Mémoire de stage de DEA

—

Certification des preuves de terminaison en Coq

—

Thierry Hubert

—

Stage effectué du 5 avril 2004 au 30 septembre 2004
Équipe DEMONS
Laboratoire de Recherche en Informatique, UMR 8623 du CNRS

Table des matières

1 Introduction

Le sujet de ce stage est la certification des preuves de terminaison des systèmes de réécriture.

La réécriture est un formalisme de calcul basé sur la donnée de règles de remplacement. A titre d' exemple nous allons considérer un système de règles de réécriture pour le calcul de l'addition de nombres codés en binaire. On note les nombres sous la forme $\#b_1 \cdots b_n$ où les b_i sont les chiffres binaires par exemple : 23 s'écrit #10111. Les règles de calcul sont :

$$
\begin{aligned}
\#0 &\rightarrow \# && (1)\\
\# + x &\rightarrow x && (2)\\
x + \# &\rightarrow x && (3)\\
x0 + y0 &\rightarrow (x+y)0 && (4)\\
x0 + y1 &\rightarrow (x+y)1 && (5)\\
x1 + y0 &\rightarrow (x+y)1 && (6)\\
x1 + y1 &\rightarrow (x+y+\#1)0 && (7)
\end{aligned}
$$

où la première règle se charge d'éliminer les zéros inutiles, les règles 2 et 3 définissent l'addition avec zéro, et les 4 dernières règles correspondent à l'algorithme d'addition habituel en procédant de droite à gauche, avec retenue pour la dernière règle. Par exemple 3+2 se calcule ainsi :

$$\#11 + \#10 \rightarrow_6 (\#1 + \#1)1 \rightarrow_7 (\# + \# + \#1)01 \rightarrow_2 (\# + \#1)01 \rightarrow_2 \#101$$

c'est à dire 5.

On dit alors qu'un système de réécriture termine si aucun terme ne peut se réduire indéfiniment, ce qui est le cas pour le système ci dessus. Il est facile de montrer qu'on ne peut pas décider la terminaison des systèmes de réécriture : en effet on peut coder n'importe quelle machine de Turing par un tel système [8].

Un article pionnier sur la terminaison des systèmes de réécriture est celui de Manna et Ness [23] en 1970, dans lequel ils proposent le critère suivant : s'il existe une relation d'ordre sur les termes qui soit monotone, bien fondée et stable par instanciation, tel que $l > r$ pour toute règle $l \rightarrow r$ du système, alors le système termine. L'utilisation de ce critère a prévalu durant de nombreuses années, les travaux de recherche consistant à trouver des définitions de tels ordres : l'ordre de Knuth-Bendix [20], les ordres par interprétation polynomiale [21, 4], les ordres récursifs sur les chemins [13, 27, 10, 19], etc. on pourra se référer à [11, 12, 3, 29] pour plus de détails.

En 1997, un nouvelle étape dans les techniques de preuves de terminaison a été franchie avec le critère des paires de dépendance proposé par Arts et Giesl [1, 2]. Ce critère est particulièrement adapté à l'implantation d'outils de recherche automatique de preuve de terminaison, et en effet plusieurs outils de ce type ont été développés ces dernières années dont l'outil CiME [24, 7] développé dans l'équipe DEMONS, mais aussi AProVe [15], TTT [18], Termptation [6], Cariboo [14], Mu-Term [22], etc.

Les techniques de preuve de terminaison implantées dans ces outils sont devenues très complexes : algorithme sur les graphes, résolutions de contraintes, etc. et la question de la confiance en la validité de telles preuves se pose. Il serait souhaitable que ces outils non seulement répondent “oui cela termine” mais également fournissent une trace vérifiable de leur preuve.

Totalement indépendemment de la réécriture et de la problématique de la terminaison, il existe des outils spécialisés dans la certification de preuve. Un tel outil est l’assistant de preuve Coq [5, 30], également développé en partie par l’équipe DEMONS. Coq permet d’exprimer des assertions dans une logique très expressive (le calcul des constructions inductives [26]). Il ne cherche pas à prouver ces assertions de manière automatique, au contraire il attend de l’utilisateur la saisie d’un script de preuve, et vérifie que ce script constitue effectivement une preuve de l’assertion. Dans ce contexte, vérifier qu’un script est une preuve de l’assertion est décidable, c’est une opération réalisé par le noyau de Coq qui est une petite partie du code bien comprise et dont on a une très grande confiance.

