Remerciements

Je tiens à remercier en premier lieu mon directeur de stage, Gérard Huet, de m'avoir accueuillie au sein du projet Coq et proposé ce sujet, qui est une excellente introduction à mon futur travail de thèse. Ce stage m'a également beaucoup appris, aussi bien en ce qui concerne la pratique de Coq que la théorie d'une partie des mathématiques.

Sur ce dernier point, je tiens à remercier tout particulièrement Gilles Dowek qui m'a beaucoup enseigné et aidée, notamment sur la théorie de l'axiomatisation des nombres réels.

Je souhaite dire également un grand merci à Cristina Cornes et à Benjamin Werner, qui m'ont expliqué certaines difficultés et subtilités de Coq ainsi qu'à Christine Paulin pour son attention particulière concernant certaines difficultés théoriques de Coq.

Enfin, merci aux deux équipes Coq et Cristal pour leur accueil chaleureux et en particulier à Daniel de Rauglaudre pour m'avoir appris beaucoup de choses concernant les stations de travail et ainsi que pour sa bonne humeur bien agréable.

Table des matières

1 Introduction **5**

2 Enoncé et preuve du théorème **7**
2.1 Enoncé. 7
2.2 Démonstration. 8

3 Cheminement - Théorie mathématique **11**
3.1 Cheminement. 11
3.1.1 Les nombres réels. 11
3.1.2 Les outils spécifiques. 11
3.2 Théorie mathématique. 12
3.2.1 Les Réels. 12
3.2.2 Les outils. 14

4 Choix pour la spécification en Coq **17**
4.1 Prop, Set et Type 17
4.1.1 Prop 17
4.1.2 Set 19
4.1.3 Type 20
4.2 Les choix 21

5 Commentaires sur l'annexe **25**
5.1 Axiomatisation. 25
5.2 Le théorème. 28

6 Architecture **31**

7 Conclusion **33**

A Compléments pour les Relatifs **35**

B Axiomatisation des Réels **37**

C Compléments pour les Naturels **57**

D Outils pour le théorème **61**

E Preuve intermédiaire 1 **67**

F Preuve intermédiaire 2 **69**

G Preuve intermédiaire 3 **71**

H Preuve intermédiaire 4 **73**

I The three gap theorem **75**

Chapitre 1

Introduction

Le but de ce stage est de démontrer -si cela est possible- un théorème des mathématiques en utilisant le langage d'aide à la preuve Coq.
Le théorème choisi est le "Théorème des trois intervalles" -ou conjecture de Steinhaus datant des années 50- dont une preuve est celle de Tony Van Ravenstein en 1986 [TR86].

Valeur de la démarche

Comme nous le verrons ci-dessous plus en détail, Coq est un système d'aide à la preuve. Il doit donc pouvoir être utilisé en tant que tel afin d'aider à prouver le plus grand nombre possible de théorèmes. Dans cette optique, l'intérêt d'une telle démarche est double:

·dans ce cas précis, la démonstration nécessite un passage obligé par les nombres réels, qui ne sont pas implémentés en Coq;

·il s'agit de résultats théoriques faisant appel à des intuitions géométriques mais qui sont difficiles à formaliser en Coq et dont le genre de preuves reste très peu exploré.
Ceci nous montre les difficultés inattendues que peuvent engendrer de tels théorèmes: dans notre cas, par exemple, l'énoncé paraît sans grande difficulté (nous pourrions croire que les entiers suffisent), néanmoins il devient vite impossible à prouver avec les outils existants.
L'idée s'intègre aisément aux objectifs du projet Coq dans le sens où le système Coq en lui même ne demande qu'à être le plus performant possible (preuves de protocoles pour les cartes à puce, applications aux mathématiques théoriques et appliquées, etc.). En ce qui concerne les réels, certains travaux ont déjà été effectués aussi bien dans léquipe Coq que dans la communauté scientifique, nous pensons notamment aux travaux de [VM94] (CAML), [AE+97] (Java et C++) et [CJ93] (LEGO).

Présentation du système Coq

Le Calcul des Constructions Inductives avec univers est l'architecture logique sur laquelle est construit le système d'aide à la preuve Coq. Ce système

comprend trois sortes: Prop qui représente l'espace des propositions, Set qui représente l'espace des données et Type qui représente l'espace des genres avec une hiérarchie d'univers.
Les notations du λ-calcul utilisées pour exprimer la fonctionnalité, peuvent donc être utilisées pour coder les preuves en déduction naturelle: les termes preuve de Coq sont représentés grâce à l'isomorphisme de Curry-Howard.
Une première version du calcul des constructions (CoC) présentée par T. Coquand en 1985 et implémentée par T.Coquand et G.Huet s'appuyait sur deux théories des types: la première est celle de Martin-Löf intuitionniste, la seconde est celle du λ-calcul polymorphe de Girard (Fω). Ce système permettait une axiomatisation puissante mais sans définitions inductives.
Le formalisme fut étendu en 1989 par T.Coquand et C.Paulin avec les définitions inductives primitives.
Deux autres précisions semblent intéressantes: l'égalité propositionnelle usuelle de Coq est celle de Leibniz et les principes de récurrence sont ceux de Peano (récurrence struturelle).
D'autres avancées importantes telles que l'implémentation d'une procédure de décision, de tactiques d'inversion, de principe de réflexion, de modification de syntaxe etc.- ont été faites ces dernières années par l'ensemble de l'équipe Coq.
De plus, un principe d'extraction permet de récupérer en langage CAML le programme correspondant à la spécification.
Pour plus de détails voir entre autres les travaux de B.Werner [BW94], S.Boutin [SB97] et le manuel de référence [BB$^+$97].

Plan

Nous commencerons par énoncer, dans un premier temps, le théorème ainsi qu'une esquisse de sa démonstration afin de discerner les outils nécessaires à la preuve.
Dans une seconde partie, nous parlerons du cheminement ainsi que de la théorie mathématique sous-jacente.
Une troisième partie portera sur les techniques et choix d'adaptation pour Coq et leurs conséquences.
Enfin, nous ferons un commentaire détaillé sur l'annexe technique puis nous donnerons l'architecture des fichiers nécessaires.

Chapitre 2

Enoncé et preuve du théorème

2.1 Enoncé.

On prendra dans toute la suite un cercle de circonférence unité et orienté dans le sens des aiguilles d'une montre.
Un énoncé dans le langage naturel serait le suivant:

Soient N points placés consécutivement sur un cercle modulo un angle α. Alors pour tout irrationnel α et naturel N, les points partitionnent le cercle en au plus trois longueurs différentes d'intervalle.(voir figure 2.1)

Figure 2.1 : Théorème des trois intervalles.

Démontrer le théorème énoncé ci-dessus revient à montrer la formulation -plus mathématique- qui va suivre. Pour ce, définissons les notations suivantes:

· Comme nous pouvons le voir sur la figure 2.1, les points sont numérotés dans l'ordre de leur apparition; or si nous effectuons plus d'un tour de cercle, de nouveaux points viennent s'interposer entre les anciens. Il est possible alors, une fois le dernier point (N-1) placé, de les renuméroter consécutivement dans le sens des aiguilles d'une montre. Les nouveaux numéros serons notés U_j, j=1..N. U_2 sera donc le point le plus proche de 0 tandis que U_N sera le plus éloigné (en tournant toujours dans le même sens). Sur la figure 2.1, U_2=8 et U_N=15. Les U_j sont donc ainsi définis:

U_j tels que $\{U_j\alpha\}<\{U_{j+1}\alpha\}$, {} partie fractionnaire

et $\{U_1,..,U_N\}$={0,..,N-1}.

· Il nous faut maintenant donner une notion pour le successeur. Sachant que dans toute la suite nous utiliserons exclusivement la suite U_j, le successeur de U_j est donc U_{j+1}. Nous le noterons Suc(m). Sur la figure 2.1 Suc(17)=2.

Nous sommes maintenant en mesure d'énoncer le théorème que nous allons prouver:

Théorème 2.1.1

$$Suc(m) - m = \begin{cases} U_2 & si\ 0 \leq m < N - U_2 \\ U_2 - U_N & si\ N - U_2 \leq m < U_N \\ -U_N & si\ U_N \leq m < N \end{cases}$$

Si nous posons (||x||=min({x},1-{x})), cette transcription signifie que le cercle de N points est divisé en N-U_2 intervalles de longueur $\|U_2\ \alpha\|$, N-U_N intervalles de longueur $\|U_N\alpha\|$ et U_2+U_N-N intervalles de longueur $\|U_2\alpha\|+\|U_N\ \alpha\|$.

Tous les résultats sont valables pour α rationnel et irrationnel. On ne s'intéressera, ici, qu'à α irrationnel pour la raison suivante: d'un point de vue mathématique, le théorème est trivialement vrai pour α rationnel. En effet, si α=p/q alors le cercle comprend, au plus, deux longueurs d'intervalle -selon que N<q ou que N=q.

2.2 Démonstration.

Dans la suite de ce chapitre, nous allons nous attacher à prouver, de manière aussi concise que possible, le théorème2.1.1.

On remarque (lemmes intermédiaires 1 et 2) que max(U_2,U_N)<N≤U_2+U_N.

La première inégalité est due à la définition même des U_j.

La seconde se montre par l'absurde en remarquant que si N>U_2+U_N il existerait un point -U_2+U_N- entre 0 et U_2 ou entre U_N et 0, ce qui est contraire à la définition des U_j, j=2,N.

Figure 2.2 : N=U_2+U_N.

Un autre lemme intermédiaire, que l'on ne démontrera pas ici, est indispensable à la preuve:
si N=U_2+U_N alors

$$Suc(m) - m = \begin{cases} U_2 & \text{si } 0 \leq m < U_N \\ -U_N & \text{si } U_N \leq m < N \end{cases}$$

Ce qui signifie, en d'autres termes, que si l'on ne fait pas plus d'un tour de cercle il n'y a que deux longueurs d'intervalle différentes: U_N intervalles de longueur $\|U_2\alpha\|$ et U_2 de longueur $\|U_N\alpha\|$ (cf. figure2.2).

Nous pouvons maintenant énoncer l'idée principale de la démonstration de notre théorème.

Elle consiste à éliminer à partir de la suite M=U_2+U_N les i=M-N points U_2+U_N-1, U_2+U_N-2,..,U_2+U_N-i au-delà de N. Effectivement le dernier point à avoir un successeur est U_N-i-1 qui est U_N-i-1+U_2. Il ne reste donc plus que les N points originaux sur le cercle.
En remarquant que U_2(M)=U_2(N) et que U_M(M)=U_N(N) et en utilisant le lemme intermédiaire précédent on en déduit que:

pour r=N-U_2,N-U_2+1,..,U_N-1 Suc(r)=r+U_2-U_N.
Les deux autres cas sont donnés directemenent.

□

Chapitre 3

Cheminement - Théorie mathématique

Le chapitre précédent nous montre la nécessité de travailler dans le corps des nombres réels. Effectivement, ce théorème n'est à la fois étonnant et intéressant que pour α irrationnel. Néanmoins, le résultat restant vrai pour α rationnel autant se placer dans les réels.

3.1 Cheminement.

Avant de spécifier le théorème en Coq, il faut s'attarder un moment sur les outils nécessaires. Deux grandes parties se détachent alors:

- Les nombres réels.
- Les outils spécifiques au théorème.

3.1.1 Les nombres réels.

Une première étape est de déterminer de quelle sorte de réels il s'agit et comment nous allons les traiter en Coq. Pour ce, une connaissance des différentes définitions des réels s'impose, comme nous le verrons ci-après. Les références [NB53], [EL51] et [JD68] permettent de comprendre ces notions. D'autre part, il faut également remarquer que nous travaillons aussi bien dans les réels positifs que négatifs. Ceci est une remarque importante du fait que la théorie qui va en découler se fera à partir des entiers relatif -travail de Valérie Ménissier en Coq- et non à partir de la librairie de base des naturels. Une seconde étape est d'appliquer cette étude à Coq. Cette partie sera développée au chapitre 3.

3.1.2 Les outils spécifiques.

L'énoncé du théorème nous montre clairement quels sont les objets principaux nécessaires: il s'agit, d'une part, des U_j -en fait et exclusivement de U_2 et U_N- et, d'autre part, du successeur d'un point sur le cercle.

Afin de définir U_2 et U_N, nous avons eu besoin, entre autres choses, de définir la partie fractionnaire d'un nombre réels ainsi qu'un ordre total. Nous verrons ci-dessous plus en détail la démarche suivie.

3.2 Théorie mathématique.

3.2.1 Les Réels.

Pour décrire les réels deux principales possibilités se présentent à nous: nous pouvons soit faire une construction des réels, soit faire une axiomatisation.

La première méthode est basée sur la logique du *premier ordre*. Toutes les variables apparaissant dans la théorie sont supposées prendre des valeurs sur les éléments d'un ensemble fixé. Les constantes de la théorie sont les connecteurs propositionnels, la négation $\neg$, la disjonction $\vee$, la conjonction $\wedge$, l'implication $\rightarrow$, l'existentiel $\exists$, le quantificateur universel $\forall$, deux prédicats binaires ($=$ et $\neq$). Comme constantes non logiques (les symboles primitifs de la théorie) on peut choisir n'importe quelle relation qui soit définissable. Cette théorie est communément connue sous le nom de corps réels clos (real closed field). Nous pourrons nous référer à [AT69] ou à [SL71] pour plus de détails et à [EL51] pour une construction ensembliste par les coupures de Dedekind.
Néanmoins, malgré l'aspect attrayant d'une théorie du premier ordre, on ne peut, dans notre cas utiliser le real close field. En effet, cette théorie n'est *pas finiment axiomatisable*. La démonstration s'appuie sur le fait que cette théorie dit que tous les polynômes de degré impair admettent des zéros dans $\mathbb{R}$; il faudrait donc un axiome pour chaque polynôme de degré impair.

