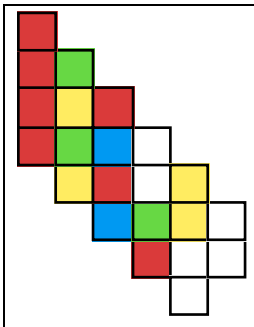


Figure 50 : cinq_quatre.

On notera que la dernière figure obtenue par décalage à partir d'un cinq_end est l'aboutement d'un quatre_end et d'un C_basic de longueur 2 sur la diagonale.

Il s'agit désormais de considérer le bord droit. Sur ce bord, le signal parvient au bout d'un temps N et se reflechit. La reflexion doit fournir les informations relatives au reste modulo 3 de N de façon à gerer correctement le point d'appel de recurrence au deux tiers de N. On definit donc les signaux reflechis suivants (les notations sont celles de J.Mazoyer) :

```
Inductive UA [t,x,cote:nat] : Prop :=
make_UA :
(1<cote)->
(Diag t x cote G_Etat A_Etat G_Etat)->
(UA t x cote).
```

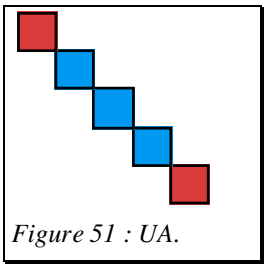


Figure 51 : UA.

```
Inductive UAB [t,x,cote:nat] : Prop :=
make_UAB :
(2<cote)->
(Diag' t x cote G_Etat G_Etat B_Etat
                              G_Etat)->
(Diag (t+1) x cote G_Etat A_Etat
                              G_Etat)->
(UAB t x cote).
```

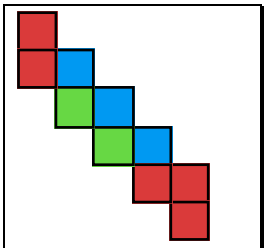


Figure 52 : UAB.

```
Inductive ZCB [t,x,cote:nat] : Prop :=
make_ZCB :
(1<cote)->
(Diag t x cote G_Etat C_Etat G_Etat)->
(Diag (t+1) x cote G_Etat B_Etat
                              G_Etat)->
(ZCB t x cote).
```

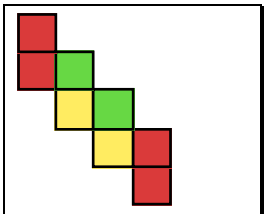


Figure 53 : ZCB.

Selon la dernière diagonale produite lors de la propagation aller du signal, on a :

```
Lemma B_UA :
(B_basic t x cote)->
(G_Etat (t+1) (x+cote+1))->
(UA (t+1) (x+1) cote).
```

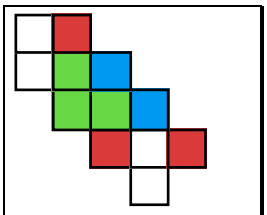


Figure 54 : B_UA.

La démonstration utilise principalement le lemme D'D_D.

```
Lemma C_UAB :
(2<cote)->
(C_basic t x cote)->
(G_Etat (t+1) (x+cote+1))->
(G_Etat (t+2) (x+cote+1))->
(UAB (t+1) (x+1) cote).
```

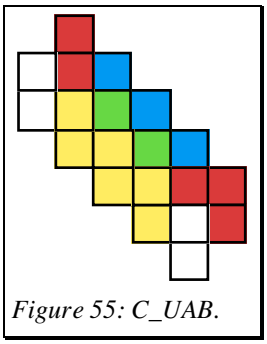


Figure 55: C_UAB.

On applique successivement les lemmes DD_D' et D_D'D pour prouver les deux lignes.

```
Lemma A_ZCB :
(A_basic t x cote)->
(G_Etat (t+1) (x+cote+1))->
(G_Etat (t+2) (x+cote+1))->
(ZCB (t+1) (x+1) cote).
```

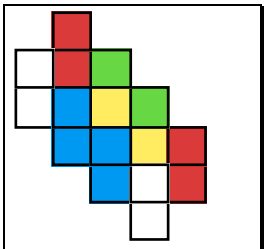


Figure 56: A_ZCB.

Ce sont les lemmes DD_D et D_DD qu'il faut appliquer cette fois.

Comme seules les diagonales de type B acceptent une diagonale de type quiescent au-dessus (c'est d'ailleurs la raison pour laquelle on a employé des Diag' et non des Diag dans les constructions precedentes) il faut fermer les deux premieres figures :

```
Lemma B_ZCB :
(B_basic t x cote)->
(G_Etat (t+1) (x+cote+1))->
(G_Etat (t+2) (x+cote+1))->
(G_Etat (t+3) (x+cote+1))->
(ZCB (t+2) (x+1) cote).
```

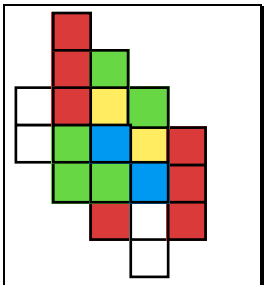


Figure 57 : B_ZCB.

On utilise successivement les lemmes B_UA, puis D_DD et DDD.

```
Lemma C_ZCB :
(2<cote)->
(C_basic t x cote)->
(G_Etat (t+1) (x+cote+1))->
(G_Etat (t+2) (x+cote+1))->
(G_Etat (t+3) (x+cote+1))->
(G_Etat (t+4) (x+cote+1))->
(ZCB (t+3) (x+1) cote).
```

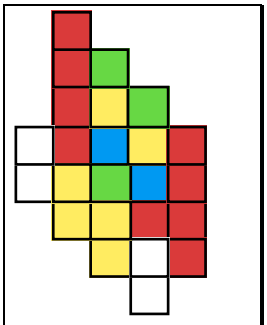


Figure 58 : C_ZCB.

On utilise C_UAB, puis D'DD et DDD pour démontrer ce lemme.

On remarque que l'alternance des états pour cette terminaison est dans l'ordre ACB, soit l'ordre inverse de celui qui prevalait aux constructions precedentes (§ briques de base), ABC qui servait à compter modulo 3 en propageant le signal vers la droite.