Ce stage avait donc pour but de générer automatiquement une certification des preuves de terminaison fournis par CiME dans l’assistant de preuve Coq. Au départ deux approches étaient envisageables : la première consiste à définir entièrement en Coq la notion de système de réécriture et leur terminaison, et à prouver dans Coq la correction des critères (de Manna et Ness , et des paires de dépendance) la deuxième possibilité est d’éviter un codage complet des notions en Coq en donnant plutôt un schéma pour généré une nouvelle preuve Coq pour chaque système de réécriture particulier. La première approche a déjà été abordé lors du stage de DEA de Sébastien Hinderer [17] en 2004 : il a prouvé la correction du critère de Manna et Ness combiné avec un ordre par interprétation polynômiale. Vu la complexité de cette première preuve, nous avons préféré la deuxième approche qui semblait plus simple pour obtenir des certificats de preuve utilisant les critères plus récents basés sur les paires de dépendance. Nous reviendrons sur la comparaison entre les deux approches dans la conclusion de ce rapport.

Il est à noter que la formalisation de théorèmes de terminaison (par exemple le théorème de Kruskal) avec des preuves constructives (c’est-à-dire sans utiliser le tiers-exclu) pose des problèmes en soit [16, 9], mais pour nous ce type de problème ne se posera pas.

Dans la suite nous allons présenter successivement les notions et critères utilisés en donnant au fur et à mesure la facon dont on les code en Coq. Dans la section 2, nous présentons des généralités sur les relations bien fondées, puis dans la section 3 les relations sur les algèbres de termes et enfin dans la section 4 le cas des relations de réécriture. Ce qui nous amènera dans le chapitre 5 à la preuve de terminaison par le critère classique, puis dans le chapitre 6 à la preuve de terminaison par le critères des paires de dépendance, et enfin dans le chapitre 7 celle par le critère des graphes de dépendance.

2 Généralités sur les relations bien fondées

Pour une relation binaire R sur un ensemble A, sa terminaison est traditionnellement définie comme l’absence de séquence infinie d’éléments en relation $x_1 R x_2 R \cdots$

Pour les preuves en Coq on préfère définir la terminaison par un prédicat défini inductivement : un élément est fortement normalisable si et seulement si tous ses réduits en un pas le sont.

```
Variable A : Set.

Variable R : A->A->Prop.
```

```
Inductive SN : A -> Prop :=
     SN_intro : forall x:A, (forall y:A, R x y -> SN y)
                            ->SN x.
```

Notons qu'un prédicat similaire (Acc) existe déjà dans la librairie standard de Coq mais où la relation inverse est considérée (R y x) autrement dit sur les entiers pour nous c'est la relation $>$ qui est bien fondée, alors que c'est la relation $<$ qui vérifie Acc.

Montrons que la définition inductive ci-dessus implique la définition habituelle par absence de séquence infinie.

```
Theorem no_infinite_sequence : forall x : A, SN x ->
 ~(exists f : nat -> A, f 0 = x /\ forall i : nat,
 R (f i) (f  (i+1))).
intros.
induction H.
intuition.
elim H1.
intros.
apply (H0 (x0 1)).
intuition.
subst.
apply (H4 0).
exists (fun i => (x0 (i+1))).
intuition.
Qed.
```

On définit maintenant le prédicat `termine` qui exprime qu'une relation donnée est bien fondée en affirmant que tous les éléments sont SN.

```
Definition termine :=
forall x : A, SN x.
```

2.1 Preuve de terminaison à l'aide d'une mesure

La méthode la plus simple pour établir en Coq la terminaison d'une relation est de fournir une mesure qui décroît à chaque réduction par R :

```
Variable f : A -> nat.

Hypothesis H_compat : forall x y:A, R x y -> f x > f y.
```

Le théorème suivant prouve que cette méthode est correcte. Ce théorème est l'analogue du théorème `well_founded_lt_compat` de la librairie standard de Coq, où ce dernier utilise Acc au lieu de SN.

```
Theorem termine_gt_compat : termine.
Proof.
```

```
red.
cut (forall n (a:A), f a < n -> SN a).
intros H a; apply (H (S (f a))); auto with arith.
induction n.
intros; absurd (f a < 0); auto with arith.
intros a ltSma.
apply SN_intro.
intros b ltfafb.
apply IHn.
assert (f a > f b).
apply H_compat.
trivial.
omega.
Qed.
```

2.2 La clôture réflèxive transitive

Pour la définition des systèmes de paires de dépendance on aura besoin d'une relation intermédiaire : La clôture réflèxive transitive. Cette relation Retoile est définie à partir de n'importe quelle relation R de la façon suivante.

Définition 2.1 *La clôture réflèxive transitive* `Retoile` *est définie inductivement comme ceci :*

```
Inductive Retoile : term -> term -> Prop :=
| Retoile1 : forall x : term, Retoile x x
| Retoile2 : forall x  y z : term, R x y -> Retoile y z
             -> Retoile x z.
```

3 Relations sur des algèbres de termes

On s'intéresse maintenant plus particulièrement aux relations sur des algèbres de termes, et à la propriété de monotonie.