Nous avons donc choisi la deuxième méthode. Il s'agit d'une axiomatisation du *second ordre* basée sur le fait que $\mathbb{R}$ est un corps commutatif ordonné archimédien et complet. Pour ces notions nous nous sommes appuyé sur l'axiomatisation de Dieudonné [JD68]. Avec les notations suivantes:
$\#$ représente le $+$ ou le $\times$
e est l'élément neutre de l'opération en question
y est une inconnue
Faisons-en un bref rappel: voir figure 2.1

En résumé, on a:

($\mathbb{R}$,$+$) possède un élément neutre
($\mathbb{R}$,$\times$) possède un élément neutre
$+$ est commutative
$+$ est associative
$\times$ est associative
$\times$ est commutative
$\times$ est distributive / $+$

•

$$
\begin{array}{cc}
\multicolumn{2}{c}{\mathbb{R}\text{ corps commutatif}} \\
\swarrow & \searrow \\
\mathbb{R}\text{ anneau commutatif} & (\mathbb{R}^*,\times)\text{ groupe} \\
\downarrow & \downarrow \\
\mathbb{R}\text{ anneau et } \times \text{ commutative} & (\mathbb{R}^*,\times)\text{ monoïde et tout élément} \\
\text{dans } \mathbb{R} & \text{est symétrisable} \\
\downarrow & \downarrow \\
(\mathbb{R},+)\text{ groupe abélien et } \times \text{ associative} & R^+ \text{ possède un élement} \\
\text{et } \times \text{ distributive / addition} & \text{neutre, } \times \text{ associative et} \\
\downarrow & \forall b\ b\#y=y\#b=e \text{ admet} \\
\text{monoïde commutatif et} & \text{une solution} \\
\text{tout élément est symétrisable} & \\
\downarrow & \\
(\mathbb{R},+)\text{ monoïde et } + \text{ commutative} & \\
\downarrow & \\
\mathbb{R}\text{ possède un element neutre et } + \text{ associative} &
\end{array}
$$

Figure 3.1 : $\mathbb{R}$ est un corps commutatif.

$\forall b\ b\#y=y\#b=e$ admet une solution, si $\#=\times$, $b\neq 0$

• $\mathbb{R}$ est ordonné:

$<$ est le symbole utilisé pour définir l'ordre, $\leq$ s'écrit ($x<y$ ou $x=y$)
$\neg(x<x)$
$x<y$ et $y<z$ impliquent $x<z$
$x<y$ implique $\neg(y<x)$
$x\leq y$ ou $y\leq x$
$x\leq y$ implique $z+x\leq z+y$

• $\mathbb{R}$ est archimédien:

$\forall r\in\mathbb{R}\ \exists N\in\mathbb{N}$ t.q. $N>r$

• $\mathbb{R}$ est complet:

$\forall E$ partie de $\mathbb{R}$, si E est borné alors $\exists$ une borne supérieure pour E.

Il apparaît clairement maintenant que cette axiomatisation est du second ordre de par la complétude de $\mathbb{R}$.

Il semble également intéressant de noter que la majeure différence entre la construction et l'axiomatisation d'un corps est que cette première ne sert qu'à montrer l'existence des propriétés que l'on défini directement lors d'une axiomatisation. De ce fait, on peut montrer toutes les propositions de $\mathbb{R}$ si ceux-ci sont construit. Entre autres, le théorème d'Archimède est donné ici en tant qu'axiome et non en tant que lemme pour la raison énoncée ci_dessus. En fait, celui-ci peut se démontrer si l'on décide de faire une construction des réels à partir de la complétion de $\mathbb{Q}$ par exemple, mais il est indispensable -en tant qu'axiome- à toute caractérisation axiomatique des nombres réels. Une preuve de ce théorème pour une construction des réels est donnée dans [NB53, chapitre 4].

Nous reviendrons plus tard sur la transcription de cette théorie axiomatique en Coq, néanmoins nous tenons à préciser que la complétude et la décidabilité de notre axiomatisation n'a pas été montrée, quant à la correction elle est relativement claire puisqu'il suffit de vérifier que nos axiomes sont vrais dans la théorie de $\mathbb{R}$.

3.2.2 Les outils.

La partie fractionnaire

Un outil fondamental à la spécification de ce théorème est la définition de la partie fractionnaire, que nous noterons $\{\}$. Comment donc la définir grâce à notre axiomatisation de $\mathbb{R}$? Donnons-en tout d'abord plusieurs définitions mathématiques. Dans le cas des relatifs, $\{z\} = 0$. Dans celui des rationnels, $\{\frac{p}{q}\} = \frac{p \ mod \ q}{q}$. Dans le cas d'un réel, elle est définie telle que, si [] représente la partie entière, $r = [r] + \{r\}$. D'où $\{r\} = r - [r]$. Définissons maintenant la partie entière.
La partie entière d'un nombre positif est clairement définie. Par exemple $[3,14] = 3$. Par contre, plusieurs définitions peuvent être prises pour celle d'un nombre négatif. Ici nous utiliserons la définition la plus courante. Par exemple $[-3,14] = -4$. Pour ce faire, nous avons concrètement besoin de ppNsup qui représente le plus petit entier relatif supérieur strictement à r (un réel). L'existence d'un tel entier provient du fait que $\mathbb{R}$ est archimédien et que tout ensemble de relatifs ordonné et minoré admet un plus petit élément.
ppNsup peut être ainsi caractérisé:

$\forall r \in \mathbb{R}\ (ppNsup\ r) > r\ et\ (ppNsup\ r) - r \leq 1$

On a bien alors:

$r < (ppNsup\ r) \leq r + 1$

La définition de la partie entière pEr est immédiate:

$pEr(r) = (ppNsup\ r) - 1$

puis celle de la partie fractionnaire:

$part\_frac(r) = r - pEr(r)$

Outils spécifiques

Une autre définition particulièrement importante pour la spécification et la preuve du théorème est celle des U_j. Plus exactement, on se restreindra à la définition de U_2 et U_N qui sont en fait les deux seuls points utilisés. Nous rappelons que, une fois les N points placés sur le cercle, U_2 est le point le plus proche de 0 dans le sens des aiguilles d'une montre et U_N dans le sens trigonométrique. $U_2(N)$ et $U_N(N)$ appartiennent donc à l'intervalle $[0, N[$ et sont respectivement tel que:

$\forall i \in [1..N]\ \{U_2\alpha\} \leq \{U_i\alpha\}$ et

$\forall i \in [1..N]\ \{U_i\alpha\} \leq \{U_N\alpha\}$

Leur existence est intuitive de par leur intervalle d'appartenance.
Afin de formaliser ces deux définitions, nous avons défini l'ordre total sur les naturels suivant:

$ordre\_total(n, m) = \{n\alpha\} \leq \{m\alpha\} \qquad \alpha \in [0, 1[\subset\mathbb{R}$

Deux méthodes peuvent alors être utilisées:

· celle qui consiste à montrer l'existence des U_j puis à les définir par cas (ce qui est fait actuellement);

· celle qui consiste à définir un principe de minimisation sur les naturels selon un ordre approprié (cf. exemple 3.2.1) puis de dire,par exemple, que U_2 est le minimum sur l'ensemble $[0, N[$ muni de l'ordre en question -*ordre_total* ici (spécifications entre parenthèse dans le fichier tools.v). Nous verrons au chapitre 4 les avantages et inconvénients de cette stratégie en Coq.

Exemple 3.2.1 *Principe de minimisation sur les naturels*
$\forall Predicat\ P\ decidable \forall n : nat P(n) \Rightarrow \exists m.(P(m) \wedge \forall i : nat P(i) \Rightarrow i \succ m)$
où $\succ$ dénote l'ordre nécessaire.

Chapitre 4

Choix pour la spécification en Coq

Comme nous l'avons dit dans l'introduction lors de la brève présentation de Coq, nous avons trois possibilités pour définir un ensemble tel que $\mathbb{R}$. Nous allons, avant toute chose, nous attarder sur des différences plus fondamentales, que celles énoncées en introduction, entre Prop, Set et Type. Deux critères de comparaison nous intéressent surtout, l'imprédicativité et l'élimination forte.
Dans toute la suite la syntaxe utilisée sera celle de Coq, en rappelant que la notation {A}+{B} est employée pour désigner un OU dont le résultat est dans Set, A et B étant des propositions, tout comme la notation {n:S|A} désigne un IL EXISTE où S est un Set, A une proposition. On précise également qu'une preuve de {A}+{B} est Left ou Right.

4.1 Prop, Set et Type

4.1.1 Prop

Prop est une sorte imprédicative c'est-à-dire où les types peuvent être polymorphes (cf. exemple 4.1.1).

Exemple 4.1.1 *Imprédicativité.*
Le type T:(P:Prop)P->P est de type Prop:

Coq < Check (X:Prop)X->X.
(X:Prop)X->X
: Prop

Prop ne contient pas l'élimination forte (cf. exemple 4.1.2). En effet, le principe d'élimination est relatif à la construction d'une propriété ou d'un type par cas sur une structure. L'élimination forte n'est donc pas possible sur un A:Prop car on pourrait alors montrer qu'il existe a,b:A avec a≠b ce qui rendrait la logique classique incohérente (comme nous pourrons le voir plus loin).

Exemple 4.1.2

```
Coq < Lemma elim_prop:(n,m:nat)((n=m)\/(~(n=m)))->{n=m}+{~n=m}.
1 subgoal

  ============================
   (n,m:nat)n=m\/~n=m->{n=0}+{~n=0}

elim_prop < Intros.
1 subgoal

  n : nat
  m : nat
  H : n=m\/~n=m
  ============================
   {n=0}+{~n=0}

elim_prop < Elim H.

Toplevel input, characters 0-7:
> Elim H.
> ^^^^^^^
Error: or_rec not declared
```

Revenons un instant sur l'imprédicativité. Notons l'interprétation d'un type A par |A|. Un modèle naïf serait de prendre $|A\text{->}B|=|B|^{|A|}$ (toutes les fonctions de |A| dans |B|). Cette interprétation implique une incohérence si l'on a de l'imprédicativité puisque par exemple on aurait *nat->nat* $\in$ *nat*. Néanmoins, bien que Prop soit imprédicative, on peut montrer le théorème suivant.

Théorème 4.1.1 *Prop est cohérent.*

Preuve *idée:*
On utilise le modèle de proof irrelevance*: une proposition (P:Prop) sera interprétée par sa* valeur de vérité *i.e. si P est vraie* $|P|=\{o\}$ *si P est faux* $|P|=\emptyset$ *où* $\{o\}$ *désigne un singleton.*

□

On a alors, le résultat intéressant suivant sur le tiers exclus (EM).

Corollaire 4.1.1 *EM est cohérent dans Prop.*

Preuve
$EM = \forall A : Prop\ (A \vee \tilde{}A)$
Si A est vraie alors $|EM|=\{o\}$
Si A est fausse alors $|EM|=\emptyset \rightarrow \emptyset=\{o\}$

□

4.1.2 Set

Set est une sorte imprédicative.
Set contient l'élimination forte (cf. exemple 4.1.3).

Exemple 4.1.3

```
Coq < Lemma elim_set:(n,m:nat){n=m}+{~n=m}->{n=0}+{~n=0}.
1 subgoal

  ============================
   (n,m:nat){n=m}+{~n=m}->{n=0}+{~n=0}

elim_set < Intros.
1 subgoal

  n : nat
  m : nat
  H : {n=m}+{~n=m}
  ============================
   {n=0}+{~n=0}

elim_set < Elim H.
2 subgoals

  n : nat
  m : nat
  H : {n=m}+{~n=m}
  ============================
   n=m->{n=0}+{~n=0}

subgoal 2 is:
 ~n=m->{n=0}+{~n=0}

..etc.
```

Or, l'élimination forte implique le fait que $0 \neq 1$ -il s'agit du principe de la tactique *Discriminate*. Il est alors possible de montrer le théorème suivant.

Théorème 4.1.2 *EM est contradictoire dans Set.*

Preuve *Idée:*
Il s'agit de montrer ~EM. Or Pour tout système polymorphe, on peut montrer que EM->0=1. Mais, dans Set 0≠ 1 donc on a bien une contradiction.

□

En fait, il y a contradiction lorsque l'on ne peut avoir la propriété de décidabilité. Le contre-exemple 4.1.4 met en relief cet aspect.

Exemple 4.1.4 *Contre-exemple: une fonction non calculable.*
Soit $A : Set \rightarrow 1$ *si A est habité*
$\rightarrow 0$ *si A est vide*
A est donc une fonction de Set→Bool donc en particulier de nat→Prop telle que:
$0 \rightarrow False$
$1 \rightarrow True$
Or de par l'imprédicativité et la non décidabilité on a 0=1 ce qui implique que False=True et on peut en déduire alors que l'on ne sait pas interpréter (dans Set) les fonctions non calculables.

Pour cette raison, les modèles de Set sont tels que:
|A→ B|=ensemble des fonctions calculables de |A| dans |B|.
Il s'agit de la réalisabilité.

4.1.3 Type

Type est une sorte prédicative et est basée sur un modèle ensembliste. En d'autres termes, on a $|A\text{->}B|=|B|^{|A|}$. Donc, pour un tel modèle:

EM est cohérent dans Type.