Le but de ces dernières constructions est de pouvoir empiler un triangle d'état quiescent sur le retour du signal vers la gauche afin de retrouver une situation semblable à la situation initiale (appel recursif) de longueur 1/3. Pour cela, on montre successivement les lemmes suivants :

```
Lemma ZCB_GLC :
(2<cote) ->
(ZCB t x cote) ->
(Verticale (t+2) (x+cote) cote
                        G_Etat) ->
(Diag (t+2) (x+1) (cote-1) G_Etat
                  L_Etat C_Etat).
```

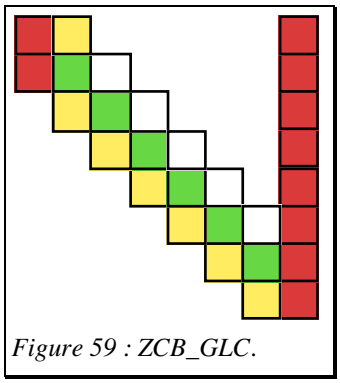


Figure 59 : ZCB_GLC.

application du lemme DD$_D,

```
Lemma ZCB_l :
(2<cote) ->
(ZCB t x cote) ->
(Verticale (t+2) (x+cote) cote
                      G_Etat) ->
(Semi_Diag (t+3) (x+2) (cote-2)
               G_Etat L_Etat).
```

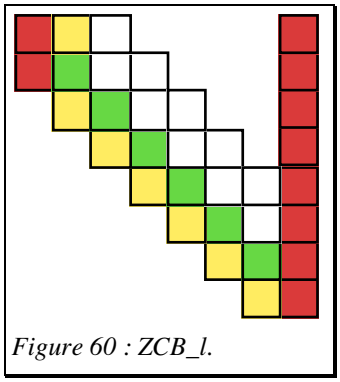


Figure 60 : ZCB_l.

application de DD_d au lemme précédent,

```
Lemma ZCB_ll :
(3<cote) ->
(ZCB t x cote) ->
(Verticale (t+2) (x+cote) cote
                      G_Etat) ->
(Semi_Diag (t+4) (x+3) (cote-3)
               G_Etat L_Etat).
```

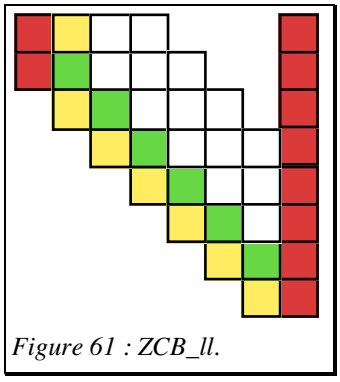


Figure 61 : ZCB_ll.

application de Dd_d au lemme précédent,

et enfin le résultat général de ce triangle d'états quiescents :

```
Lemma ZCB_lll : (dcote:nat)
(2≤dcote) ->
(dcote<cote) ->
(ZCB t x cote) ->
(Verticale (t+2) (x+cote) cote
                      G_Etat) ->
(Semi_Diag (t+dcote+1) (x+dcote)
   (cote-dcote) G_Etat L_Etat).
```

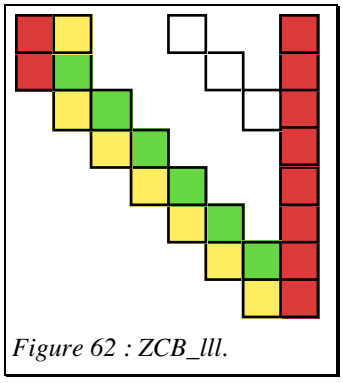


Figure 62 : ZCB_lll.

qui se démontre par récurrence sur la variable dcote et en appliquant le lemme dd_d.

Une conséquence intéressante de ce lemme est :

```
Lemma ZCB_Ht1 :
(2<cote) ->
(ZCB t x cote) ->
(Verticale (t+2) (x+cote) cote
                      G_Etat) ->
(Horizontale_t1 (t+cote+1) x
   (cote-3) G_Etat C_Etat L_Etat).
```

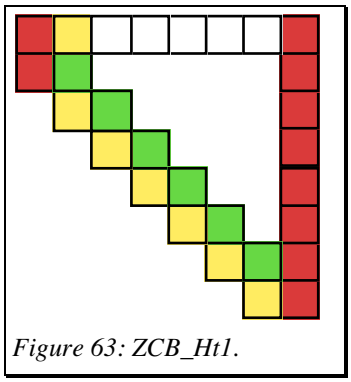


Figure 63: ZCB_Ht1.

La démonstration se fait par décomposition de la structure Horizontale_t1 et application des lemmes precedents.

Cette section s'achève par l'établissement de quelques lemmes relatifs au bord gauche de ces figures qui contient l'amorce d'une verticale d'etats G.

```
Lemma A_Vg :
(A_basic t x cote)->
(G_Etat (t+1) (x+cote+1))->
(G_Etat (t+2) (x+cote+1))->
(Verticale (t+cote+1) (x+1) (1) G_Etat).
```

conséquence immédiate du lemme A_ZCB,

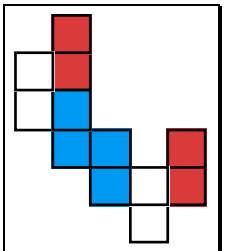


Figure 64 : A_Vg.

```
Lemma B_Vg :
(B_basic t x cote)->
(G_Etat (t+1) (x+cote+1))->
(G_Etat (t+2) (x+cote+1))->
(G_Etat (t+3) (x+cote+1))->
(Verticale (t+cote+1) (x+1) (2) G_Etat).
```

conséquence des lemmes B_ZCB et B_UA,

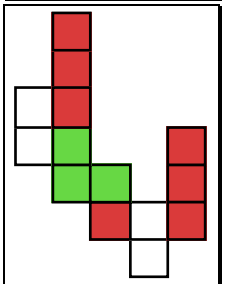


Figure 65 : B_Vg.

```
Lemma C_Vg :
(2<cote)->
(C_basic t x cote)->
(G_Etat (t+1) (x+cote+1))->
(G_Etat (t+2) (x+cote+1))->
(G_Etat (t+3) (x+cote+1))->
(G_Etat (t+4) (x+cote+1))->
(Verticale (t+cote+1) (x+1) (3) G_Etat).
```

conséquence des lemmes C_ZCB et C_UAB.

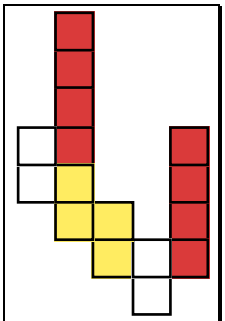


Figure 66 : C_Vg.

Les double-diagonales.

Après avoir construit des briques de base (en un certain sens des listes de cases), nous allons aborder la construction clef utilisant ces briques (en poursuivant l'analogie une liste de listes).

En simplifiant la figure obtenue en développant dans le temps l'évolution de la ligne d'automate, on obtient une figure rectangulaire (§ le probleme). Ce rectangle peut etre schematiquement divise en trois triangles par la propagation du signal de gauche à droite et par son retour. Le triangle inferieur est quiescent, le triangle superieur est essentiellement construit recursivement.

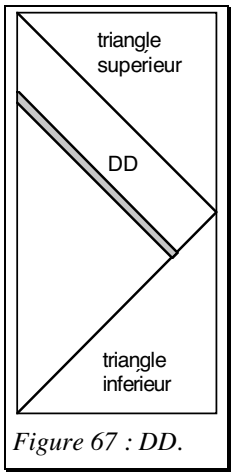


Figure 67 : DD.