3.1 Les algèbres de termes

Une algèbre de termes du premier ordre est définie par une signature, c'est-à-dire un ensemble F de symboles munis d'une arité. Pour chaque signature donné il est possible de définir un type inductif Coq qui représente l'algèbre des termes correspondante.

Définition 3.1 *Le type coq* F`_term` *des termes sur une signature donnée F est le type inductif qui possède autant de constructeurs que de symboles de F, le constructeur associé à chaque symbole f d'arité n étant*

$$f : \underbrace{F\_\mathtt{term} \rightarrow}_{n\ fois} F\_\mathtt{term}$$

Exemple 3.2 *La signature de l'arithmétique de Peano contient la constante* 0*, le symbole unaire s (fonction successeur) et les symboles binaires* $+$ *et* $*$*. L'algèbre de termes correspondante se définit ainsi :*

```
Inductive Peano_term : Set :=
| Zero : Peano_term
| s : Peano_term -> Peano_term
| plus : Peano_term -> Peano_term -> Peano_term
| mult : Peano_term -> Peano_term -> Peano_term.
```

Associé à une algèbre de termes on a systématiquement la relation sous termes immédiat, définie inductivement :

Définition 3.3 *La relation de sous termes immédiats* `F_ST` *est définie inductivement par des constructeurs* `F_ST_f_i` *pour chaque* f *de* F *et chaque i compris entre* 1 *et l'arité n de* f*.*

$$\texttt{F\_ST\_f\_i} : forall\ x1 \cdots xn : \texttt{F\_term}, \texttt{F\_ST}\ (f\ x1 \cdots xn)\ xi$$

Exemple 3.4 *La relation sous termes pour l'algèbre de Peano :*

```
Inductive Peano_ST : term->term->Prop:=
|Peano_ST_s_1 : forall x1 : term, Peano_ST (s x1) x1
|Peano_ST_plus_1 : forall x1 x2 : term, Peano_ST
(plus x1 x2) x1
|Peano_ST_plus_2 : forall x1 x2 : term, Peano_ST
(plus x1 x2) x2
|Peano_ST_mult_1 : forall x1 x2 : term, Peano_ST
(mult x1 x2) x1
|Peano_ST_mult_2 : forall x1 x2 : term, Peano_ST
(mult x1 x2) x2
```

Lemme 3.5 *La relation* `F_ST` *termine toujours.*

Preuve 3.6 *La preuve Coq est simple et systématique :*

```
Lemma Termine_F_ST : termine F_ST.
unfold termine.
intros;induction x;split;intros;inversion H;subst;auto.
Qed.
```

3.2 Généralités sur les relations monotones

Définition 3.7 *Le prédicat F_Monotone est défini pour chaque relation binaire R en construisant la conjonction des formules : pour chaque* f *de* F *et pour chaque i compris entre* 1 *et l'arité n de* f

$$\forall x_1 \cdots x_i \cdots x_n y : \texttt{F\_term}\ ,\ R\ x_i\ y \rightarrow R\ (f\ x_1 \cdots x_n)\ (f\ x_1 \cdots y \cdots x_n)$$

Exemple 3.8 *La relation F_monotone pour l'algèbre de Peano :*

```
Variable R : term -> term -> Prop.

Definition Peano_Monotone : Prop:=
(forall x y: Peano_term, R x y -> R (s x) (s y)) /\
(forall x y z: Peano_term, R x y -> R (plus x z)
(plus y z))/\
(forall x y z : Peano_term, R x y -> R (plus z x)
(plus z y)) /\
(forall x y z: Peano_term, R x y -> R (mult x z)
(mult y z)) /\
(forall x y z : Peano_term, R x y -> R (mult z x)
(mult z y)).
```

Lemme 3.9 *Si R est monotone alors pour tout terme x fortement normalisable, les sous termes directs de x le sont aussi.*

Preuve 3.10 *On prouve ceci par induction sur x par rapport à la relation R. La preuve est :*

```
Lemma STSN : Monotone ->
forall  x :term, SN R x -> forall t : term,(ST x t)
->SN R t.
intros Hmon x HSNx.
apply SN_ind with (R:=R)(P:= fun x =>forall t : term,
ST x t -> SN R t); auto.
intros.
elim Hmon.
intuition.
split;intros.
inversion H1;subst;(eapply H0; [eauto | constructor]).
Qed.
```

3.3 Les réductions dans les sous termes

Pour les preuves de terminaison à base de paires de dépendances on aura besoin d'une relation intermédiaire : la réduction dans les sous termes immédiats. Cette relation RST est définie à partir de n'importe quel relation R de la façon suivante.