Revenons sur la prédicativité de Type. C'est grâce aux *univers* que, malgré *l'apparence* de ce qu'affiche Coq (cf. exemple 4.1.5), nous avons la prédicativité. Le paradoxe de Girard [YG72] montre que la règle *Type:Type* permet de typer des termes non normalisables (et que la théorie des types correspondante est incohérente). Dans Coq, les univers sont cachés. En fait, conceptuellement, l'exemple 4.1.5 s'écrit $Type{:}Type_1$ et ainsi de suite: $Type_i{:}Type_{i+1}$.

Exemple 4.1.5

```
Coq < Check Type.
Type
     : Type
```

Il n'y a pas d'élimination forte dans Type. Nous verrons comment pallier ce problème au paragraphe ci-dessous.

En résumé:

· Prop est imprédicative et n'a pas l'élimination forte,
son modèle est la proof irrelevance;
· Set est imprédicative et a l'élimination forte,
son modèle est la réalisabilité;
· Type est prédicative et n'a pas l'élimination forte,
son modèle est ensembliste.

4.2 Les choix

Dans notre axiomatisation, nous avons donc le choix entre trois solutions:

Variable R:Prop
Variable R:Set
Variable R:Type

Nous avons opté pour la dernière principalement pour la raison suivante: nous avons vu au chapitre précédent que nous avions besoin de l'existence de la fonction *partie entière*. Or, vu l'absence de type algébrique pour $\mathbb{R}$ celle-ci ne peut être définie récursivement. Nous utilisons alors la fonction ppNsup qui, rappelons-le, donne le plus petit relatif strictement supérieur à un r. De plus, elle n'est pas constante donc discontinue et donc *non calculable*. Dans ce cas, comme nous l'avons vu dans la section précédente, le fait de poser R:Type semble judicieux.
D'autre part, à partir du moment où nous pensons avoir une théorie cohérente, nous somme sûrs, alors, que même si un de nos axiomes implique EM notre théorie reste cohérente. En fait, il y a contradiction pour EM lorsque l'on ne peut avoir la propriété de décidabilité.
Par exemple, (n,m:nat){n=m}+{~n=m} n'est pas contradictoire mais (p,q:R){p=q}+{~p=q} l'est car l'égalité dans R n'est pas décidable. En règle générale, dès que l'on travaille avec des fonctions calculables (continues) on peut se placer sans risque dans Set car l'on sait interpréter ces fonctions (ce n'est pas notre cas).
On peut se demander effectivement pourquoi (n:nat){n=0}+{~n=0} n'implique pas EM et n'est donc pas un lemme de la logique classique alors que (r:R){r=0}+{~r=0} est une instanciation du tiers exclus? Pour la même raison qui est due à la calculabilité (donc à la décidabilité). Dans **nat**, on peut décider du fait que n=0 ou n≠0 sans pour autant que ce soit une instanciation de EM, ce qui n'est pas le cas dans **R**. Il faudrait pour cela prendre une théorie des réels calculables (cf.[VM94]).
Nous pouvons remarquer que la solution choisie est satisfaisante à condition de ne pas monter trop haut dans la hierarchie des univers, ce que nous respectons.

Cette dernière solution nous conduit à effectuer un autre choix, celui de la skolémisation (cf. exemples 4.2.1 et 4.2.2). Ce choix tient au fait que nous ne disposons pas dans Type de l'élimination forte et donc, on ne peut définir un IL EXISTE constructif -en vue de récupérer le témoin par un case.

Exemple 4.2.1 *Skolémisation d'un* ou.
L'idée est de remplacer (x:R) {x=0}+{~x=0}
par

Variable f:R->bool.
Axiom n1:(x:R)(f x)=true<->(x=0).
Axiom n2:(x:R)(f x)=false<->(~x=0).

Exemple 4.2.2 *Skolémisation d'un* il existe.
L'idée est de remplacer {r:R | (gtR r 0R)}
par
Variable pos:R.
Axiom n:(gtR pos 0R).

Il ne s'agit pas d'une autre théorie mathématique sinon d'une autre formulation. En effet, replaçons-nous dans Set un instant. Le fait de skolémiser et de ne pas définir explicitement un { }+{ } ou un { | } pourrait laisser croire à une disparition du risque de contradiction. Ce n'est pas le cas. Comme nous le montrons dans l'exemple 3.3, la forme skolémisée implique la forme usuelle constructive qui, comme nous le savons, implique le tiers exclus.

Exemple 4.2.3

```
Coq < Variable R:Set.
R is assumed

Coq < Variable P:R->Prop.
P is assumed

Coq < Variable f:R->bool.
f is assumed

Coq < Axiom p1:(x:R)(f x)=true<->(P x).
p1 is assumed

Coq < Axiom p2:(x:R)(f x)=false<->~(P x).
p2 is assumed

Coq < Goal (x:R){(P x)}+{~(P x)}.
1 subgoal

  ============================
   (x:R){(P x)}+{~(P x)}

Unnamed_thm < Intro;Cut {(f x)=true}+{(f x)=false}.
2 subgoals

  x : R
  ============================
   {(f x)=true}+{(f x)=false}->{(P x)}+{~(P x)}

subgoal 2 is:
 {(f x)=true}+{(f x)=false}

Unnamed_thm < Induction 1;Intros.
```

```
3 subgoals

  x : R
  H : {(f x)=true}+{(f x)=false}
  y : (f x)=true
  ============================
   {(P x)}+{~(P x)}

subgoal 2 is:
 {(P x)}+{~(P x)}
subgoal 3 is:
 {(f x)=true}+{(f x)=false}

Unnamed_thm < Left;Elim p1 with x;Auto.
2 subgoals


  x : R
  H : {(f x)=true}+{(f x)=false}
  y : (f x)=false
  ============================
   {(P x)}+{~(P x)}

subgoal 2 is:
 {(f x)=true}+{(f x)=false}

Unnamed_thm < Right;Elim p2 with x;Auto.
1 subgoal

  x : R
  ============================
   {(f x)=true}+{(f x)=false}

Unnamed_thm < Elim (f x);Auto.
Subtree proved!
```

La technique de skolémisation nous permet donc de contourner le manque d'élimination forte dans Type. En l'occurrence, elle permet de remplacer les destructurations, afin de récupérer le témoin d'un il existe, qui ne sont possibles que dans Set (cf. exemple 4.2.4). Elle permet également une utilisation plus simple du symbole aussi bien lors d'une spécification que lors d'une preuve (en utilisant directement l'axiome). Elle présente toutefois un inconvénient, celui de rallonger la liste des variables (ex.:suparch et ppNsup).

Exemple 4.2.4

```
Coq < Hypothesis trivial:{n:nat|(gt n O)}.
trivial is assumed
```

```
Coq < Definition un_n:=(Cases (trivial) of
Coq <                      (exist a b)=>a
Coq <                    end).
un_n is defined

Coq < Check un_n.
un_n
     : nat
```

Chapitre 5

Commentaires sur l'annexe

Vue le nombre d'axiomes et de lemmes, nous ne parlerons que de ceux qui nous semblent requérir une attention spéciale, aussi bien d'un point de vue théorique que pour une élaboration de preuve Coq.

5.1 Axiomatisation.

Dans toute le suite nous utiliserons le mot *axiome* pour dire *axiom* ou *lemma* sauf mention explicite.
Nous commencerons par donner une signification aux noms de variables:

```
Variable R:Type.
Variable OR:R.
Variable IR:R.
Variable addR:R->R->R.
Variable multR:R->R->R.
Variable ltR:R->R->Prop.
Variable unminus:R->R.
Variable inver:R->R.
Variable suparch:R->nat.
```

`R` est $\mathbb{R}$
`OR` est le `0` des $\mathbb{R}$
`IR` est le `1` des $\mathbb{R}$
`addR` est l'addition dans $\mathbb{R}$, i.e. elle prend deux réels et rend un réel
`multR` est la multiplication dans $\mathbb{R}$
`ltR` est $<$ dans $\mathbb{R}$
`unminus` est le moins unaire -r (ou opposé)
`inver` est l'élément inverse r^{-1}
`suparch` est l'entier du théorème d'Archimède

Notre axiomatisation de $\mathbb{R}$ en Coq est divisée en cinq parties: les axiomes de base classiques, les axiomes de corps commutatif, les axiomes définissant le relation d'ordre, l'axiome pour le théorème d'Archimède et celui pour la complétude. Nous noterons qu'elle contient des axiomes qui ne sont pas forcément nécessaires à la spécification de $\mathbb{R}$ puisqu'ils sont conséquence des autres. Beaucoup ont été ajoutés au fur et à mesure selon les besoins.

Les axiomes de base classiques définissent, en quelque sorte, une décidabilité propositionnelle pour $\mathbb{R}$.

Les axiomes de corps transcrivent la théorie énoncée au chapitre 2. Un axiome intéressant est l'axiome *base0*. A quoi sert-il? Il traduit le fait que ($\mathbb{R}^*$,$\times$) est un groupe et que $\mathbb{R}$ est un anneau. Ces deux propriétés font que ($\mathbb{R}$,+) et que ($\mathbb{R}^*$,$\times$) possèdent un élément neutre et donc que 0 (élément neutre de ($\mathbb{R}$,+)) doit être différent de 1 (élément neutre de ($\mathbb{R}^*$,$\times$)).
La définition du carré servira par la suite pour montrer que $0 < 1$.
Une preuve intéressante -bien que usuelle- est celle de mult_Od qui dit que $\forall$r $r \times 0 = 0$. On utilise les éléments neutres de l'addition et de la multiplication, la distributivité et que si r+b=r alors b=0, ce qui peut se résumer par la démonstration, en langage mathémathique, suivante:

$$r \times (1+0) = r + r \times 0$$

$$r = r + r \times 0$$

$$r \times 0 = 0$$

Les axiomes d'ordre permettent de spécifier ltR. leR, gtR et geR sont définis à partir de ltR. Puisque **R** n'est pas défini ici comme un type algébrique, nous ne pouvons comme pour **nat**, définir *lt* à partir de *le* en utilisant une propriété similaire à (lt n m)$\leftrightarrow$(le n (S m)). Ici, (leR p q)$\equiv$(ltR p q)\/p=q -en d'autres termes p$\leq$q $\leftrightarrow$ p<q ou p=q. (gtR p q) est défini comme étant (ltR q p).

Les axiomes d'archimède et de complétude tanscrivent exactement leur définition mathématique.

Nous noterons que si l'on veut comparer un **nat** et un **R**, aucune des définitions de lt (lt ou ltR) ne le permet. Il faut donc une injection de $\mathbb{Z}$ dans $\mathbb{R}$. Cette injection est définie à partir de celle de $\mathbb{N}$ dans $\mathbb{R}$ et du type inductif des **Z** spécifiés par Valérie Ménissier-Morain.
L'injection récursive de $\mathbb{N}$ dans $\mathbb{R}$ filtre le type inductif **nat** et rend comme résultat un réel:

```
INR n= O       => OR
      |(S O)   => IR
      |(S n)   => (INR n)+IR
```

L'injection de $\mathbb{Z}$ dans $\mathbb{R}$ n'est pas inductive et rend selon le filtrage du type **Z**:

```
IZR z= OZ       => OR
      |(pos n) => (INR n)+IR
      |(neg n) => -(INR n)-IR
```

On remarque que pour `INR`, le filtrage sur la cas `(S O)` peut sembler superflu -notons que d'un point de vue théorique il l'est. Par contre, lors de l'utilisation de `INR` pour des preuves en Coq, il est beaucoup plus simple d'avoir directement `(S O) => IR` que `(S O) => (INR O)+IR` qu'il faudrait ensuite réécrire en `IR`.

Un lemme intéressant à montrer de par sa relative simplicité théorique et sa relative difficulté en Coq est Z_R_minus:
Notons (IZR z) par ≪z≫ et (INR n) par <n>.
On a
≪z1≫ $-_{\mathbb{R}}$ ≪z2≫= ≪z1 $-_{\mathbb{Z}}$ z2≫
On montre d'abord ≪z≫ $+_{\mathbb{R}}$ ≪t≫= ≪z $+_{\mathbb{Z}}$ t≫ (1) puis on remplace t par -t en montrant que -≪t≫=≪-t≫.
Pour prouver (1) on décompose par cas sur z puis pour chaque cas, à nouveau par cas sur t (2 inductions).
Prenons par exemple le cas (pos z) et (pos t):

$$\ll pos(k) \gg + \ll pos(t) \gg = \ll pos(k) + pos(t) \gg$$

$$< k > +1 + < t > +1 = \ll pos(k+t+1) \gg$$

$$< k > + < t > +1 + 1 = < k+t+1 > +1$$

et enfin, en utilisant la propriété <n>+<m>=<n+m> que l'on aura pris soin de démontrer auparavant:

$$< k > + < t > +1 + 1 = < k > + < t > +1 + 1$$

Cette preuve nous incite à formuler une remarque: même s'il est en général simple d'écrire une preuve sur le papier, il est parfois beaucoup plus fastidieux de l'exécuter en Coq. Effectivement, certaines preuves nécessitent une grande quantité de réécriture. Par exemple (<k>+1)+(<t>+1) ne se réécrit pas rapidement en ((<k>+<t>)+1)+1. Il faut au minimum quatre étapes:

$$(a+c)+(b+c)$$

$$\downarrow add\_commu$$

$$(c+a)+(b+c)$$

$$\downarrow add\_assoc$$

$$c+(a+(b+c))$$

$$\downarrow add\_assoc$$

$$c+((a+b)+c)$$

$$\downarrow add\_commu$$

$$((a+b)+c)+c$$

Nous verrons dans les travaux futurs comment simplifier ces raisonnements égalitaires.