Le triangle central est forme des briques de bases que nous avons définies précedemment, les figures A_basic, B_basic, C_basic, quatre_end et cinq_end. Ces briques sont assemblees sur les diagonales en une structure unique appelee DD (pour double-diagonale).

La definition de DD utilise cinq constructeurs (on utilise les notations de congruence modulo 3 et (cote:3) designe la division entiere par 3); la figure placee en regard contient les dessins de DD pour les premieres valeurs de cote (de 3 à 7) soit respectivement DD_4, DD_5, DD_C, DD_A et DD_B :

```
Inductive DD :
   nat -> nat -> nat -> Prop :=
DD_4 : (t,x:nat)
   (quatre_end t x)->
         (DD t  x (3))
| DD_5 : (t,x:nat)
   (cinq_end t x)->
         (DD t x (4))
| DD_A : (t,x,cote:nat)
   (6≤cote)->
   (cote=0 mod 3)->
   (A_basic t (x+2.(cote:3)-1)
              ((cote:3)+1))->
   (DD (t+(cote:3)+1) x
              (2.(cote:3)-1))->
         (DD t  x cote)
| DD_B : (t,x,cote:nat)
   (7≤cote)->
   (cote=1 mod 3)->
   (B_basic t (x+2.(cote:3))
              ((cote:3)+1))->
   (DD (t+(cote:3)+1) x
              (2.(cote:3)))->
         (DD t  x cote)
| DD_C : (t,x,cote:nat)
   (5≤cote)->
   (cote=2 mod 3)->
   (C_basic t (x+2.(cote:3)+1)
              ((cote:3)+1))->
   (DD (t+(cote:3)+1) x
              (2.(cote:3)+1))->
         (DD t  x cote).
```

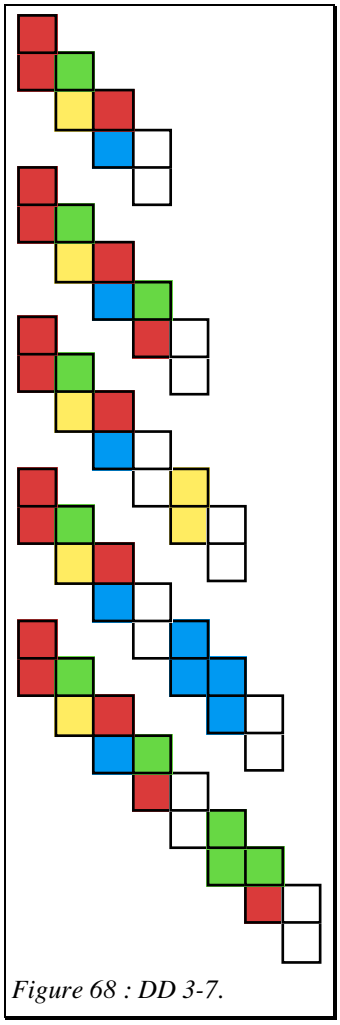


Figure 68 : DD 3-7.

On établit d'abord un petit lemme facile disant que le bord gauche d'une structure DD est forme de deux cases dans l'etat G :

```
Lemma DD_GG : (t,x,cote:nat)
(DD t x cote) ->
(G_Etat (t+cote) x) /\ (G_Etat (t+cote+1) x).
```

On termine cette section en montrant deux lemmes importants relatifs à l'empilement des structures DD d'abord sans decalage horizontal puis avec un decalage.

Le lemme relatif à l'empilement sans décalage :

```
Theorem DD_hh : (t, x,cote:nat)
(DD t x cote)->
(L_Etat (t+2) (x+cote))->
(L_Etat (t+3) (x+cote))->
(DD (t+2) x cote).
```

peut s'illustrer avec la figure suivante :

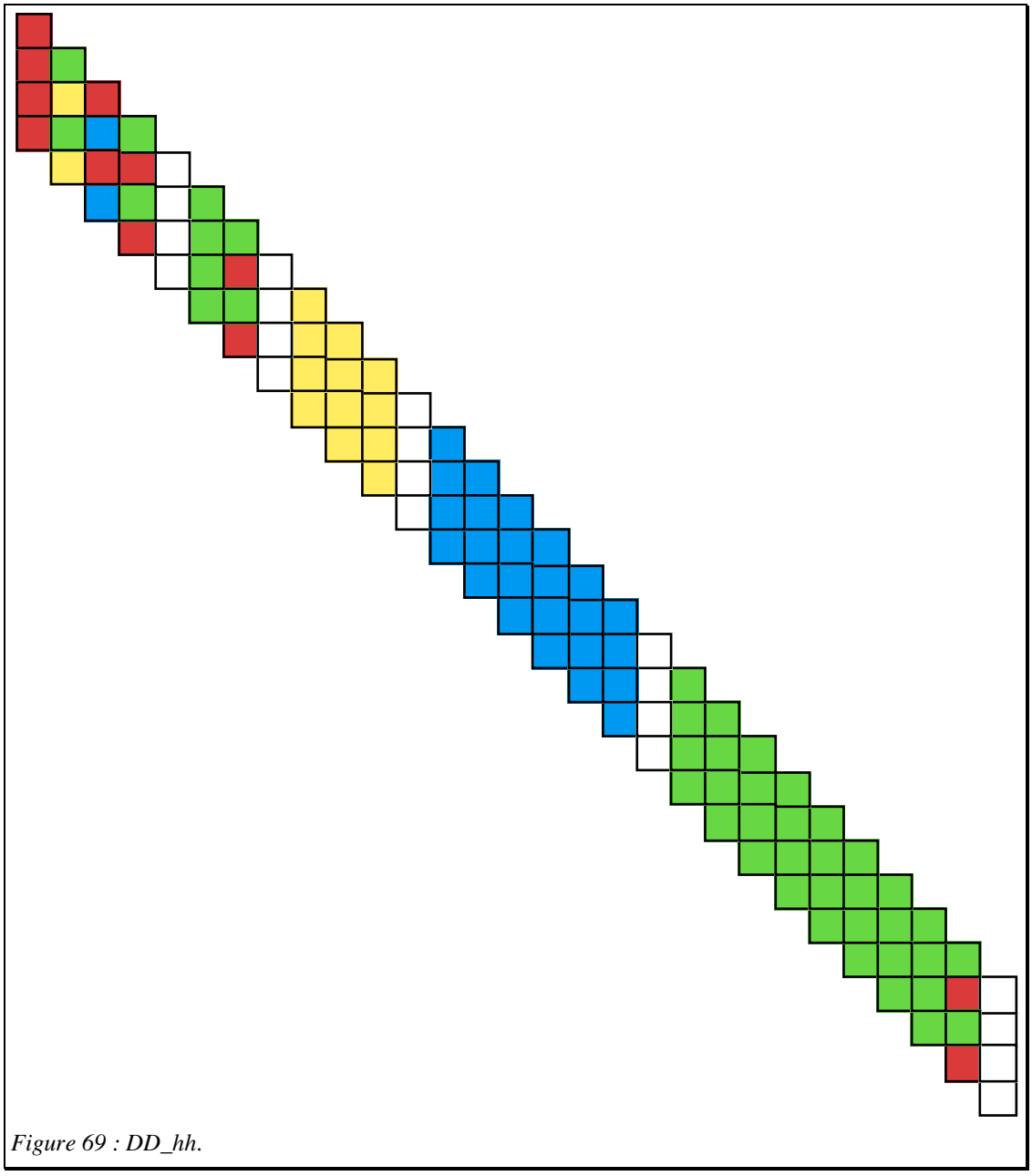


Figure 69 : DD_hh.