Définition 3.11 *La relation de réduction dans les sous termes immédiats* `F_RST` *est définie inductivement par des constructeurs* `F_RST_f_i` *pour chaque f de F et chaque i compris entre* 1 *et l'arité n de f.*

```
F_RST_f_i :forall x_1 ... x_n  y : F_term, F_R  x_i  y -> F_RST (f
x_1 ... x_i ... x_n) (f  x_1 ... y ... x_n)
```

Exemple 3.12 *La relation F_RST pour l'algèbre de Peano :*

```
Inductive RST : term -> term -> Prop :=
| RSTs : forall x1 y : term, R x1 y -> RST (s x1) (s y)
```

```
| RSTplus1 : forall x1 x2 y : term, R x1 y -> RST
(plus x1 x2) (plus y x2)
| RSTplus2 : forall x1 x2 y : term, R x2 y -> RST
(plus x1 x2) (plus x1 y)
| RSTmult1 : forall x1 x2 y : term, R x1 y -> RST
(mult x1 x2) (mult y x2)
| RSTmult2 : forall x1 x2 y : term, R x2 y -> RST
(mult x1 x2) (mult x1 y).
```

Maintenant pour chaque f de F d'arité supérieur ou égal à 1 il faut montrer le lemme suivant :

Lemme 3.13

$$\forall\, x_1 : \texttt{F\_term}, SN\ R\ x_1 \rightarrow \cdots \forall\, x_n : \texttt{F\_term}, SN\ R\ x_n \rightarrow SN\ R\ (f\ x_1\ x_2\ \cdots\ x_n)$$

Preuve 3.14 *Cette preuve se fait par induction sur chacun des sous termes directs de* $(f\ x_1\ x_2 ... x_n)$.

```
Lemma RST_f : forall x1 : F_term, SN R x1 ->
forall x2 : F_term, SN R x2 ->...->forall xn : F_term,
SN R xn ->  SN RST (f x1... xn).
intros x1 H.
generalize H.
apply SN_ind with (R:=R) (P:= fun x1 =>SN R x1 ->
forall x2 : term, SN R x2 ->... -> forall xn : term,
SN R xn -> SN RST (f x1... xn));auto.
clear H ;clear x1.
intros x1 H IHx1 H0 x2 H1.
generalize H1.
apply SN_ind with (R:=R) (P:= fun x2 =>SN R x2->
forall x3 : term, SN R x3 ->... forall xn : term,
SN R xn-> SN RST (f x1... xn));auto.
clear H1;clear x2.
intros x2 H1 IHx2 H2.
generalize H2.
.
.
.
apply SN_ind with (R:=R)(P:=fun xn => SN R xn ->SN RST
(f x1 ... xn);auto.
clear Hn;clear xn.
intros xn Hn IHxn Hn+1.
split;intros;inversion Hn+2;subst.
apply IHx1;auto.
apply IHx2;auto.
.
```

```
.
.
apply IHxn;auto.
Qed.
```

Et enfin grâce au lemme précédent on peut montrer le résultat qui nous permettra de faire une induction sur le système RST : si les sous termes de t sont fortement normalisable pour R alors t est fortement normalisable pour RST.

Lemme 3.15

$$\forall\, x\ :\ term,\ (\forall\, y\ :\ term,\ ST\ x\, y \to SN\ R\ y) - > SN\ RST\ x$$

Preuve 3.16 *La preuve s'effectue par cas à l'aide des lemmes précèdent.*

```
Lemma RSTSN : forall x : term , (forall y : term , ST x y -> SN R y)
   -> SN RST x.
intros.
destruct x .
split;intros;inversion H0.
```

Puis pour chacun des constructeurs f :

```
apply RSTSN_f.
apply H;constructor.
```

et enfin on termine par :

```
Qed.
```

4 Relations de réécriture

Après avoir parlé de ces quelques généralités sur les relations binaires monotones, nous allons parler des systèmes de réécriture dont on cherche à certifier la terminaison.

Définition 4.1 *Un système de réécriture R arbitraire sera représenté en Coq par un type inductif comprenant un constructeur Ri par règle de réécriture de R, i étant compris entre* 1 *et M où M est le nombre de règles de réécriture de R. Ces constructeurs s'écrivent ainsi : Pour chaque règle $l \to r$, i-ème règle de R, avec pour variable $x1 \cdots x_n$, on écrit le constructeur suivant :*

$$Ri : forall\ x1 \cdots xn : \texttt{F\_term}, R\ \bar{l}\ \bar{r}$$

où $\bar{l}$ est la curryfication de l.