5.2 Le théorème.

La partie complément pour les naturels comprend surtout des lemmes techniques sans grand intérêt mais utiles pour la suite. Un seul est important; il s'agit de *princip_min* dont nous avons parlé au chapitre 2. Ici, l'ordre utilisé est l'ordre usuel sur les entiers naturels. On énonce le fait que si l'on a un prédicat décidable -i.e. si l'on sait dire si un n appartient à un ensemble-alors cet ensemble contient un plus petit élément (selon l'ordre en question). Une preuve peut être faite par récurrence sur n avec comme hypothèse de récurrence:

```
Ex [m:nat](P m)/\((i:nat)(P i)->(ge i m)))
         \/((p:nat)(ge n p)->~(P p))
```

Ce qui signifie que soit il existe un tel n minimal, soit que tous les p p$\leq$n n'appartiennent pas à l'ensemble. On introduit cette hypothèse grâce à un *Cut* puis on commence la récurrence grâce à un *Elim n*.
Ce principe peut se démontrer également de manière constructive (Martin Löf). Ceci restera à explorer dans un futur proche.

Le lemme *inser2_trans* nous sert à prouver le théorème des trois intervalles, à partir des preuves intermédiaires 4. Nous verrons plus loin la preuve (très simple) en détail.

En ce qui concerne les outils, revenons sur l'intérêt de définir un principe de minimisation. L'intérêt augmente avec la fréquence des définitions nécessitant une propriété de minimisation. Ceci reste d'autant plus vrai si l'ordre utilisé ne change pas et si l'on se place sur des ensembles de même type. Par exemple, dans notre cas, nous pensons qu'il serait possible de définir *ppNsup* grâce à un principe de minimisation sur un ensemble de relatifs minoré et basé sur l'ordre habituel des relatifs. Nous pensons également qu'il est possible de définir U_2 de manière analogue en prenant un ensemble de naturels et l'ordre *ordre_total* dont nous avons déjà parlé. De même pour U_N mais cette fois il s'agirait d'un principe de maximisation sur un ensemble d'entiers majoré et avec *ordre_total*. Donc, pour définir trois objets nous avons besoin de trois principes différents.
Néanmoins, l'idée reste à étudier car il est peut-être possible de montrer un principe plus général pouvant être utilisé dans nos trois cas.

Attardons-nous un instant sur la preuve intermédiaire 4. Les lemmes *three_gap1, 2 et 3* représentent respectivement les trois cas de théorème énoncé au chapitre 1 :
$0 \leq n < N - U_2$
$N - U_2 \leq n < U_N$
$U_N \leq n < N$
La variable *after* représente le *Suc* (sous forme skolémisée) et est définie dans preuve intermédiaire 3.

Détaillons maintenant la preuve de théorème. En d'autres termes, *three_gap* revient à prouver l'exhaustivité des cas énoncés dans preuve 4. Le théorème

```
Lemma three_gap:(N:nat)(n:nat)((le 0 n)/\(lt n N))->
      {(minusZ (INZ (after n)) (INZ n))=(INZ (U2 N))}+
 {(minusZ (INZ (after n)) (INZ n))=(minusZ (INZ (U2 N)) (INZ (UN N)))}+
      {(minusZ (INZ (after n)) (INZ n))=(oppZ (INZ (UN N)))}.
```

s'écrit donc:

$\forall n \forall N \in [0, N[$ on a soit:
$Suc(m) - m = U_2$
$Suc(m) - m = U_2 - U_N$
$Suc(m) - m = -U_N$

ce qui équivaut bien à dire que l'on a au plus trois longueurs d'intervalle différentes, qui est l'énoncé initial du *théorème des trois intervalles*.

Prouvons maintenant le théorème 5.2.2 à l'aide du théorème 5.2.1.

Théorème 5.2.1

$Lemme1 = 0 \leq n < a \rightarrow A$
$Lemme2 = a \leq n < b \rightarrow B$
$Lemme3 = b \leq n < c \rightarrow C$

Théorème 5.2.2
$Lemme = 0 \leq n < c \rightarrow \{A\} + \{B\} + \{C\}$

On montre, dans un premier temps, que
$\{0 \leq n < a\} + \{a \leq n < b\} + \{b \leq n < c\} \rightarrow \{A\} + \{B\} + \{C\}$
ce qui est simple en destructurant le terme hypothèse (*Elim*) puis en choisissant la bonne proposition de conclusion (*Left* ou *Right*).
Il reste à montrer que
$0 \leq n < c \rightarrow \{0 \leq n < a\} + \{a \leq n < b\} + \{b \leq n < c\}$.
C'est ici qu'intervient *inser2_trans*:
$n \leq m < t \rightarrow \{n \leq m < q\} + \{q \leq m < p\} + \{p \leq m < t\}$.

Chapitre 6

Architecture

La figure ci-dessous représente l'architecture des modules nécessaires à la preuve du théorème, ce qui permet une visualisation plus rapide du cheminement.
Les bibliothèques internes à Coq (contrib ou pas) sont notées entre parenthèses.

$$
\begin{tikzcd}
\text{(Z)} & & & & \text{(Arith,Compare\_dec)} \\
\text{Z\_compl} \arrow[u] & & & & \text{Nat\_compl} \arrow[u] \\
\text{Rbase} \arrow[u] & & & & \\
& & \text{tools} \arrow[ull] \arrow[uurr] & & \\
& \text{preuve1} \arrow[ur] & \text{preuve2} \arrow[u] & \text{preuve3} \arrow[ul] & \\
& & \text{preuve4} \arrow[ul] \arrow[u] \arrow[ur] & & \\
& & \textbf{three\_gap} \arrow[u] & &
\end{tikzcd}
$$

Chapitre 7

Conclusion

Comme nous pouvons le constater, beaucoup de preuves sont inachevées. Nous nous proposons donc, dans un futur proche, de finir la démonstration des preuves intermédiaires de ce théorème. Toutefois, nous devons avant tout nous pencher sur la meilleur façon de spécifier les outils nécessaires au théorème (partie entière,U_j,...), ce qui peut avoir une très grande importance sur la difficulté des preuves en Coq à venir. Certaines de nos spécifications pourraient donc être modifiées si les conclusions de cette étude le montraient nécessaire.

Il est également intéressant de remarquer que ce théorème, bien qu'énoncé sous forme de conjecture dans les années 50, ne trouva de démonstration satisfaisante que vers le milieu des années 80. Il est intéressant de noter que la démonstration telle que la faite Tony Van Ravenstein ne contient aucun résultat particulier des mathématiques, ce qui la rend d'autant plus faisable en Coq. Par contre, l'astuce de Tony Van Ravenstein consiste à transformer l'énoncé du théorème comme nous l'avons expliqué au chapitre 1, i.e. à ne pas essayer de montrer qu'il y a au plus trois longueurs d'intervalle différentes mais à montrer un résultat beaucoup plus fort qui consiste à donner les longueurs des trois intervalles et leur emplacement par rapport aux points sur le cercle.

Une question relativement importante dont nous avons sous-entendu la réponse dans ce rapport, consiste à se demander si ce théorème est prouvable en Coq.
Il semble assez claire que dans le cadre classique où nous nous trouvons, le théorème est prouvable en Coq. Ce système est, en effet, adapté à formaliser la plupart des notions mathématiques -quelques réserves pouvant être émises lorsque l'on utilise les grands ordinaux ou autres grands ensembles.
Par contre, dans un cadre constructif le doute subsiste ne serait-ce qu'à cause de la formalisation de la notion de cercle et de la partie entière qui nécessitent l'introduction de fonctions non calculables.

La spécification du théorème des trois intervalles en Coq a donc soulevé et soulève quantité de problèmes intéressants qui, comme nous le disions en introduction, peuvent apporter de grands éclaircicements pour de futurs travaux (réels calculables ou non par exemple...).

Annexe A

Compléments pour les Relatifs

```
(*******************************************************)
(*             Complements pour les Relatifs           *)
(*                                                     *)
(*******************************************************)

Require Export Z.
(*******************************************************)

(********************************************************)
(*      Operateurs supplementaires                      *)
(********************************************************)

(**********)
Definition minusZ:Z->Z->Z:=[z1,z2:Z](addZ z1 (oppZ z2)).

(**********)
Definition geZ:Z->Z->Prop:=[z1,z2:Z](leZ z2 z1).

(**********)
Definition gtZ:Z->Z->Prop:=[z1,z2:Z](ltZ z2 z1).

(********************************************************)
(*      Proprietes supplementaires                      *)
(********************************************************)

(**********)
Lemma z_add_add:(z,z1,z2:Z)(addZ z z1)=(addZ z z2)->(z1=z2).
(Intros;
 Apply (regular_l Z [x:Z]True addZ OZ oppZ Z_group z1 z2 I I z);
 Auto).
Save.
```

```
(**********)
Lemma tech_minus_neg_succZ:(x,y:nat)(lt y x)->
  (neg (minus x (S y)))=(succZ (neg (minus x y))).
(Intros;
 Cut (addZ (neg (S y)) (neg (minus x (S y))))
      =(addZ (neg (S y)) (succZ (neg (minus x y)))); Intros).
Apply (z_add_add (neg (S y)) (neg (minus x (S y)))
        (succZ (neg (minus x y))) H0).
Rewrite -> tech_add_neg_negZ.
Rewrite <- le_plus_minus.
Rewrite -> tech_add_neg_predZ.
Cut (addZ (neg y) (succZ (neg (minus x y))))
     =(addZ (succZ (neg (minus x y))) (neg y)).
(Intro; Rewrite -> H0).
Rewrite <- pred_addZ_l.
Rewrite -> pred_succZ.
Rewrite -> tech_add_neg_negZ.
Rewrite -> plus_sym.
(Rewrite <- le_plus_minus; Auto).
(Elim (addZ_commutativity (neg y) (succZ (neg (minus x y)))); Auto).
Auto.
Save.

(**********)
Lemma tech_add_pos_neg_negZ:(n1,n2:nat)
  (lt n1 n2)->(addZ (pos n1) (neg n2))=(neg (minus n2 (S n1))).
(Induction n1; Intros).
Simpl.
Inversion H.
(Inversion H; Auto).
(Rewrite -> tech_minus_neg_succZ; Auto).
Rewrite -> addZ_eq3.
Rewrite -> tech_minus_neg_succZ.
(Rewrite -> H;Auto).
Auto.
Save.

(*******************************************************)
(*        Injection de N dans Z                        *)
(*******************************************************)

(**********)
Definition INZ:nat->Z:=[n:nat](Cases n of
                                 O     => OZ
                                |(S n) => (pos n)
                               end).
```