La démonstration procède d'abord à une décomposition par cas de l'hypothèse (DD t x cote). Les deux premiers cas (DD_4 et DD_5) sont des applications directes des lemmes quatre_quatre et cinq_cinq du chapitre bord gauche. Les trois autres cas se font par recurrence en appliquant les lemmes A_A, B_B et C_C du chapitre briques de base.

Le lemme relatif à l'empilement avec un décalage :

```
Theorem DD_hd$ : (t, x,cote:nat)
(DD t x cote)->
(L_Etat (t+1) (x+cote+1))->
(L_Etat (t+2) (x+cote+1))->
(DD (t+1) x (cote+1)).
```

peut s'illustrer par la figure suivante :

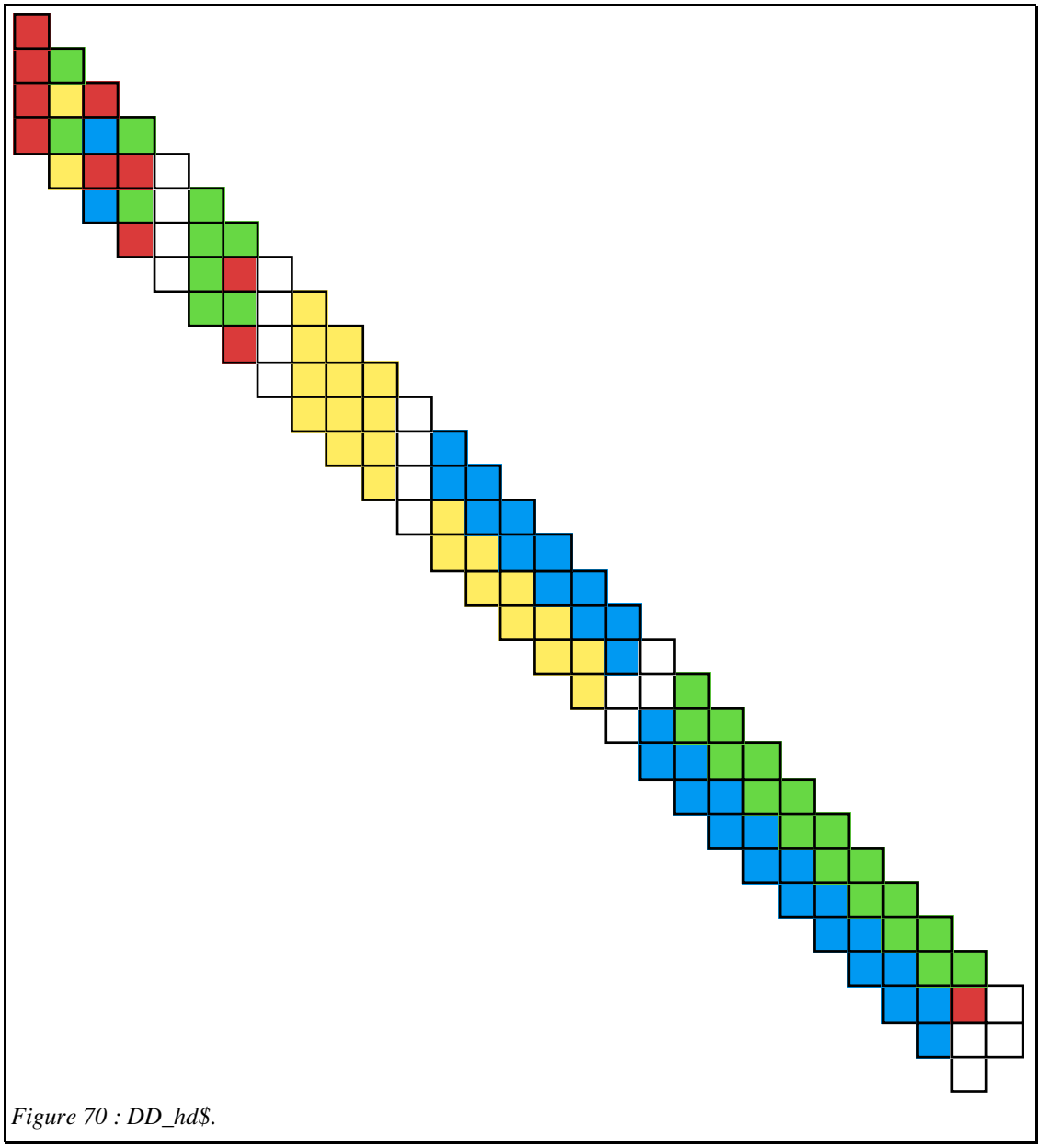


Figure 70 : DD_hd$.

Sa démonstration, bien que basée sur le même principe est un peu plus délicate. On distingue les cas selon (DD t x cote) :

- s'il s'agit d'un DD_4, le résultat est un DD_5 ce que le lemme quatre_cinq établit;
- s'il s'agit d'un DD_5, le résultat (DD_C 6) se montre en utilisant le lemme cinq_quatre qui donne à la fois le quatre_end final et la structure C_basic de longueur 2;
- s'il s'agit d'un DD_A, le résultat est un DD_B, il faut utiliser une récurrence et appliquer le théorème A_B;
- de même le DD_B donne DD_C par récurrence et application de B_C;
- par contre le cas DD_C qui donne DD_A rallonge la structure basic terminale d'une unité, il y a application du lemme C_A pour la partie terminale et du lemme précédent DD_hh pour la structure DD restante qui de ce fait se retrouve de même longueur que sa génératrice.

De plus cette démonstration fait appel à plusieurs lemmes d'arithmétique (placés en bibliothèque) pour gérer le reste modulo 3 de cote+1 en fonction du reste modulo 3 de cote ainsi que les différentes longueurs des diagonales.

Cette section contient un certain nombre de petits lemmes permettant d'avoir tous les outils pour mener à bien les recursivites du triangle superieur.