On complète ce type inductif à l'aide de règles de monotonie : pour chaque fonction f présente dans la signature F d'arité n, on crée n constructeurs écrit comme ceci :

$$\texttt{R\_f\_i} : forall\ x1 \cdots xn\ y : \texttt{F\_term}, R\ xi\ y \to R\ (f\ x1 \cdots xi \cdots xn)(f\ x1 \cdots y \cdots xn)$$

pour i de 1 à n.

Exemple 4.2 *Pour le système de Peano suivant :*

1. $+(0,x) \to x$
2. $+(s(x),y) \to s(+(x,y))$
3. $*(0,x) \to 0$
4. $*(s(x),y) \to +(*(x,y),y)$

le système R est :

```
Inductive R : Peano_term -> Peano_term -> Prop :=
| R1 :forall x : Peano_term, (R (plus Zero x) x)
| R2 :forall x y : Peano_term, (R (plus (s x) y)
(s (plus x y)))
| R3 : forall x : Peano_term, (R (mult Zero x) Zero)
| R4 : forall x y: Peano_term, (R (mult (s x) y)
(plus (mult x y) y))
| RS :forall x y : Peano_term, (R x y) -> (R (s x)
(s y))
| RP0 :forall  x y z: Peano_term, (R x y) -> (R (plus x z)
(plus y z))
| RP1 :forall x y z : Peano_term, (R x y) -> (R (plus z x)
(plus z y))
| RM0 : forall x y z : Peano_term, (R x y) -> (R (mult x z)
(mult y z))
| RM1 : forall x y z : Peano_term, (R x y) -> (R (mult z x)
(mult z y)).
```

Après ceci nous allons montrer que la relation R est monotone.

Lemme 4.3 *R est monotone.*

Preuve 4.4 *La preuve de la monotonie de R se fait ainsi :*

```
Lemma MonotoneR : Monotone R.
unfold Monotone.
```

Ensuite on se retrouve avec tous les cas de la monotonie que l'on prouve à l'aide des règles de monotonie de R par exemple si on est sur le ième argument du symbole f de la signature F avec n étant l'arité de f :

```
split;intros.(* pour toutes les regles de monotonie sauf la derniere*)
apply R_f_i;auto.
```

Puis on finit bien sûr par :

```
Qed.
```

5 Preuve de Terminaison par le critère classique

Les preuves de terminaison par le critère classique du système de réécriture R consiste à trouver une interprétation polynômiale I de façon à ce que pour tous termes x et y en relation par R on a $I(x) > I(y)$.

5.1 La fonction mesure

Cette fonction est l'interprétation qui nous sera utile dans la preuve à l'aide de termine_gt_compat. Ainsi chaque système de réécriture doit avoir sa propre fonction mesure suivant l'ordre polynômial qui permet de prouver sa terminaison. Dans le système de l'addition de Peano l'interprétation donné par CiME est :

```
[Zero] = 1
[s](x) = x+1
[plus]( x, y) = x +2*y
```

on interprète alors ceci en une fonction Coq donnant la mesure des termes :

```
Fixpoint mesure (t:term) {struct t} : nat :=
match t with
| Zero  => 1
| s x0  => mesure x0 + 1
| plus x0 x1  => mesure x0 +2* mesure x1
end.
```

5.2 La preuve

Une fois la fonction mesure écrite, la preuve consiste à prouver que tout terme t est fortement normalisable, par une induction structurelle sur t. On a alors comme sous-buts à prouver que la partie gauche de chaque règle est supérieure à la partie droite de cette même règle.

Si l'interprétation polynômiale est linéaire (c'est-à-dire ne contient aucune multiplication de variable) : ces sous-buts se prouvent facilement à l'aide des tactiques simpl et omega de Coq. Le schéma général de traduction d'une preuve par le critère classique et un ordre par interprétation polynômiale s'écrit alors

```
Theorem WfPeano : termine R.
apply termine_gt_compat with mesure.
induction 1;simpl;omega.
Qed.
```

En cas de multiplication, la tactique omega ne sais résoudre automatiquement le but. Cela demande donc un travail de l'utilisateur, il serait ici bien sûr souhaitable d'utiliser la technique de Sébastien Hinderer [17].

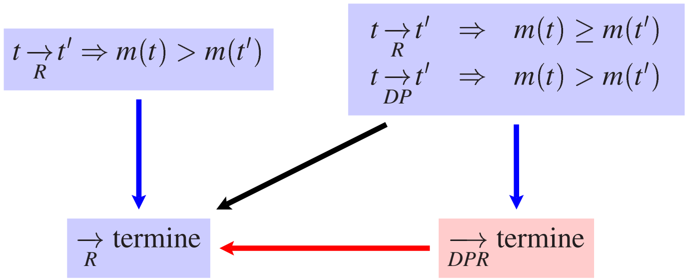


FIG. 1 – Schéma d'une preuve par paires de dépendance

6 Le critère des paires de dépendance

Pour prouver la terminaison de R, on peut utiliser le critère des paires de dépendance [2].