Annexe B

Axiomatisation des Réels

```
(*******************************************************)
(*            Axiomatisation des Reels                 *)
(*                                                     *)
(*******************************************************)

Require Export Z_compl.

Variable R:Type.
Variable OR:R.
Variable IR:R.
Variable addR:R->R->R.
Variable multR:R->R->R.
Variable ltR:R->R->Prop.
Variable unminus:R->R.
Variable inver:R->R.
Variable suparch:R->nat.
(*******************************************************)

(**********)
Definition gtR:R->R->Prop:=[r1,r2:R](ltR r2 r1).

(*******************************************************)
(*      Axiomes de base Classiques                     *)
(*******************************************************)

(**********)
Axiom base1:(r1,r2:R)((r1==r2)\/(~(r1==r2))).
Hint base1.

(**********)
Axiom base2:(r1,r2:R)((ltR r1 r2)\/(r1==r2)\/(gtR r1 r2)).

(**********)
```

```
Lemma base3:(r1,r2:R)(~(r1==r2))->((ltR r1 r2)\/(gtR r1 r2)).
(Intros; Case (base2 r1 r2); Intros).
(Left; Exact H0).
(Case H0; Intros).
(Absurd r1==r2; Auto).
(Right; Exact H1).
Save.

(*********************************************************)
(*       Axiomes de corps                                *)
(*********************************************************)
(*********************************************************)
(*       Element neutres de add et mult                  *)
(*********************************************************)

(**********)
Axiom base0:(~IR==OR).

(**********)
Axiom add_en:(r:R)(((addR r OR)==r)/\((addR OR r)==r)).
Hint add_en.

(**********)
Axiom mult_en:(r:R)(((multR r IR)==r)/\((multR IR r)==r)).
Hint mult_en.

(*********************************************************)
(*       Addition                                        *)
(*********************************************************)

(**********)
Axiom add_commu:(r1,r2:R)((addR r1 r2)==(addR r2 r1)).

(**********)
Axiom add_assoc:(r1,r2,r3:R)
  ((addR (addR r1 r2) r3)==(addR r1 (addR r2 r3))).

(**********)
Axiom pos_neg:(r:R)((addR r (unminus r))==OR).
Hint pos_neg.

(**********)
Lemma neg_pos:(r:R)((addR (unminus r) r)==OR).
(Intro; Rewrite -> add_commu; Auto).
Save.

(**********)
```

```
Lemma oppose:(x,y:R)((addR x y)==OR)->(y==(unminus x)).
(Intros; Replace y with (addR (addR (unminus x) x) y)).
Rewrite -> add_assoc.
Rewrite -> H.
(Elim (add_en (unminus x)); Auto).
Rewrite -> neg_pos.
(Elim (add_en y); Auto).
Save.

(**********)
Lemma add_add_r:(r,r1,r2:R)(r1==r2)->((addR r r1)==(addR r r2)).
(Intros; Apply (congr_eqT R R (addR r) r1 r2 H)).
Save.
Hint add_add_r.

(**********)
Lemma r_add_add:(r,r1,r2:R)((addR r r1)==(addR r r2))->(r1==r2).
(Intros;
 Cut (addR (addR (unminus r) r) r1)==(addR (addR (unminus r) r) r2)).
Rewrite -> neg_pos.
(Replace (addR OR r1) with r1; Replace (addR OR r2) with r2; Auto).
(Elim (add_en r2); Auto).
(Elim (add_en r1); Auto).
(Elim (add_en r2); Auto).
(Rewrite -> add_assoc; Rewrite -> add_assoc).
(Rewrite <- H; Auto).
Save.

(**********)
Lemma add_en_i:(r,b:R)((addR r b)==r)->(b==OR).
(Intros r b; Replace r with (addR r OR)).
(Rewrite -> add_assoc; Replace (addR OR b) with b).
(Intro; Apply (r_add_add r b OR); Auto).
(Elim (add_en b); Auto).
(Elim (add_en r); Auto).
Save.

(*******************************************************)
(*       Multiplication                                *)
(*******************************************************)

(**********)
Axiom mult_commu:(r1,r2:R)((multR r1 r2)==(multR r2 r1)).
Hint mult_commu.

(**********)
Axiom mult_assoc:(r1,r2,r3:R)
```

```
  ((multR (multR r1 r2) r3)==(multR r1 (multR r2 r3))).
Hint mult_assoc.

(**********)
Axiom inv1:(r:R)(~(r==OR))->((multR (inver r) r)==IR).

(**********)
Lemma inv2:(r:R)(~(r==OR))->((multR r (inver r))==IR).
(Intros; Rewrite -> mult_commu; Apply (inv1 r H); Auto).
Save.

(**********)
Axiom distri:(r1,r2,r3:R)
  ((multR r1 (addR r2 r3))==(addR (multR r1 r2) (multR r1 r3))).
Hint distri.

(**********)
Lemma mult_Od:(r:R)((multR r OR)==OR).
(Intro; Cut (multR r (addR IR OR))==(addR r (multR r OR))).
(Replace (addR IR OR) with IR; Replace (multR r IR) with r).
(Intro;
 Apply (add_en_i r (multR r OR) (sym_eqT R r (addR r (multR r OR)) H))).
(Elim (mult_en r); Auto).
(Elim (add_en IR); Auto).
(Elim (mult_en r); Auto).
Cut r==(multR r IR).
(Intro; Pattern 2 r; Rewrite -> H; Apply (distri r IR OR)).
(Elim (mult_en r); Auto).
Save.
Hint mult_Od.

(**********)
Lemma mult_Og:(r:R)((multR OR r)==OR).
(Intro; Rewrite -> (mult_commu OR r); Auto).
Save.
Hint mult_Og.

(**********)
Lemma mult_mult_r:(r,r1,r2:R)(r1==r2)->((multR r r1)==(multR r r2)).
(Intros; Apply (congr_eqT R R (multR r) r1 r2 H)).
Save.
Hint mult_mult_r.

(**********)
Lemma r_mult_mult:(r,r1,r2:R)((multR r r1)==(multR r r2))->
  (~(r==OR))->(r1==r2).
(Intros;
```

```
 Cut (multR (multR (inver r) r) r1)==(multR (multR (inver r) r) r2)).
(Rewrite -> inv1; Auto).
(Replace (multR IR r1) with r1; Replace (multR IR r2) with r2; Auto).
(Elim (mult_en r2); Auto).
(Elim (mult_en r1); Auto).
(Elim (mult_en r2); Auto).
(Rewrite -> mult_assoc; Rewrite -> mult_assoc).
(Rewrite <- H; Auto).
Save.
Hint mult_mult_r.

(**********)
Lemma sans_div_0d:(r1,r2:R)
  (((multR r1 r2)==0R)->((r1==0R)\/(r2==0R))).
(Intros; Cut r1==0R\/~r1==0R).
(Intro; Elim H0; Intro).
(Left; Auto).
(Right; Auto).
Replace r2 with (multR 0R (inver r1)).
Apply (mult_0g (inver r1)).
Cut (multR r1 (multR 0R (inver r1)))==(multR r1 r2).
(Intro; Apply (r_mult_mult r1 (multR 0R (inver r1)) r2 H2 H1)).
Rewrite -> mult_0g.
(Replace (multR r1 0R) with 0R; Auto).
Auto.
Save.

(**********)
Lemma sans_div_0i1:(r1,r2:R)
  ((r1==0R)->((multR r1 r2)==0R)).
(Intros; Rewrite -> H; Apply mult_0g).
Save.

(**********)
Lemma sans_div_0i2:(r1,r2:R)
  ((r2==0R)->((multR r1 r2)==0R)).
(Intros; Rewrite -> H; Apply mult_0d).
Save.

(**********)
Lemma sans_div_0i:(r1,r2:R)
  (((r1==0R)\/(r2==0R))->((multR r1 r2)==0R)).
(Intros; Elim H; Intro).
(Rewrite -> H0; Apply mult_0g).
(Rewrite -> H0; Apply mult_0d).
Save.
```

```
(***********)
Definition carre:R->R:=[r:R](multR r r).

(***********)
Lemma carre_O:((carre OR)==OR).
(Unfold carre; Auto).
Save.

(*******************************************************)
(*      Tout element est symetrisable                  *)
(*******************************************************)

(**********)
Lemma invers_mul:(r:R)(~(r==OR))->(ExT [r1:R] ((multR r r1)==IR)).
(Intros; Exists (inver r); Apply inv2; Auto).
Save.

(**********)
Lemma invers_add:(r:R)(ExT [r1:R] ((addR r r1)==OR)).
(Intros; Exists (unminus r); Auto).
Save.

(*******************************************************)
(*      Moins                                          *)
(*******************************************************)

(**********)
Definition minusR:R->R->R:=[r1,r2:R](addR r1 (unminus r2)).

(*******************************************************)
(*      Opposes                                        *)
(*******************************************************)

(**********)
Lemma neg_neg:(r:R)((unminus (unminus r))==r).
(Intro;
 Cut (addR (addR (unminus (unminus r)) (unminus r)) r)
      ==(unminus (unminus r))).
Replace (addR (unminus (unminus r)) (unminus r)) with OR.
(Replace (addR OR r) with r; Auto).
(Elim (add_en r); Auto).
(Apply sym_eqT; Apply (neg_pos (unminus r))).
(Rewrite -> add_assoc; Rewrite -> neg_pos;
 Elim (add_en (unminus (unminus r))); Auto).
Save.

(**********)
```

```
Lemma unminus_O:((unminus OR)==OR).
Cut (addR OR OR)==OR.
(Intro; Apply sym_eqT; Apply (oppose OR OR H)).
(Elim (add_en OR); Auto).
Save.
Hint unminus_O.

(**********)
Lemma unmin_distr1:(r1,r2:R)
 ((unminus (addR r1 r2))==(addR (unminus r1) (unminus r2))).
(Intros;
 Cut (addR (addR r1 r2) (unminus (addR r1 r2)))
      ==(addR (addR r1 r2) (addR (unminus r1) (unminus r2)))).
(Intro;
 Apply (r_add_add (addR r1 r2) (unminus (addR r1 r2))
         (addR (unminus r1) (unminus r2)) H)).
Rewrite -> pos_neg.
(Rewrite -> add_assoc;
 Replace (addR r1 (addR r2 (addR (unminus r1) (unminus r2))))
  with (addR r1 (addR (addR (unminus r1) (unminus r2)) r2))).
(Rewrite -> add_assoc; Rewrite -> neg_pos).
(Replace (addR (unminus r1) OR) with (unminus r1); Auto).
(Elim (add_en (unminus r1)); Auto).
(Replace (addR (addR (unminus r1) (unminus r2)) r2)
  with (addR r2 (addR (unminus r1) (unminus r2))); Auto).
Apply add_commu.
Save.

(**********)
Lemma unmin_distr2:(r1,r2:R)((unminus (minusR r1 r2))==(minusR r2 r1)).
(Intros; Unfold minusR; Rewrite -> add_commu).
(Rewrite -> unmin_distr1; Rewrite -> neg_neg; Auto).
Save.

(**********)
Lemma eq_minus:(r1,r2:R)(r1==r2)->((minusR r1 r2)==OR).
(Intros; Unfold minusR; Rewrite <- H; Auto).
Save.

(**********)
Lemma minus_eq:(r1,r2:R)((minusR r1 r2)==OR)->(r1==r2).
(Intros r1 r2; Unfold minusR; Rewrite -> add_commu; Intro).
(Rewrite <- (neg_neg r2); Apply (oppose (unminus r2) r1 H)).
Save.

(**********)
Lemma eq_unmin:(r1,r2:R)(r1==r2)->((unminus r1)==(unminus r2)).
```

```
(Intros; Rewrite <- H; Auto).
Save.

(**********)
Lemma eq_unmin0:(r:R)(r==0R)->((unminus r)==0R).
(Intros; Rewrite -> H; Auto).
Save.

(**********)
Lemma unmin_mul1:(r1,r2:R)
 ((multR (unminus r1) r2)==(unminus (multR r1 r2))).
(Intros; Cut (addR (multR (unminus r1) r2) (multR r1 r2))==0R).
Replace (addR (multR (unminus r1) r2) (multR r1 r2))
 with (minusR (multR (unminus r1) r2) (unminus (multR r1 r2))).
(Intro;
 Apply (minus_eq (multR (unminus r1) r2) (unminus (multR r1 r2)) H)).
(Unfold minusR; Rewrite -> neg_neg; Auto).
Replace (addR (multR (unminus r1) r2) (multR r1 r2))
 with (multR r2 (addR (unminus r1) r1)).
(Rewrite -> (neg_pos r1); Auto).
Replace (multR (unminus r1) r2) with (multR r2 (unminus r1)).
(Replace (multR r1 r2) with (multR r2 r1); [ Auto | Auto ]).
Auto.
Save.

(**********)
Lemma unmin_mul2:(r1,r2:R)
 ((multR (unminus r1) (unminus r2))==(multR r1 r2)).
(Intros; Rewrite -> unmin_mul1; Rewrite -> mult_commu;
 Rewrite -> unmin_mul1; Rewrite -> mult_commu; Apply neg_neg).
Save.

(*********************************************************)
(*       Axiomes d'ordre                                 *)
(*********************************************************)
(*********************************************************)
(*       Inferieur                                       *)
(*********************************************************)

(**********)
Axiom not_lt:(r1,r2:R)(ltR r1 r2)->~(ltR r2 r1).

(**********)
Lemma not_lt_r:(r:R)~(ltR r r).
(Red; Intros; Apply (not_lt r r H); Auto).
Save.
```

```
(**********)
Definition leR:R->R->Prop:=[r1,r2:R]((ltR r1 r2)\/(r1==r2)).

(**********)
Lemma le_le_eq:(r1,r2:R)(leR r1 r2)/\(leR r2 r1)<->(r1==r2).
(Intros; Split; Intro).
(Elim H; Unfold leR; Intros).
(Elim H0; Elim H1; Intros).
Absurd (ltR r2 r1).
(Red; Intro).
Apply (not_lt r1 r2 H3 H4).
Auto.
Auto.
Auto.
Auto.
(Rewrite <- H; Split; Unfold leR; Right; Auto).
Save.

(**********)
Lemma lt_le:(r1,r2:R)(ltR r1 r2)->(leR r1 r2).
(Intros; Unfold leR; Left; Auto).
Save.

(**********)
Axiom lt_transi:(r1,r2,r3:R)
  (ltR r1 r2)->(ltR r2 r3)->(ltR r1 r3).

(**********)
Lemma eq_le:(r1,r2:R)(r1==r2)->(leR r1 r2).
(Intros; Unfold leR; Right; Auto).
Save.

(**********)
Lemma le_transi:(r1,r2,r3:R)
  (leR r1 r2)->(leR r2 r3)->(leR r1 r3).
(Intros r1 r2 r3; Unfold leR; Intros).
(Elim H; Elim H0; Intros).
Cut (ltR r1 r3).
Intro.
(Apply (lt_le r1 r3); Auto).
(Apply (lt_transi r1 r2 r3); Auto).
(Rewrite <- H1; Auto).
(Rewrite -> H2; Auto).
(Rewrite -> H2; Auto).
Save.

(**********)
```

```
Lemma lte_transi1:(r1,r2,r3:R)
  (leR r1 r2)->(ltR r2 r3)->(ltR r1 r3).
(Intros; Unfold leR in H; Elim H; Intro).
Apply (lt_transi r1 r2 r3 H1 H0).
(Rewrite -> H1; Auto).
Save.

(**********)
Lemma lte_transi2:(r1,r2,r3:R)
  (ltR r1 r2)->(leR r2 r3)->(ltR r1 r3).

(Intros; Unfold leR in H0; Elim H0; Intro).
Apply (lt_transi r1 r2 r3 H H1).
(Rewrite <- H1; Auto).
Save.

(**********)
Axiom lt_add_add_r:(r,r1,r2:R)(ltR r1 r2)->
      (ltR (addR r r1) (addR r r2)).

(**********)
Lemma lt_r_add_add:(r,r1,r2:R)(ltR (addR r r1) (addR r r2))->
      (ltR r1 r2).
(Intros;
 Cut (ltR (addR (addR (unminus r) r) r1)
          (addR (addR (unminus r) r) r2))).
Rewrite -> neg_pos.
(Elim (add_en r1); Elim (add_en r2); Intros; Rewrite <- H3;
 Rewrite <- H1; Auto).
(Rewrite -> add_assoc; Rewrite -> add_assoc;
 Apply (lt_add_add_r (unminus r) (addR r r1) (addR r r2) H)).
Save.

(**********)
Lemma le_add_add_r:(r,r1,r2:R)(leR r1 r2)->
      (leR (addR r r1) (addR r r2)).
(Unfold leR; Intros; Elim H; Intro).
(Left; Apply (lt_add_add_r r r1 r2 H0)).
(Right; Rewrite <- H0; Auto).
Save.

Lemma le_r_add_add:(r,r1,r2:R)(leR (addR r r1) (addR r r2))->
      (leR r1 r2).
(Unfold leR; Intros; Elim H; Intro).
(Left; Apply (lt_r_add_add r r1 r2 H0)).
(Right; Apply (r_add_add r r1 r2 H0)).
Save.
```

```
(**********)
Axiom lt_monotony:(r,r1,r2:R)(ltR OR r)->(ltR r1 r2)->
      (ltR (multR r r1) (multR r r2)).

(**********)
Axiom lt_anti_monotony:(r,r1,r2:R)(ltR r OR)->(ltR r1 r2)->
      (gtR (multR r r1) (multR r r2)).

(**********)
Lemma lt_minus:(r1,r2:R)(ltR r1 r2)->(ltR (minusR r1 r2) OR).