On commence par gerer le triangle inferieur, non plus à partir de la situation initiale car elle ne se retrouve plus dans le triangle superieur mais à partir de la troisieme ligne qui est du type Horizontale_t1.

```
Lemma Ht1_End2 :
   (t,x,long :nat)
(Horizontale_t1 t x long G_Etat
                C_Etat L_Etat) ->
(deux_end t x).
```

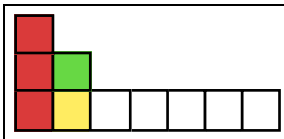


Figure 71 : Ht1_End2.

```
Lemma Ht1_End4 :
   (t,x,long :nat)
(0<long) ->
(Horizontale_t1 t x long G_Etat
                C_Etat L_Etat) ->
(quatre_end t x).
```

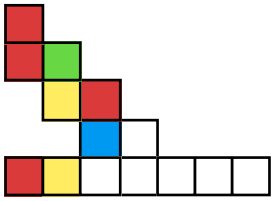


Figure 72 : Ht1_End4.

```
Lemma Hor_tr_inf :   (t,x,cote:nat)
(Horizontale t x cote L_Etat) ->
(Triangle_inf t x cote L_Etat).
```

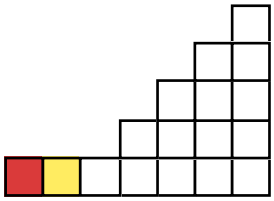


Figure 73 : Hor_tr_inf.

On poursuit avec le rectangle median dans lequel on montre que les structures DD et la verticale d'etat G sur le bord gauche sont les consequence d'une Horizontale_t1.

```
Lemma Ht1_bissect :
   (t,x,cote:nat)
(0<cote) ->
(Horizontale_t1 t x cote G_Etat
                C_Etat L_Etat) ->
((dx:nat) ((dx+1)≤cote) ->
   (L_Etat (t+dx) (x+dx+3)) /\
   (L_Etat (t+dx+1) (x+dx+3))).
```

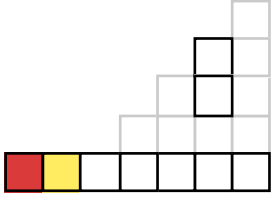


Figure 74 : Ht1_bissect.

C'est une consequence directe du lemme precedent.

```
Theorem Ht1_DD :     (t,x,cote:nat)
(0<cote) ->
(Horizontale_t1 t x cote   G_Etat
C_Etat L_Etat) ->
((dx:nat) ((dx+1)≤cote) ->
(DD (t+dx) x (dx+3))).
```

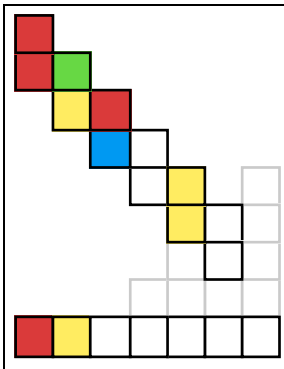


Figure 75 : Ht1_DD.

Ce theoreme s'etablit par recurrence avec le lemme DD_hd$.

Il admet le lemme suivant comme cas particulier, c'est la frontiere superieure du triangle median.

```
Lemma Ht1_DDf : (t,x,haut:nat)
(Horizontale_t1 t x (haut+1) G_Etat
C_Etat L_Etat) ->
(DD (t+haut) x (haut+3)).
```

Le lemme suivant regit le bord gauche du triangle median qui est une verticale d'etat G :

```
Lemma Ht1_VV : (t,x,cote:nat)
(Horizontale_t1 t x cote G_Etat
              C_Etat L_Etat) ->
(Verticale (t+1) x (2.cote+1)
                          G_Etat).
```

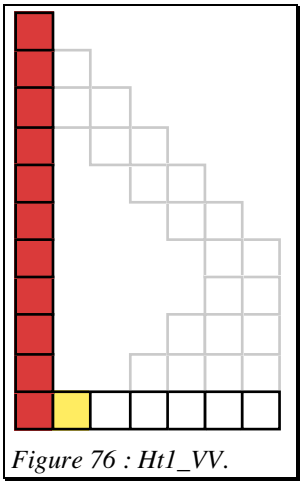


Figure 76 : Ht1_VV.

Pour établir ce lemme, on utilise le lemme rec_vert qui construit une verticale de longueur paire par aboutement de segments de longueur 2, obtenu par le lemme DD_GG.

Cette section se termine par des lemmes relatifs aux lignes t=0 et t=1 qui sont specifiques et ne participent pas à la recurrence.

On a donc ces hypotheses locales à cette section :

```
Hypothesis Assez_grand : (1<long).

Hypothesis Base :
(Horizontale_t0 t O long G_Etat
                          L_Etat).
```

On retrouve des versions particulieres des lemmes precedents :

```
Lemma Ht0_bissect : (dx:nat)
((dx+1)≤long) ->
(L_Etat (t+dx) (dx+2)) /\ (L_Etat (t+dx+1) (dx+2)).

Lemma Ht0_End2 : (deux_end (t+1) (0)).

Lemma Ht0_End4: (quatre_end (t+1) (0)).

Theorem Ht0_DD : (dx:nat)
((dx+2)≤long) -> (DD (t+dx+1) (0) (dx+3)).

Lemma Ht0_DDf : (DD (t+long-1) (0) (long+1)).
```

Les "remark"(s) sont des lemmes locaux à la section, ils ont pour but de simplifier l'ecriture des preuves en etablissant des faits utilises plusieurs fois. Par exemple

```
Remark B21 : (Etat (t+2) (1))=B.
```

Les trapèzes.

Le triangle superieur se decompose en un alignement de trapezes dont les cotes paralleles sont verticaux (verticales d'etats G). La section commence par une petite serie de lemmes arithmetiques tres specifiques à ce chapitre et sans interet (c'est pourquoi ils n'ont meme pas ete places en bibliotheque).

Il se poursuit par l'etude du trapeze de hauteur 2. Il est defini par les hypotheses suivantes :

```
Hypothesis Hh :
(Horizontale_t1 t x (0) G_Etat C_Etat
                                  L_Etat).

Hypothesis Hv :
(Verticale t (x+3) (2) G_Etat).
```

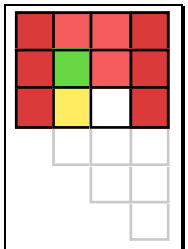


Figure 77 : trapèze_2.

Les propriétés intéressantes de ce trapèze sont :

```
Lemma H2_Vg :
(Verticale (t+1) x (1) G_Etat).

Lemma H2_Hh :
(Horizontale_t1 (t+1) x (0) G_Etat B_Etat
                                  G_Etat).

Lemma H2_Hg :
(Horizontale (t+2) x (2) G_Etat).
```

Puis on étudie successivement les trapèzes construits sur des diagonales A_basic, B_basic et C_basic.