Définition 6.1 *L'ensemble des paires de dépendance est*

$$DP = \{l \to u \mid l \to r \in R, u \text{ sous-terme de } r, root(u) \text{ symbole défini}\}$$

Théorème 6.2 (Arts, Giesl [2]) *S'il existe un ordre bien fondé, stable par instanciation, dont la parti large est monotone, tel que :*

- $l \geq r$ *pour toute règle* $l \to r$ *de* R
- $u > v$ *pour toute paire de dépendance* $u \to v$ *de* DP

alors R *termine*

6.1 Le système DPR de réduction par paires des dépendance

Pour la formalisation en Coq du critère des paires de dépendance, on ne l'exprime pas directement à la manière d'Arts et Giesl, on défini un système intermédiaire de la façon suivante :

$$t \xrightarrow[DPR]{} t' \quad =_{def} \quad t \ (\xrightarrow[R]{\neq\Lambda})^* \ t'' \xrightarrow[DP]{\Lambda} t'$$

Donc une réduction de DPR équivaut à un nombre quelquonque de réduction R dans les sous termes, et une réduction DP en tête. Les hypothèses du critère impliquent

$$\left\{ \begin{array}{lll} t \xrightarrow[R]{} t' & \Rightarrow & m(t) \geq m(t') \\ t \xrightarrow[DP]{} t' & \Rightarrow & m(t) > m(t') \end{array} \right\}$$

et par conséquent

$$t \xrightarrow[DPR]{} t' \Rightarrow m(t) > m(t')$$

et donc $\xrightarrow[DPR]{}$ termine.

La preuve se fait donc en deux étapes, comme indiqué sur la figure 1. La mesure fournie implique la terminaison de système DPR, et nous allons montrer dans la section 6.2 comment on prouve la terminaison de R en supposant celle de DPR, ceci de manière systématique.

Définition 6.3 *Le système* `DPR` *est défini inductivement en coq par des constructeurs* `DPRi` *pour chaque règle i de DPR (c'est-à-dire chaque paire de dépendance). Soit* $f(t_1,\cdots,t_n) \to r$ *la ième règle de* `DPR`*, avec pour variables* $y_1 \cdots y_m$*. Alors le constructeur DPRi s'écrit :*

$$\texttt{DPRi} : forall\ x_1 \cdots x_n\ y_1 \cdots y_m : \texttt{F\_term},$$

puis ensuite pour tous les i de 1 *à n :*

$$\texttt{F\_Retoile}\ x_i\ t_i \to$$

puis enfin on termine avec :

$$\texttt{DPR}\ (f\ x_1 \cdots x_n)\ \bar{r}$$

Exemple 6.4 *La relation DPR pour les systèmes de l'arithmétique de Peano :*

```
Inductive DPR : term-> term-> Prop :=
| DPR1 : forall x y x1 y1: term, Retoile x (s x1)->
Retoile y y1 -> DPR (plus x y)(plus x1 y1)
| DPR2 : forall x y x1 y1: term, Retoile x (s x1) ->
Retoile y y1 -> DPR (mult x y) (plus (mult x1 y1) y1)
| DPR3 : forall x y x1 y1 : term, Retoile x (s x1) ->
Retoile y y1-> DPR (mult x y) (mult x1 y1).
```

6.2 Le passage de la terminaison des paires de dépendances à la terminaison du système

Dans cette partie, nous allons donc établir la preuve de terminaison de *R* en admettant que *DPR* termine. Pour cette preuve nous aurons besoin de plusieurs lemmes.

D'abord nous avons besoin d'un lemme constructeur par symbole constructeur de *F* défini comme ceci :

Lemme 6.5 *Les lemmes Constructeurs sont les preuves que pour des termes donnés qui sont fortement normalisables alors si on rajoute un symbole constructeur f devant, les termes obtenus sont fortement normalisables. Ceci s'écrit en coq :*

```
Lemma constructeur_f :
```

puis pour tout i de 1 *à n l'arité de f*

```
forall  xi : F_term, SN R x_i ->
```

enfin on finit par :

```
SN R (f x1 xn).
```

Preuve 6.6 *La preuve se fait par induction sur chacun des* x_i *sur la relation R de la façon suivante :*

```
Lemma constructor_f : forall x1, SN R x1
->... forall xn :F_term,SN R xn -> SN R (f x1... xn).
```