(Intros; Cut (ltR (minusR r1 r2) (minusR r2 r2))).
(Rewrite -> (eq_minus r2); Auto).
(Unfold minusR; Rewrite -> (add_commu r1 (unminus r2));
 Rewrite -> (add_commu r2 (unminus r2));
 Apply (lt_add_add_r (unminus r2) r1 r2 H)).
Save.

(**********)
Lemma le_minus:(r1,r2:R)(leR r1 r2)->(leR (minusR r1 r2) OR).
(Unfold leR; Intros; Elim H; Intro).
(Left; Apply (lt_minus r1 r2 H0)).
(Right; Apply (eq_minus r1 r2 H0)).
Save.

(***********)
Lemma base_carre:(r:R)(leR OR (carre r)).
(Intro; Cut (ltR r OR)\/r==OR\/(gtR r OR)).
(Intro; Elim H; Intro; Unfold leR).
(Left; Replace OR with (multR r OR)).
(Unfold carre; Apply (lt_anti_monotony r r OR H0 H0)).
Auto.
(Elim H0; Intro).
(Right; Rewrite -> H1; Apply sym_eqT; Apply carre_0).
(Left; Unfold gtR in H1; Replace OR with (multR r OR)).
(Unfold carre; Apply (lt_monotony r OR r H1 H1)).
Auto.
Apply (base2 r OR).
Save.

(***********)
Lemma base_carre1:(r:R)(~(r==OR))->(ltR OR (carre r)).
(Intros; Cut (ltR r OR)\/(gtR r OR)).
(Intro; Elim H0; Intro).
(Replace OR with (multR r OR); Unfold carre).
Apply (lt_anti_monotony r r OR H1 H1).
Auto.
```

```
(Unfold gtR in H1; Replace OR with (multR r OR)).
(Unfold carre; Apply (lt_monotony r OR r H1 H1)).
Auto.
Apply (base3 r OR H).
Save.

(**********)
Lemma OR_IR:(ltR OR IR).
Cut ~IR==OR->(ltR OR (carre IR)).
(Intro; Replace IR with (carre IR); Cut ~IR==OR; Auto).
(Red; Intro; Apply (base0 H0)).
(Unfold carre; Elim (mult_en IR); Auto).
(Red; Intro; Apply (base0 H0)).
(Intro; Apply (base_carre1 IR H)).
Save.

(*******************************************************)
(*      Superieur                                      *)
(*******************************************************)

(**********)
Definition geR:R->R->Prop:=[r1,r2:R]((gtR r1 r2)\/(r1==r2)).

(**********)
Lemma gt_ge:(r1,r2:R)(gtR r1 r2)->(geR r1 r2).
(Intros; Unfold geR; Left; Auto).
Save.

(**********)
Lemma lt_commu:(r1,r2:R)
 (((ltR r1 r2)->(gtR r2 r1))/\((gtR r2 r1)->(ltR r1 r2))).
(Intros; Split; Intro; Unfold gtR; Auto).
Save.

(**********)
Lemma le_commu1:(r1,r2:R)(leR r1 r2)->(geR r2 r1).
(Intros r1 r2; Unfold geR; Unfold leR; Intro; Elim H; Intro).
(Left; Unfold gtR; Auto).
(Right; Auto).
Save.

(**********)
Lemma le_commu2:(r1,r2:R)(geR r2 r1)->(leR r1 r2).
(Intros r1 r2; Unfold geR; Unfold leR; Intro; Elim H; Intro).
(Left; Unfold gtR in H0; Auto).
(Right; Auto).
Save.
```

```
(**********)
Lemma le_commu:(r1,r2:R)
 (((leR r1 r2)->(geR r2 r1))/\((geR r2 r1)->(leR r1 r2))).
(Intros; Split; Intro).
(Apply le_commu1; Auto).
(Apply le_commu2; Auto).
Save.

(**********)
Lemma gte_transi1:(r1,r2,r3:R)
 (geR r1 r2)->(gtR r2 r3)->(gtR r1 r3).
(Intros r1 r2 r3; Unfold gtR; Unfold geR; Intros; Elim H; Intros).
(Unfold gtR in H1; Apply (lt_transi r3 r2 r1 H0 H1)).
(Rewrite -> H1; Auto).
Save.

(**********)
Lemma gte_transi2:(r1,r2,r3:R)
 (gtR r1 r2)->(geR r2 r3)->(gtR r1 r3).
(Intros r1 r2 r3; Unfold gtR; Unfold geR; Intros; Elim H0; Intros).
Apply (lt_transi r3 r2 r1 H1 H).
(Rewrite <- H1; Auto).
Save.

(**********)
Lemma gt_transi:(r1,r2,r3:R)
 (gtR r1 r2)->(gtR r2 r3)->(gtR r1 r3).
(Intros; Cut (geR r1 r2)).
(Intro; Apply (gte_transi1 r1 r2 r3 H1 H0)).
Apply (gt_ge r1 r2 H).
Save.

(**********)
Lemma ge_transi:(r1,r2,r3:R)
 (geR r1 r2)->(geR r2 r3)->(geR r1 r3).
(Intros; Cut (leR r2 r1)).
(Cut (leR r3 r2); Intros).
Cut (leR r3 r1).
(Intro; Apply (le_commu1 r3 r1 H3)).
Apply (le_transi r3 r2 r1 H1 H2).
Apply (le_commu2 r3 r2 H0).
Apply (le_commu2 r2 r1 H).
Save.

(**********)
Lemma eq_ge:(r1,r2:R)(r1==r2)->(geR r1 r2).
```

```
(Intros; Unfold geR; Right; Auto).
Save.

(**********)
Axiom gt_unmin:(r1,r2:R)
 (gtR r1 r2)->(ltR (unminus r1) (unminus r2)).

(**********)
Lemma lt_unmin:(r1,r2:R)
 (ltR r1 r2)->(gtR (unminus r1) (unminus r2)).
(Intros; Cut (gtR r2 r1)).
(Intro; Cut (ltR (unminus r2) (unminus r1)); Auto).
Apply (gt_unmin r2 r1 H0).
Auto.
Save.

(**********)
Lemma le_unmin:(r1,r2:R)
 (leR r1 r2)->(geR (unminus r1) (unminus r2)).
(Unfold leR; Intros; Unfold geR; Elim H; Intro).
(Left; Apply (lt_unmin r1 r2 H0)).
(Right; Apply (eq_unmin r1 r2 H0)).
Save.

(**********)
Lemma gt_unmin0:(r:R)(gtR r OR)->(ltR (unminus r) OR).
(Intros; Rewrite <- unminus_0; Apply (gt_unmin r OR); Auto).
Save.

(**********)
Lemma lt_unmin0:(r:R)(ltR r OR)->(gtR (unminus r) OR).
(Intros; Rewrite <- unminus_0; Apply (lt_unmin r OR); Auto).
Save.

(***********)
Lemma gt_add_add_r:(r,r1,r2:R)(gtR r1 r2)->
      (gtR (addR r r1) (addR r r2)).
(Unfold gtR; Intros; Apply (lt_add_add_r r r2 r1 H)).
Save.

(***********)
Lemma gt_r_add_add:(r,r1,r2:R)(gtR (addR r r1) (addR r r2))->
                              (gtR r1 r2).
(Unfold gtR; Intros; Apply (lt_r_add_add r r2 r1 H)).
Save.

(***********)
```

```
Lemma ge_add_add_r:(r,r1,r2:R)(geR r1 r2)->
      (geR (addR r r1) (addR r r2)).
(Unfold geR; Intros; Elim H; Intro).
(Left; Apply (gt_add_add_r r r1 r2 H0)).
(Right; Apply (add_add_r r r1 r2 H0)).
Save.

(***********)
Lemma ge_r_add_add:(r,r1,r2:R)(geR (addR r r1) (addR r r2))->
                              (geR r1 r2).
(Unfold geR; Intros; Elim H; Intro).
(Left; Apply (gt_r_add_add r r1 r2 H0)).
(Right; Apply (r_add_add r r1 r2 H0)).
Save.

(***********)
Lemma gt_minus:(r1,r2:R)(gtR r1 r2)->(gtR (minusR r1 r2) OR).
(Intros; Cut (gtR (minusR r1 r2) (minusR r2 r2))).
(Rewrite -> (eq_minus r2 r2); Auto).
(Unfold minusR; Rewrite -> (add_commu r1 (unminus r2));
 Rewrite -> (add_commu r2 (unminus r2));
 Apply (gt_add_add_r (unminus r2) r1 r2 H)).
Save.

(**********)
Lemma ge_minus:(r1,r2:R)(geR r1 r2)->(geR (minusR r1 r2) OR).
(Unfold geR; Intros; Elim H; Intro).
(Left; Apply (gt_minus r1 r2 H0)).
(Right; Apply (eq_minus r1 r2 H0)).
Save.

(*******************************************************)
(*      Injection de N dans R                          *)
(*******************************************************)

(**********)
Fixpoint INR [n:nat]:R:=(Cases n of
                          O      => OR
                         |(S O)  => IR
                         |(S n)  => (addR (INR n) IR)
                        end).

(**********)
Lemma S_INR:(n:nat)(INR (S n))==(addR (INR n) IR).
(Intro; Simpl; Case n).
(Simpl; Elim (add_en IR); Auto).
Auto.
```

```
Save.

(**********)
Lemma S_O_add_INR:(n:nat)
    (INR (plus (S O) n))==(addR (INR (S O)) (INR n)).
(Intro; Replace (plus (S O) n) with (S (plus O n))).
Replace (INR (S (plus O n))) with (addR (INR (plus O n)) IR).
Replace (INR (S O)) with (addR (INR O) IR).
(Replace (INR (S O)) with (addR (INR O) IR); Simpl; Elim (add_en IR);
 Intros; Rewrite -> H0; Auto; Rewrite -> add_commu; Auto).
Simpl.
(Elim (add_en IR); Auto).
(Apply sym_eqT; Apply (S_INR (plus O n))).
Auto.
Save.

(**********)
Lemma add_INR:(n,m:nat)(INR (plus n m))==(addR (INR n) (INR m)).
Induction n.
(Simpl; Intro; Elim (add_en (INR m)); Auto).
(Intros; Rewrite -> plus_Snm_nSm;
 Replace (INR (S n0)) with (addR (INR n0) IR)).
(Rewrite -> add_assoc; Rewrite -> (add_commu IR (INR m));
 Replace (addR (INR m) IR) with (INR (S m))).
Apply (H (S m)).
Apply (S_INR m).
(Apply sym_eqT; Apply (S_INR n0)).
Save.

(**********)
Lemma minus_INR:(n,m:nat)(le m n)->
   (INR (minus n m))==(minusR (INR n) (INR m)).
(Intros;
 Cut (addR (INR m) (INR (minus n m)))
     ==(addR (INR m) (minusR (INR n) (INR m)))).
(Intro;
 Apply (r_add_add (INR m) (INR (minus n m))
                  (minusR (INR n) (INR m)) H0)).
(Rewrite <- add_INR; Rewrite -> le_plus_minus_r).
(Unfold minusR; Rewrite -> add_commu; Rewrite -> add_assoc;
 Rewrite -> neg_pos; Elim (add_en (INR n)); Auto).
Auto.
Save.


(**********************************************************)
(*       Injection de Z dans R                            *)
```

```
(******************************************************)

(**********)
Definition IZR:Z->R:=[z:Z](Cases z of
                         OZ      => OR
                        |(pos n) => (addR (INR n) IR)
                        |(neg n) => (addR (unminus (INR n))
                                          (unminus IR))
                           end).

(**********)
Lemma pos_IZR:(n:nat)(IZR (pos n))==(addR (INR n) IR).
Auto.
Save.

(**********)
Lemma neg_IZR:(n:nat)(IZR (neg n))==(addR (unminus (INR n))
                                          (unminus IR)).
Auto.
Save.

(**********)
Lemma add_IZR:(z,t:Z)(IZR (addZ z t))==(addR (IZR z) (IZR t)).
Induction z.
     (*OZ*)
(Simpl; Intro; Elim (add_en (IZR t)); Auto).
     (*pos*)
(Intro; Induction t).
(Rewrite -> add_OZ; Simpl; Elim (add_en (IZR (pos n))); Auto).
(Intro; Rewrite -> tech_add_pos_posZ;
 Replace (IZR (pos (S (plus n n0))))
  with (addR (INR (S (plus n n0))) IR)).
Replace (IZR (pos n)) with (addR (INR n) IR).
Replace (IZR (pos n0)) with (addR (INR n0) IR).
Replace (INR (S (plus n n0))) with (addR (INR (plus n n0)) IR).
(Rewrite -> add_INR; Rewrite -> add_commu;
 Rewrite -> (add_assoc (INR n) (INR n0) IR);
 Rewrite <- (add_assoc IR (INR n) (addR (INR n0) IR));
 Rewrite -> (add_commu (INR n) IR); Auto).
(Apply sym_eqT; Apply (S_INR (plus n n0))).
Auto.
Auto.
Auto.
(Intro; Cut ({(lt n n0)}+{n=n0})+{(lt n0 n)}).
(Intro; Elim H; Intro).
(Elim y; Intro).
(Rewrite -> tech_add_pos_neg_negZ; Simpl).
```

```
Rewrite -> minus_INR.
(Rewrite -> unmin_distr2; Unfold minusR; Rewrite -> S_INR;
 Apply (add_assoc (addR (INR n) IR) (unminus (INR n0)) (unminus IR))).
Auto.
Auto.
(Rewrite -> y0; Rewrite -> tech_add_pos_negZ; Simpl;
 Rewrite <- unmin_distr1;
 Cut (minusR (addR (INR n0) IR) (addR (INR n0) IR))==OR).
(Unfold minusR; Auto).
(Apply (eq_minus (addR (INR n0) IR) (addR (INR n0) IR)); Auto).
(Rewrite -> tech_add_pos_neg_posZ; Simpl).
Rewrite -> minus_INR.
(Unfold minusR; Rewrite -> S_INR; Rewrite -> unmin_distr1;
 Rewrite -> add_commu;
 Rewrite <- (add_assoc IR (INR n)
             (addR (unminus (INR n0)) (unminus IR)));
 Rewrite -> (add_commu IR (INR n)); Auto).
Auto.
Auto.
Apply lt_eq_lt_dec.
     (*neg*)
(Intro; Induction t).
(Rewrite -> add_OZ; Simpl; Elim (add_en (IZR (neg n))); Auto).
(Intro; Rewrite -> add_commu;
 Replace (addZ (neg n) (pos n0)) with (addZ (pos n0) (neg n))).
Cut ({(lt n0 n)}+{n0=n})+{(lt n n0)}.
(Intro; Elim H; Intro).
(Elim y; Intro).
(Rewrite -> tech_add_pos_neg_negZ; Simpl).
Rewrite -> minus_INR.
(Rewrite -> unmin_distr2; Unfold minusR; Rewrite -> S_INR;
 Apply (add_assoc (addR (INR n0) IR) (unminus (INR n)) (unminus IR))).
Auto.
Auto.
(Rewrite -> y0; Rewrite -> tech_add_pos_negZ; Simpl;
 Rewrite <- unmin_distr1;
 Cut (minusR (addR (INR n) IR) (addR (INR n) IR))==OR).
(Unfold minusR; Auto).
(Apply (eq_minus (addR (INR n) IR) (addR (INR n) IR)); Auto).
(Rewrite -> tech_add_pos_neg_posZ; Simpl).
Rewrite -> minus_INR.
(Unfold minusR; Rewrite -> S_INR; Rewrite -> unmin_distr1;
 Rewrite -> add_commu;
 Rewrite <- (add_assoc IR (INR n0)
             (addR (unminus (INR n)) (unminus IR)));
 Rewrite -> (add_commu IR (INR n0)); Auto).
Auto.
```

```
Auto.
Apply lt_eq_lt_dec.
Apply addZ_commutativity.
(Intro; Rewrite -> tech_add_neg_negZ;
 Replace (IZR (neg (S (plus n n0))))
  with (addR (unminus (INR (S (plus n n0)))) (unminus IR))).
Replace (IZR (neg n)) with (addR (unminus (INR n)) (unminus IR)).
Replace (IZR (neg n0)) with (addR (unminus (INR n0)) (unminus IR)).
(Rewrite -> S_INR; Rewrite -> add_INR; Rewrite -> add_commu;
 Replace (addR (unminus IR) (unminus (addR (addR (INR n) (INR n0)) IR)))
  with (addR (addR (unminus IR) (unminus (INR n)))
         (addR (unminus (INR n0)) (unminus IR)))).
(Rewrite <- (add_commu (unminus IR) (unminus (INR n))); Auto).
(Rewrite -> unmin_distr1; Rewrite -> unmin_distr1;
 Rewrite -> add_assoc; Rewrite -> add_assoc; Auto).
Auto.
Auto.
Auto.
Save.

(**********)
Lemma unmin_opp_IZR:(z:Z)(IZR (oppZ z))==(unminus (IZR z)).
(Induction z; Simpl).
(Apply sym_eqT; Apply unminus_O).
(Intro; Apply sym_eqT; Apply (unmin_distr1 (INR n) IR)).
(Intro; Cut (unminus (unminus IR))==IR).
(Intro; Cut (unminus (unminus (INR n)))==(INR n)).
(Intro; Pattern 1 (INR n) ; Rewrite <- H0; Pattern 1 IR ; Rewrite <- H;
 Apply sym_eqT; Apply (unmin_distr1 (unminus (INR n)) (unminus IR))).
Apply (neg_neg (INR n)).
Apply (neg_neg IR).
Save.

(**********)
Lemma Z_R_minus:(z1,z2:Z)
  (minusR (IZR z1) (IZR z2))==(IZR (minusZ z1 z2)).
(Unfold minusR; Unfold minusZ; Intros; Apply sym_eqT;
 Replace (unminus (IZR z2)) with (IZR (oppZ z2));
 [ Apply (add_IZR z1 (oppZ z2)) | Apply (unmin_opp_IZR z2) ]).
Save.

(*******************************************************)
(*       R Archimedien                                 *)
(*******************************************************)

(**********)
Axiom archimed:(r:R)(geR r OR)->(gtR (INR (suparch r)) r).
```

```
(*******************************************************)
(*      R Complet                                      *)
(*******************************************************)

(**********)
Definition majorant:=[E:R->Prop][m:R]
  (x:R)((E x)->(ExT [m:R](leR x m))).

(**********)
Definition borne:=[E:R->Prop](ExT [m:R](majorant E m)).

(**********)
Definition borne_sup:=[E:R->Prop][m:R]
  ((ExT [m:R](majorant E m))/\
  (~(ExT [b:R]((majorant E b)/\(ltR b m))))).

(**********)
Axiom complet:(E:R->Prop)((borne E)->(ExT [m:R](borne_sup E m))).
```