On établit tout d'abord la frontière gauche :

```
Lemma Ha_Vg : (t,x,cote:nat)
(A_basic t x (cote+1))->
(Verticale (t+1) (x+cote+2) (3.cote)
                            G_Etat)->
(Verticale (t+cote+2) (x+1) (2.cote-1)
                            G_Etat).
```

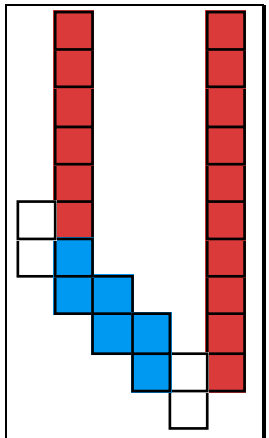


Figure 78 : Ha_Vg.

à l'aide des lemmes A_ZCB et ZCB_Vg pour la base, puis de ZCB_Ht1 et Ht1_VV pour la partie superieure.

On distingue ensuite le cas particulier cote=2 :

```
Lemma Ha3_Hg : (t,x:nat)
(A_basic t x (3))->
(Verticale (t+1) (x+4) (6) G_Etat)->
(Horizontale (t+7) (x+1) (2) G_Etat).
```

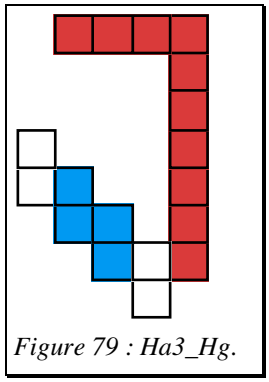


Figure 79 : Ha3_Hg.

où le trapèze est trop étroit pour avoir un appel récursif et où l'on conclut par une horizontale d'etat G en utilisant le lemme H2_Hg demontre precedemment.

Le cas général :

```
Lemma Ha_DD : (t,x,cote:nat)
(2<cote) ->
(A_basic t x (cote+1))->
(Verticale (t+1) (x+cote+2) (3.cote)
                            G_Etat)->
(DD (t+2.cote)) (x+1) cote).
```

amène un appel récursif. Il se démontre à l'aide des lemmes A_ZCB, ZCB_Ht1 et Ht1_DD.

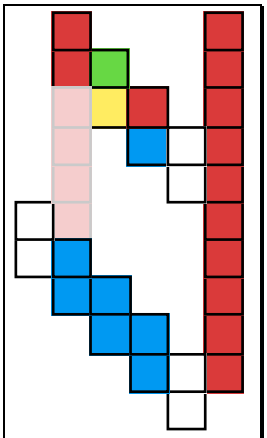


Figure 80 : Ha_DD.

On procède de même avec les trapèzes de base B_basic.

D'abord la verticale de gauche :

```
Lemma Hb_Vg : (t,x,cote:nat)
(B_basic t x (cote+1))->
(Verticale (t+1) (x+cote+2) (3.cote+1)
                            G_Etat)->
(Verticale (t+cote+2) (x+1) (2.cote)
                            G_Etat).
```

qui se démontre de la même façon avec B_Vg, B_ZCB, ZCB_Ht1, Ht1_VV et vv_vert pour assembler les morceaux.

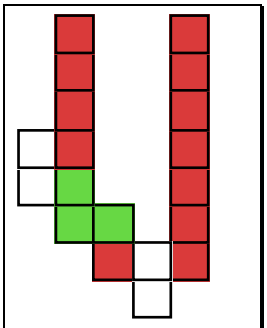


Figure 81 : Hb_Vg.

puis le cas particulier de longueur 3 :

```
Lemma Hb3_Hg : (t,x:nat)
(B_basic t x (3))->
(Verticale (t+1) (x+4) (7) G_Etat)->
(Horizontale (t+8) (x+1) (2) G_Etat).
```

qui utilise aussi B_ZCB, ZCB_Ht1 et H2_Hg,

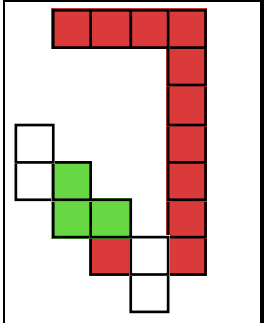


Figure 82: Hb3_Hg.

enfin le cas général :

```
Lemma Hb_DD : (t,x,cote:nat)
(2<cote) ->
(B_basic t x (cote+1))->
(Verticale (t+1) (x+cote+2)
                 (3.cote+1) G_Etat)->
(DD (t+2.cote+1) (x+1) cote).
```

qui se démontre aussi avec B_ZCB, ZCB_Ht1 et Ht1_DD.

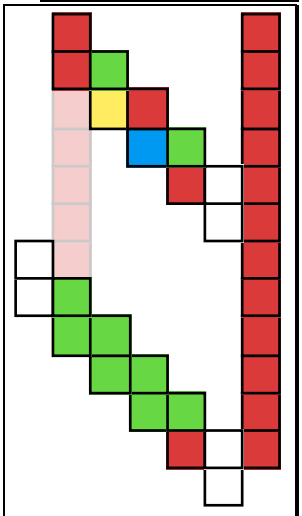


Figure 83 : Hb_DD.

L'étude des trapèzes basés sur une diagonale C_basic possède deux cas particuliers pour des longueurs 2 et 3. – On notera qu'en reorganisant un peu ces demonstrations, ces deux constructions mériteraient d'être rendues plus generales car elles permettraient d'etendre le theoreme general en s'affranchissant de la contrainte "necessaire : 2<N" –.

La construction du cas particulier de longueur 2 se construit avec :

```
Hypothesis Hc : (C_basic t x (2)).

Hypothesis  Hv :
(Verticale (t+1) (x+3) (5) G_Etat).

Lemma Hc2_Vg :
(Verticale (t+3) (x+1) (3) G_Etat).

Lemma Hc2_Hg :
(Horizontale (t+6) (x+1) (1) G_Etat).
```

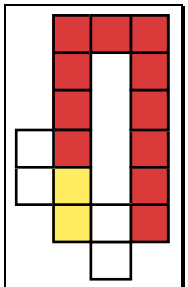


Figure 84 : Hc2.

Le reste de la démonstration est semblable aux démonstrations pour A_basic et B_basic.