Ensuite on fait ce qui suit pour i compris entre 1 *et n :*

```
intros xi SNxi.
generalize SNxi.
apply SN_ind with (R:=R) (P:= fun xi =>SN R xi ->
forall x(i+1) : F_term, SN R x(i+1)->... ->
forall xn : F_term, SN R xn ->  SN R (f x1 ... xn));
auto.
clear SNRxi;clear xi.
intros xi H IHxi SNRxi;clear H.
```

Après on regarde les différents cas grâce à :

```
split;intros.
inversion H;subst.
```

Puis on applique les hypothèses d'induction pour chaque i allant de 1 *à n :*

```
apply IHxi;auto.
inversion SNRxi;apply H0;auto.
```

Puis on termine la preuve avec :

```
Qed.
```

Lemme 6.7 *Les lemmes Sous-termes sont les preuves que pour les termes $x_1 \cdots x_n$ donnés, f un symbole constructeur d'arité n et $(f\ x1 \cdots xn)$ est fortement normalisable. Alors pour tout i compris entre* 1 *et n, x_i est fortement normalisable.*

Preuve 6.8 *La preuve de ces lemmes dépend du symbole et de l'argument de cette fonction, par exemple pour le symbole f de F d'arité n et le ième argument de ce symbole :*

```
Lemma Soustermes_f_i : forall x1 ... xn: term, SN R
(f x1 ... xn) -> SN R xi.
intros x1 ... xn H.
apply STSN with (f x1 ... xn).
apply MonotoneR.
auto.
constructor.
Qed.
```

Puis après tous ces lemmes nous pouvons enfin montrer le théorème qui nous intéresse :

Théorème 6.9 *Si DPR termine alors R termine*

Preuve 6.10 *Cette preuve s'effectue grâce à des inductions sur les systèmes ST,puis DPR et enfin RST. Donc le début de la preuve ne change jamais :*

```
Theorem TermineR : termine R.
unfold termine.
intro.
apply SN_ind with (R:=ST).
clear x;intros x H IST;clear H.
generalize IST;clear IST.
apply SN_ind with (R:=DPR)(P:= fun x0 =>(forall y : term,
 ST x0 y -> SN R y) -> SN R x0).
clear x;intros x H IDPR IST;clear H.
generalize IDPR.
clear IDPR.
generalize IST.
apply SN_ind with (R:=(RST R))(P:= fun x1 =>
(forall y : term, ST x1 y -> SN R y) ->
(forall y : term, DPR x1 y -> (forall y0 : term,
ST y y0 -> SN R y0) -> SN R y) -> SN R x1).
clear IST;clear x;intros x H IRST IST IDPR;clear H.
split;intros.
inversion H4;subst.
```

Une fois ce début effectué, on a un sous but par règle de R (dans le même ordre). On va maintenant distingué les sous buts venus des règles de monotonie de R des autres :

- *Les règles de réécriture : la première chose à faire dans tous les sous buts est d'enlever le cas échéant tous les symboles constructeurs en tête à l'aide des lemmes constructor. Si $f1f2\cdots fn$ sont des symboles constructeurs qui commencent le sous but que l'on traite, on fait ceci :*
  `apply constructor_f1;apply constructor_f2;...;apply constructor_fn.`
  Ceci fait disparaître tout les f mais fait apparaître autant de sous buts que la somme des arités des fi. Pour régler ces sous buts il faut distinguer plusieurs cas.
  1. *cas 1 : Si le sous but à prouver est réduit à une variable, alors il faut regarder dans la partie gauche de la règle qu'on est en train de traiter la position de la variable en question. Soit $f1f2...fn$ appartenant à F tel que la variable soit sous ces symboles dans la partie gauche de la règle dans le même ordre. Alors il suffit de faire :*
     ```
     generalize (IST (f2 x2_1 x2_2...x2_(i2-1)
     (f3 x2_(i2+1)... x2_(n2)) (ST_f_j x1_1 ...
     x1_(i1-1) (f2 x1...xn) x1_(i1+1) ... x1_(n1))).
     intros.
     ```
     puis on répète ensuite pour j allant de 2 à n :
     ```
     generalize (Soustermes_fj_ij H0);intros.
     ```
     puis on termine par :
     ```
     auto.
     ```
  2. *cas 2 : Si le but commence par un symbole défini on applique l'hypothèse d'induction IDPR. Ceci fait apparaître 2 sous but de plus, l'un étant réglé facilement à l'aide des règles de DPR (les règles de DPR seront toutes appelées*

les unes après les autres dans l'ordre de DPR),pour l'autre sous but nous devrons recommencer tout le cas des règles non monotones. Tout ceci s'écrit donc en coq avec i un compteur qui est incrémenté de 1 à chaque fois que DPRi est utilisé :

```
apply IDPR.
apply DPRi with x0;apply Retoile1.
intros;inversion H0;subst.
```