Annexe C

Compléments pour les Naturels

```
(*****************************************************)
(*            Complements pour les Naturels          *)
(*                                                   *)
(*****************************************************)

Require Export Arith.
Require Export Compare_dec.
(*****************************************************)

Definition decidable:=[P:Prop]P\/~P.
(******************************************************)
(*      Lemmes techniques                             *)
(******************************************************)

(**********)
Lemma princip_min:(P:nat->Prop)((n:nat)(decidable (P n)))->
  (n:nat)(P n)->(Ex [m:nat] ((P m)/\(i:nat)(P i)->(ge i m))).
(Intros;
 Cut (Ex [m:nat](P m)/\((i:nat)(P i)->(ge i m)))
     \/((p:nat)(ge n p)->~(P p))).
(Intro; Elim H1; Auto).
(Intro; Generalize (H2 n (le_n n)); Intros; Absurd (P n); Auto).
Elim n.
Elim (H O);Intros.
(Left; Exists O; Auto).
(Right; Unfold ge; Intros; Cut p=O).
(Intro;Rewrite -> H3; Auto).
Auto.
(Intros; Elim H1; Auto).
(Intro; Elim (H (S n0)); Intro).
(Left; Exists (S n0); Split; Auto).
```

```
Cut (p:nat)(le (S n0) p)\/(lt p (S n0)).
(Intros; Elim (H4 i); Auto).
(Unfold lt; Intro; Generalize (le_S_n i n0 H6); Intro;
 Unfold ge in H2; Elim (H2 i H7); Auto).
(Intro; Apply le_or_lt).
(Right; Intros; Cut (lt p (S n0))\/p=(S n0)).
(Intro; Elim H5; Intro).
Generalize (lt_n_Sm_le p n0 H6).
(Intro; Unfold ge in H2; Generalize (H2 p H7); Auto).
(Rewrite -> H6; Auto).
Apply (le_lt_or_eq p (S n0) H4).
Save.

(**********)
Lemma inser_trans:(n,m,p,q:nat)(le n m)/\(lt m p)->
      {(le n m)/\(lt m q)}+{(le q m)/\(lt m p)}.
(Intros; Cut {(le q m)}+{(lt m q)}).
(Intro; Elim H; Intros).
(Elim H0; Intro).
(Right; Auto).
(Left; Auto).
Apply (le_lt_dec q m).
Save.

(**********)
Lemma inser2_trans:(n,m,p,q,t:nat)(le n m)/\(lt m t)->
 {(le n m)/\(lt m q)}+{(le q m)/\(lt m p)}+{(le p m)/\(lt m t)}.
(Intros; Cut {((le n m)/\(lt m p))}+{((le p m)/\(lt m t))}).
(Intro; Elim H0; Intro).
(Left; Apply (inser_trans n m p q y)).
(Right; Auto).
Apply (inser_trans n m t p H).
Save.

(*******************************************************)
(*       Operateur supplementaire                      *)
(*******************************************************)

(**********)
Fixpoint max [n:nat]:nat->nat:=
  [m:nat]Cases n m of
          O b => b
         |(S a) O => (S a)
         |(S a) (S b) => (S (max a b))
        end.
Hint max.
```

```
(*******************************************************)
(*       Proprietes du max                             *)
(*******************************************************)

(**********)
Lemma max_or:(n,m:nat)(max n m)=n\/(max n m)=m.
Induction n.
Intro.
Auto.
(Induction m; Intros).
Auto.
(Elim (H n1); Intros).
Left.
(Simpl; Auto).
Right.
(Simpl; Auto).
Save.


(**********)
Lemma lt_lt_max:(n,m,N:nat)(lt n N)/\(lt m N)->(lt (max n m) N).
Induction n.
Intros.
(Elim H; Intros).
Auto.
(Induction m; Intros).
Simpl.
(Elim H0; Intros; Auto).
(Cut (max (S n0) (S n1))=(S n0)\/(max (S n0) (S n1))=(S n1); Intros).
(Elim H2; Intros; Rewrite -> H3; Auto; Elim H1; Auto).
Apply max_or.
Save.
```