D'abord la verticale de gauche :

```
Lemma Hc_Vg : (t,x,cote:nat)
(C_basic t x (cote+1))->
(Verticale (t+1) (x+cote+2) (3.cote+2)
                            G_Etat)->
(Verticale (t+cote+2) (x+1) (2.cote+1)
                            G_Etat).
```

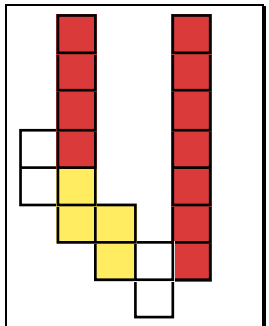


Figure 85 : Hc_Vg.

puis le cas particulier de longueur 3 :

```
Lemma Hc3_Hg : (t,x:nat)
(C_basic t x (3))->
(Verticale (t+1) (x+4) (8) G_Etat)->
(Horizontale (t+9) (x+1) (2) G_Etat).
```

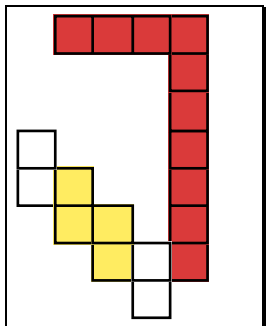


Figure 86 : Hc3_Hg.

enfin le cas général :

```
Lemma Hc_DD : (t,x,cote:nat)
(2<cote) ->
(C_basic t x (cote+1))->
(Verticale (t+1) (x+cote+2) (3.cote+2)
                            G_Etat)->
(DD (t+2.cote+2) (x+1) cote).
```

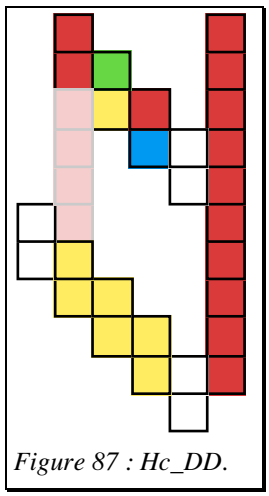


Figure 87 : Hc_DD.

On a désormais tous les outils nécessaires pour établir le théorème principal relatif au triangle superieur. celui-ci etablit le fait que si l'hypotenuse est de nature DD et le cote vertical une verticale d'etats G alors le cote horizontal est une horizontale d'etats G.

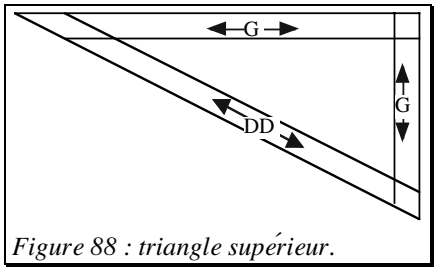


Figure 88 : triangle supérieur.

L'étude se fait par cas selon la nature du côté DD.

• Si DD est un quatre_end :

```
Variables t,x:nat.

Hypothesis He :
(quatre_end t x).

Hypothesis  Hv :
(Verticale (t+1) (x+4) (3)

   G_Etat).
```

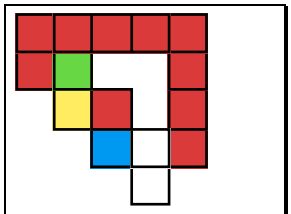


Figure 89 : quatre_Hg.

le lemme s'établit en calculant tous les états :

```
Lemma quatre_Hg :
(Horizontale (t+4) x (3) G_Etat).
```

• Si DD est un cinq_end :

```
Variables t,x:nat.

Hypothesis He : (cinq_end t x).

Hypothesis  Hv :
(Verticale (t+1) (x+5) (4) G_Etat).
```

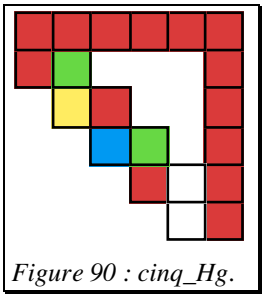


Figure 90 : cinq_Hg.

on calcule de même :

```
Lemma cinq_Hg :
(Horizontale (t+5) x (4) G_Etat).
```

• Il reste à étudier le cas général :

```
Theorem DD_Hg :  (t,x,cote:nat)
(DD t x cote) ->
(Verticale (t+1) (x+cote+1) cote
                      G_Etat) ->
(Horizontale (t+cote+1) x cote
                      G_Etat).
```

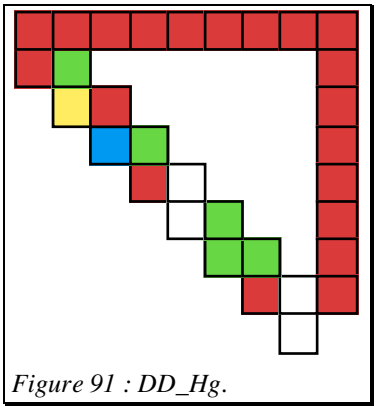


Figure 91 : DD_Hg.

Ce théorème s'établit par induction sur la structure de DD, en utilisant les lemmes quatre_Hg et cinq_Hg pour les cas terminaux, Ha3_DD, Hb3_DD ou Hc3_DD pour les recurrences simples et Ha_DD, Hb_DD ou Hc_DD pour les recurrences doubles ou les deux figures de la decomposition font appel au theoreme. Dans tous les cas sauf les deux premiers, la ligne horizontale d'etats G est construite par aboutement de deux lignes d'etats G (lemme hh_hor). Ce theoreme demande egalement quelques calculs auxiliaires de longueur qui ont ete etablit dans des lemmes ("remark") preliminaires.

Cette section se termine par un lemme facile prouvant qu'une ligne d'etats F se superpose à une ligne d'etats G.

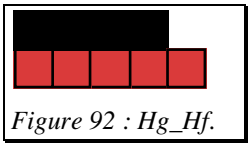


Figure 92 : Hg_Hf.

```
Lemma Hg_Hf : (t,long:nat)
(0<long) ->
(Horizontale t (0) long G_Etat) ->
(G_Etat t (long+1)) ->
(Horizontale (t+1) (0) long F_Etat).
```

En fin.

Tout est en place pour conclure.

• Dabord la situation initiale :

```
Lemma base1 :
(Horizontale_t0 (0) (0)
   (N-1) G_Etat L_Etat).
```

• puis le bord supérieur du triangle médian :

```
Lemma diagonale :
(DD (N-2) (0) N).
```

en appliquant Ht0_DDf;

•la situation initiale à droite :

```
Lemma base2 :
(Horizontale_t1 O (N+1)
   (N-1) G_Etat C_Etat
   L_Etat).
```

•et sa conséquence, le bord droit :

```
Lemma vert_droite :
(Verticale (1) (N+1)
(2.N-1) G_Etat).
```

en appliquant Ht1_VV;

•enfin le bord haut du triangle supérieur :

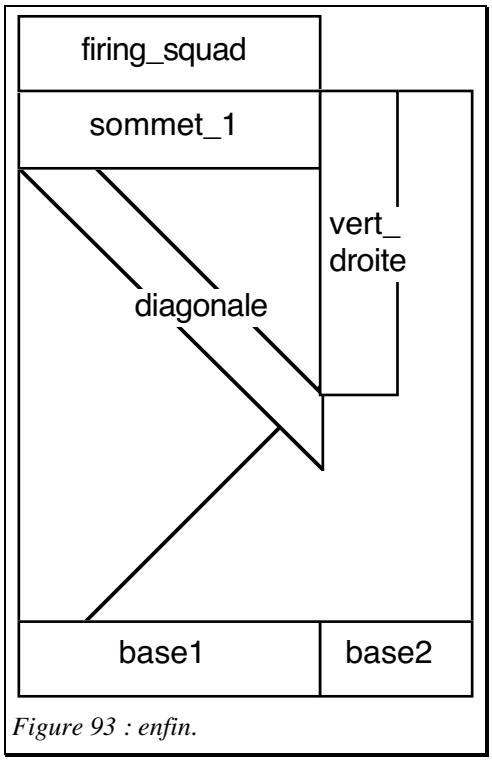


Figure 93 : enfin.

```
Lemma sommet_1 :
(Horizontale (2.N-1) (0) N G_Etat).
```

avec le lemme DD_Hg;

•qui permet de conclure :

```
Theorem firing_squad :
(Horizontale (2.N) (0) N F_Etat).
```

grâce au lemme Hg_Hf. –On rappelle que le fait de commencer à zéro transforme le 2N-1 du papier de J.Mazoyer en 2(N+1)-1-1–.

Première conclusion : aux concepteurs de Coq.

Une petite moitié de ce travail est constituée de manipulations élémentaires de formules sur les naturels. Il y a d'une part les petits lemmes de la bibliotheque et quelques lemmes locaux ne formalisant que des calculs numeriques simples, d'autre part les nombreuses occurrences de tactiques comme "Rewrite plus_Sn_m". Le but de ces manipulations est generalement de faire coïncider une expression parametre du but avec l'expression correspondante de l'hypothese ou du lemme que l'on veut appliquer. Ces manipulations sont fastidieuses, l'utilisateur souhaiterait evidemment se contenter de "auto" ou de "apply" avec une hypothese ou un lemme choisi (en general, l'utilisateur sait parfaitement quel lemme il veut utiliser). Bien sur ce n'est pas toujours simple et l'utilisateur comprend bien qu'il y a plusieurs façons d'unifier (S (S (S (plus t dt)))) avec (plus t' dt'), mais il aimerait avoir une façon plus commode de dire qu'il veut t':=(S (S t)) et dt':=(S dt) que d'appeler plusieurs regles de reecriture de plus et S. Le confort serait d'avoir un "apply" lie à un solveur d'equations (un peu ce qui existe dans PROLOG III de PrologIA) qui traduirait les unifications en contraintes, resoudrait celles qu'il saurait resoudre et placerait les autres en buts à prouver.

Par contre, Coq permet une mise au point très fine de la preuve, j'ai particulièrement noté :

• l'écriture de conditions minimales, (en l'absence de certitude, il suffit d'essayer);

• la verification de tous les cas sans risque d'erreurs, dans la preuve du firing squad, il suffit de penser au nombre de fois que les tables de transitions ont ete utilisees;

• la validation de modifications en cours de développement avec un minimum de travail lorsqu'une modification est faite sur un point déjà ecrit, il suffit de repasser toutes les preuves postérieures à ce point pour savoir lesquelles sont concernees et verifier ainsi sans risque d'oubli, que la suite du developpement est encore valide, voire de faire les modifications pour qu'il le devienne à nouveau.

Enfin, j'ai acquis la conviction que les logiciels d'aide à la preuve tels que Coq ont un champ d'utilisation comme langage universel de specification de probleme et d'ecriture de preuves. Verra-t-on un jour fleurir les demonstrations "à la Coq" dans les papiers de mathematiques discretes et d'informatique theorique comme on a vu fleurir les algorithmes "à la pascal" ? Avec en plus, l'avantage de trouver sur le reseau la demonstration complete, de pouvoir la verifier (reviewers), la comprendre et la reutiliser. J'ai la conviction que l'on y gagnera en lisibilite et en rigueur.

Un petit détail maintenant, il est difficile dans un travail important de mémoriser tous les lemmes déjà démontres. Meme en utilisant des noms mnemotechniques et des ecritures les plus standardisees possibles, il est frequent de rechercher le nom ou l'enonce d'un lemme. L'outil "search" est certes parfois pratique, mais s'avere souvent trop rudimentaire. Un outil plus puissant serait apprecie.

Deuxième conclusion : aux futurs utilisateurs.

Pourquoi choisir un assistant de preuves? Le fait d'avoir à suggérer des tactiques à Coq jusqu'à l'aboutissement de la preuve apparaît de prime abord comme un point faible par rapport aux veritables demonstrations automatiques. Apres usage, la conclusion est plutot inverse.

Certes, il est souvent fastidieux d'expliquer à Coq comment on démontre quelque chose qui nous semble facile, et la tentation est forte de ne pas trouver Coq bien intelligent. Mais l'usage habile de lemmes, ou la simple utilisation du couper-coller permet de ne pas repeter cette operation trop souvent. Par contre, le fait de rester le maître du processus de demonstration prend tout son interet dans l'echec. Une tentative de demontrer un lemme inexact ne peut aboutir, mais au fur et à mesure que les tactiques sont essayees, le point dur apparaît de plus en plus clairement permettant à l'utilisateur non seulement de se persuader de la faussete de l'assertion mais en plus d'en percevoir la raison. L'utilisateur reçoit alors des informations precieuses pour poursuivre sa demonstration en modifiant son raisonnement.

On note le confort qu'il y a eu dans cette longue preuve à laisser gérer toutes les petites conditions de longueur des figures par Coq. De meme, les transitions, une fois ecrites sont pratiquement oubliees, des qu'elles sont utilisees dans la preuve, on laisse Coq faire les appels necessaires.

Il faut bien se persuader que Coq ne peut servir à trouver une démonstration. Par contre, l'expressivité de son langage permet facilement d'ecrire la preuve des que l'idee de la demonstration est claire. D'une certaine

façon, on peut dire que la preuve est à la démonstration ce que le programme est à l'algorithme : une formalisation rigoureuse.

Il y a d'ailleurs une grande similitude de méthodologie entre l'écriture d'une grosse preuve et celle d'un grand programme. Il faut commencer par une analyse rigoureuse, structurer sa preuve, eviter les noms passe-partout et utiliser au contraire des denominations precises, eviter les longues preuves et decouper autant que faire se peut en petits lemmes.

Références.

[MOO] E.F.Moore, Sequential machines,

Selected Papers, Addison-Wesley, Reading, MA, 1964, (213-214).

[MAZ] J.Mazoyer, A six-state minimal time solution to the firing squad synchronization problem, Theoretical Computer Science 50, North-Holland, 1987, (183-238).

[CO&] C.Cornes, J.Courant, J-C.Filliatre, G.Huet, P.Manoury, C.Muñoz, C.Murthy, C.Parent, C.Paulin-Mohring, A.Saibi, B.Werner, The Coq Proof Assistant Reference Manual, Version 5.10, Rapport technique n°0177, Juillet 1995, INRIA.