Annexe D

Outils pour le théorème

```
(*******************************************************)
(*           Outils pour The three gap theorem         *)
(*                                                     *)
(*******************************************************)

Require Export Rbase.
Require Export Nat_compl.

Variable ppNsup:R->Z.
Variable alpha:R.
(*******************************************************)

Definition prop_alpha:=((leR OR alpha)/\(ltR alpha IR)).

(*******************************************************)
(*      Partie fractionnaire                           *)
(*******************************************************)

(**********)
Lemma ppNsup_A:(r:R)(gtR (IZR (ppNsup r)) r)
                 /\(leR (minusR (IZR (ppNsup r)) r) IR)).

(**********)
Definition pEr:R->Z:=[r:R](minusZ (ppNsup r) IZ).

(**********)
Definition part_frac:R->R:=[r:R](minusR r (IZR (pEr r))).

(**********)
Lemma for_base_pf:(r:R)(gtR (minusR (IZR (ppNsup r)) r) OR)/\
                       (leR (minusR (IZR (ppNsup r)) r) IR).
(Intro; Split;
 Cut (gtR (IZR (ppNsup r)) r)/\
```

```
     (leR (minusR (IZR (ppNsup r)) r) IR)).
(Intro; Elim H; Intros).
Apply (gt_minus (IZR (ppNsup r)) r H0).
Apply ppNsup_A.
(Intro; Elim H; Intros).
Exact H1.
Apply ppNsup_A.
Save.

(**********)
Lemma base_pf:(r:R)(geR (part_frac r) OR)
                 /\(ltR (part_frac r) IR).
(Intro; Unfold part_frac; Unfold pEr; Split).
     (*sup a 0*)
Cut (geR (minusR r (IZR (ppNsup r))) (unminus IR)).
(Rewrite <- Z_R_minus; Simpl; Elim (add_en IR); Intros;
 Rewrite -> H0;
 Unfold minusR; Rewrite -> (unmin_distr1 (IZR (ppNsup r))
 (unminus IR));
 Rewrite -> neg_neg; Rewrite <- add_assoc; Rewrite -> add_commu;
 Replace OR with (addR IR (unminus IR))).
Apply (ge_add_add_r IR (addR r (unminus (IZR (ppNsup r))))
                               (unminus IR) H1).
Apply (pos_neg IR).
Cut (leR (minusR (IZR (ppNsup r)) r) IR).
(Intro; Rewrite <- unmin_distr2;
 Apply (le_unmin (minusR (IZR (ppNsup r)) r) IR H)).
Cut (gtR (minusR (IZR (ppNsup r)) r) OR)
    /\(leR (minusR (IZR (ppNsup r)) r) IR).
(Intro; Elim H; Intros).
Exact H1.
Apply for_base_pf.
     (*inf a 1*)
Cut (ltR (minusR r (IZR (ppNsup r))) OR).
(Rewrite <- Z_R_minus; Simpl; Elim (add_en IR); Intros;
 Rewrite -> H0;
 Unfold minusR; Rewrite -> (unmin_distr1 (IZR (ppNsup r))
                                          (unminus IR));
 Rewrite -> neg_neg; Rewrite <- add_assoc; Rewrite -> add_commu;
 Cut (addR IR OR)==IR).
(Intro; Pattern 2 IR ; Rewrite <- H;
 Apply (lt_add_add_r IR (addR r (unminus (IZR (ppNsup r)))) OR
        H1)).
(Elim (add_en IR); Auto).
Cut (gtR (minusR (IZR (ppNsup r)) r) OR).
(Intro; Rewrite <- unmin_distr2; Rewrite <- unminus_O;
 Apply (gt_unmin (minusR (IZR (ppNsup r)) r) OR H)).
```

```
Cut (gtR (minusR (IZR (ppNsup r)) r) OR)
    /\(leR (minusR (IZR (ppNsup r)) r) IR).
(Intro; Elim H; Intros).
Exact H0.
Apply for_base_pf.
Save.

(**********)
Lemma pEr_0:(x:R)(leR OR x)->(ltR x IR)->(pEr x)=OZ.

(**********)
Lemma pf_0:(x:R)(leR OR x)->(ltR x IR)->x==(part_frac x).
(Intros; Unfold part_frac; Rewrite -> pEr_0).
(Simpl; Unfold minusR; Rewrite -> unminus_0; Elim (add_en x);
 Auto).
Auto.
Auto.
Save.

(**********)
Lemma pEr_n:(n:Z)(pEr (IZR n))=n.

(**********)
Lemma pEr_minus:(x:R)(n:Z)
  (pEr (minusR x (IZR n)))=(minusZ (pEr x) n).

(**********)
Lemma pf_minus_R_Z:(x:R)(n:Z)
  (part_frac (minusR x (IZR n)))==(part_frac x).
(Intros; Unfold part_frac; Rewrite -> pEr_minus;
 Rewrite <- (Z_R_minus (pEr x) n); Unfold minusR;
 Rewrite -> unmin_distr1; Rewrite -> neg_neg;
 Rewrite -> (add_commu (unminus (IZR (pEr x))) (IZR n));
 Rewrite -> add_assoc;
 Rewrite <- (add_assoc (unminus (IZR n)) (IZR n)
              (unminus (IZR (pEr x)))); Rewrite -> neg_pos;
 Elim (add_en (unminus (IZR (pEr x)))); Auto).
Save.

(**********)
Lemma minus_R_Z_pf:(x:R)(n:Z)(leR OR (minusR x (IZR n)))->
                             (ltR (minusR x (IZR n)) IR)->
                   (minusR x (IZR n))==(part_frac x).
(Intros; Cut (minusR x (IZR n))==(part_frac (minusR x (IZR n)))).
(Intro; Rewrite -> H1; Apply pf_minus_R_Z).
Apply (pf_0 (minusR x (IZR n)) H H0).
Save.
```

```
(**********)
Lemma minus_R_Z_pEr:(x:R)(n:Z)(leR OR (minusR x (IZR n)))->
                              (ltR (minusR x (IZR n)) IR)->
                    (minusR x (IZR n))==(minusR x (IZR (pEr x))).
(Intros; Cut (minusR x (IZR n))==(part_frac x)).
(Intro; Rewrite -> H1; Unfold part_frac; Auto).
Apply (minus_R_Z_pf x n H H0).
Save.

(**********)
Lemma rel_Z_pf:(x,y:R)(n:Z)(leR OR x)->(ltR x IR)->
   (IZR n)==(minusR y x)->x==(part_frac y).

(**********)
Lemma add_part_frac1:(r1,r2:R)
  (geR (addR (part_frac r1) (part_frac r2)) IR)->
       (part_frac (addR r1 r2))=
       (minusR (addR (part_frac r1) (part_frac r2))IR).

(**********)
Lemma add_part_frac2:(r1,r2:R)
  (ltR (addR (part_frac r1) (part_frac r2)) IR)->
  (part_frac (addR r1 r2))=(addR (part_frac r1) (part_frac r2)).

(*********************************************************)
(*          U2 et UN                                     *)
(*********************************************************)

(**********)
Definition part_frac_n_alpha:nat->R:=[n:nat]
  (part_frac (multR (INR n) alpha)).

(**********)
Definition ordre_total:=[n,m:nat]prop_alpha->
  (leR (part_frac_n_alpha n) (part_frac_n_alpha m)).

(**********)
Lemma N_classic:(N:nat){N=O}+{N=(S O)}+{(ge N (S (S O)))}.
(Induction N; Auto).
(Intros; Elim H; Intro).
(Elim y; Intro).
(Left; Right; Auto).
(Right; Rewrite <- y0; Auto).
(Right; Auto).
Save.
```

```
(**********)
Lemma tech_ord_tot:(n:nat)(ordre_total n n).
(Intro; Unfold ordre_total; Intro; Unfold leR; Right; Auto).
Save.

(**********)
Lemma tech_part_frac:(part_frac 0R)==0R.

(**********)
Lemma princip_min_ord_tot:(P:nat->Prop)
 ((n:nat)(decidable (P n)))->
 (n:nat)(P n)->(Ex [m:nat] ((P m)/\(i:nat)(P i)->(ordre_total m i))).

(**********)
(*
Lemma dec_ordtot:(n,m:nat)(ordre_total n m)\/~(ordre_total n m).
*)

(**********)
(*
Lemma exist_U2:(N:nat)(ge N (S (S O)))->
      (m:nat)(principe_min ([n:nat](ordre_total n m))
                          ([n:nat](dec_ordtot n m))).*)

(**********)
Lemma exist_U2:(N:nat)(ge N (S (S O)))->
                     {n:nat|(m:nat)(lt O m)/\(le m N)
                            /\(ordre_total n m)
                            /\(lt O n)/\(lt n N)}.

(**********)
Lemma exist_UN:(N:nat)(ge N (S (S O)))->
                     {n:nat|(m:nat)(lt O m)/\(le m N)
                           /\(ordre_total m n)
                           /\(lt O n)/\(lt n N)}.

(**********)
Definition U2:=[N:nat](Cases (N_classic N) of
  (inright p)=>(<nat>Cases (exist_U2 N p) of (exist a b)=>a end)
 |     _     => O
                      end).

(**********)
Definition UN:=[N:nat](Cases (N_classic N) of
  (inright p)=>(<nat>Cases (exist_UN N p) of (exist a b)=>a end)
 |   _       => O
                      end).
```

Annexe E

Preuve intermédiaire 1

```
(********************************************************)
(*         Preuve intermediaire 1                       *)
(*                                                      *)
(********************************************************)

Require Export tools.
(********************************************************)

(**********)
Lemma U2_N:(N:nat)(ge N (S(S O)))->(lt (U2 N) N).
(Intros; Unfold U2; Case (N_classic N); Intro).
Absurd (ge N (S (S O))).
(Elim s; Intro; Rewrite -> y; Unfold ge; Auto).
Exact H.
(Case (exist_U2 N g); Intros; Elim (a x); Intros; Elim H1; Intros;
 Elim H3; Intros; Elim H5; Intros; Exact H7).
Save.

(**********)
Lemma UN_N:(N:nat)(ge N (S(S O)))->(lt (UN N) N).
(Intros; Unfold UN; Case (N_classic N); Intro).
Absurd (ge N (S (S O))).
(Elim s; Intro; Rewrite -> y; Unfold ge; Auto).
Exact H.
(Case (exist_UN N g); Intros; Elim (a x); Intros; Elim H1; Intros;
 Elim H3; Intros; Elim H5; Intros; Exact H7).
Save.

(**********)
Lemma inter1:(N:nat)(ge N (S(S O)))->(lt (max (U2 N) (UN N)) N).
(Intros; Cut (lt (U2 N) N)/\(lt (UN N) N)).
(Intro; Apply (lt_lt_max (U2 N) (UN N)); Auto).
(Split; [ Apply (U2_N N H) | Apply (UN_N N H) ]).
```

```
Save.
```

Annexe F

Preuve intermédiaire 2

```
(*******************************************************)
(*             Preuve intermediaire 2                  *)
(*                                                     *)
(*******************************************************)

Require Export tools.
(*******************************************************)

(**********)
Lemma inter2:(N:nat)(le N (plus U2 UN)).
```

Annexe G

Preuve intermédiaire 3

```
(******************************************************)
(*          Preuve intermediaire 3                    *)
(*                                                    *)
(******************************************************)

Require Export tools.

Variable after:nat->nat.
(******************************************************)

(******************************************************)
(*      Definition du prochain point                  *)
(******************************************************)

(**********)
Lemma after_A:(n:nat)(N:nat)((le O n)/\(lt n N))->
                  (ordre_total n (after n)).

(**********)
Lemma after_U2:(N:nat)((after O)=(U2 N)).

(**********)
Lemma after_UN:(N:nat)((after (UN N))=O).

(******************************************************)
(*      Lemme                                         *)
(******************************************************)

(**********)
Lemma inter31a:(N:nat)(n:nat)((le O n)/\(lt n (UN N)))->
                   (minusZ (INZ (after n)) (INZ n))=(INZ (U2 N)).

(**********)
```

```
Lemma inter31b:(N:nat)(n:nat)((le (UN N) n)/\(lt n N))->
     (minusZ (INZ (after n)) (INZ n))=(oppZ (INZ (UN N))).

(**********)
Lemma inter32a:(N:nat)(n:nat)((le O n)/\(lt n (minus N (U2 N))))->
                    (minusZ (INZ (after n)) (INZ n))=(INZ (U2 N)).

(**********)
Lemma inter32b:(N:nat)(n:nat)((le (minus N (U2 N)) n)/\(lt n N))->
     (minusZ (INZ (after n)) (INZ n))=(minusZ (INZ N) (INZ (U2 N))).
```

Annexe H

Preuve intermédiaire 4

```
(*******************************************************)
(*                Preuve intermediaire 4               *)
(*                                                     *)
(*******************************************************)

Require Export preuve1.
Require Export preuve2.
Require Export preuve3.
(*******************************************************)

(**********)
Lemma three_gap1:(N:nat)(n:nat)((le O n)/\(lt n (minus N (U2 N))))->
                   (minusZ (INZ (after n)) (INZ n))=(INZ (U2 N)).

(**********)
Lemma three_gap2:(N:nat)(n:nat)
                 ((le (minus N (U2 N)) n)/\(lt n (UN N)))->
  (minusZ (INZ (after n)) (INZ n))=(minusZ (INZ (U2 N)) (INZ (UN N))).

(**********)
Lemma three_gap3:(N:nat)(n:nat)((le (UN N) n)/\(lt n N))->
     (minusZ (INZ (after n)) (INZ n))=(oppZ (INZ (UN N))).
```

Annexe I

The three gap theorem

```
(*******************************************************)
(*                 THE THREE GAP THEOREM               *)
(*                                                     *)
(*******************************************************)

Require Export preuve4.
(*******************************************************)

(**********)
Lemma three_gap:(N:nat)(n:nat)((le O n)/\(lt n N))->
      {(minusZ (INZ (after n)) (INZ n))=(INZ (U2 N))}+
 {(minusZ (INZ (after n)) (INZ n))=(minusZ (INZ (U2 N)) (INZ (UN N)))}+
      {(minusZ (INZ (after n)) (INZ n))=(oppZ (INZ (UN N)))}.

(Intros;
 Cut ({((le O n)/\(lt n (minus N (U2 N))))}
       +{((le (minus N (U2 N)) n)/\(lt n (UN N)))})
      +{((le (UN N) n)/\(lt n N))}).
(Intro; Elim H0; Intros).
(Elim y; Intros).
(Left; Left; Apply (three_gap1 N n y0)).
(Left; Right; Apply (three_gap2 N n y0)).
(Right; Apply (three_gap3 N n y)).
Apply (inser2_trans O n (UN N) (minus N (U2 N)) N H).
Save.
```

Bibliographie

[AE+97] Abbas Edalat, Peter John Pott, *Exact Real Computer Arithmetic.* Department of Computing Imperial College. March 1997.

[AT69] Alfred Tarski, *What is elementary geometry?* Oxford Readings in Philosophy. Oxford University Press. 1969.

[BB+97] Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Jean-Christophe Filliâtre, Eduardo Giménez, Hugo Herbelin, Gérard Huet, César Muñoz, Chetan Murthy, Catherine Parent, Christine Paulin-Mohring, Amokrane Saïbi, Benjamin Werner, *The Coq Proof Assistant Reference Manual Version 6.1.* Technical Report 0203, INRIA, May 1997.

[BW94] Benjamin Werner, *Une théorie des constructions inductives.*Thèse. Mai 1994.

[CJ93] Claire Jones, *Completing the Rationals ans Metric Spaces in LEGO.* Cambridge.1993.

[EL51] Edmund Landau, *Foundations of analysis.The arithmetic of whole, Rational, irrational and complex numbers.* Chelsea Publishing Company. 1951.

[JD68] Jean Dieudonné, *Eléments d'analyse.Vol. 1. Fondements de l'analyse moderne.* Gauthier-Villars Paris. 1968.

[NB53] Nicolas Bourbaki, *Eléments de mathématique. Topologie générale.* Chapitres 1-10. Masson. 1953.

[SB97] Samuel Boutin, *Réflexions sur les quotients.* Thèse. Avril 1997.

[SL71] Serge Lang, *Algebra.* Addison-Wesley Publishing Company. 1971.

[TR86] Tony Van Ravenstein, *The three gap theorem (Steinhaus conjecture),* J.Austral.Math.Soc.(Series A) 45 (1988),360-370.

[VM94] Valérie Ménissier-Morain, *Arithmétique exacte. Conception, algorithme et performances d'une implémentation informatique en précision arbitraire.* Thèse. Décembre 1994.

[YG72] Jean Yves Girard, *Interprétation Fonctionnelle et Elimination des Coupures dans l'Arithmétique d'Ordre Supérieur.* Thèse. 1972.