– *Les règles de monotonie : Dans chacun de ces sous buts nous avons à montrer la forte normalisation de* $(f\ x_1 \cdots x_(i-1)\ y\ x_(i+1) \cdots x_n)$ *à l'aide de l'hypothèse que* x_i *et y sont en relation par R et des hypothèses d'induction du début. Pour cela il nous faut appeler l'hypothèse d'induction IRST. Ce qui s'écrit en coq :*
  `apply IRST.`
  Cette hypothèse transforme le sous but en 3 sous buts :

  1. *Le premier : est un sous but sur RST et il est réglé par la définition de RST. Soit en coq :*

     ```
     constructor;auto.
     ```

  2. *Le second : est un sous but sur les sous termes et ce sous but est réglé à l'aide de l'hypothèse d'induction IST. Soit en coq :*

     ```
     intros.
     inversion H1;subst.
     ```

     Ensuite on regarde le sous terme du sous but x_j *: Si j est différent de i, on écrit ceci en coq :*

     ```
     generalize (IST xj (ST_f_j x1 ... x(i-1) y x(i+1) ... xn));
       auto.
     ```

     et si j est égal à i, on écrit en coq :

     ```
     generalize (IST y (ST_f_i x1 ...x(i-1) y x(i+1)  xn));intros;
       inversion H2;apply H3;auto.
     ```

  3. *Le dernier : est un sous but sur le système DPR. Pour ceci il faut distinguer 2 cas :*

     (a) *Si f est un symbole défini : Dans ce cas ce sous but se résout à l'aide de l'hypothèse d'induction IDPR et de la définition de DPR. Ce qui s'écrit en coq :*

     ```
     intros.
     apply IDPR.
     inversion H1;subst.
     eapply DPRi;eauto.
     eapply Retoile2;eauto.
     auto.
     ```

     i étant le numéro de la règle de DPR qui a pour symbole de tête de sa partie gauche f.

(b) *Si f est un symbole constructeur : Dans ce cas on obtient une hypothèse absurde. Donc en coq la preuve se fait ainsi :*

```
intros;inversion H1.
```

Une fois tous les sous buts de R prouvés il ne reste plus qu'à prouver la terminaison des systèmes ST, DPR et RST. Ce qui se fait facilement à l'aide des lemmes SNST, SNDPR et SNRSN. Donc en coq la preuve se finit par :

```
apply RSTSN;auto.
apply TermineDPR.
apply TermineST.
Save.
```

6.3 Le critère avec marques

La définition originale du critères des paires de dépendance utilise en faite des termes marqués. On considère une signature étendue où chaque symbole défini f possède une copie $\hat{f}$. Les symboles de têtes des membres gauche et droit des paires sont alors marqués. Cette extension permet plus de souplesse dans les ordres de réduction possible. Pour la formalisation en Coq, les marques sont juste une façon de définir une mesure en faisant un cas particulier pour les symboles de tête.

Exemple 6.11 *Pour le système de Peano CiME donne l'interprétation suivante :*

```
[Zero] = 0;
[s](X0) = X0 + 1;
[plus](X0,X1) = X1 + X0;
[mult](X0,X1) = X1*X0;
['plus`](X0,X1) = X0;
['mult`](X0,X1) = X1*X0 + X0;
```

on l'interpréte en Coq par les fonctions

```
Fixpoint mesure (t:term) {struct t} : nat  :=
match t with
| Zero => 0
| s X0 => mesure X0 + 1
| plus X0 X1  => mesure X1 + mesure X0
| mult X0 X1  => mesure X1* mesure X0
end.

Definition mesuremark (t:term) : nat :=
match t with
|Zero => 0
| s x => mesure x +1
|plus X0 X1 => mesure X0
| mult X0 X1 => mesure X1* mesure X0 + mesure X0
end.
```

et on peut alors prouver successivement

```
Lemma decroissance : forall x y : term ,R x y -> (mesure x) >= mesure y.

Lemma decroissance_etoile : forall x y : term , Retoile R x y
  -> (mesure x) >= mesure y.

Lemma termineDPR : termine DPR.
```

6.4 Validation

La génération de la preuve de

$$termineDPR- > termineR$$

a été codée en CiME. Plusieurs tests ont été effectués, à titre d'exemple, le plus gros comprenait 61 règles, qui donnent 71 paires de dépendance. Le script Coq engendré fait 1905 lignes, et sa compilation avec Coq prend 1min 32 s.

7 Graphe de Dépendance

Pour prouver la terminaison de DPR dans certains cas nous aurons besoin du graphe de dépendance.

Définition 7.1 *Le graphe de dépendance est construit à l'aide du système des paires de dépendance. En effet chacun des sommets du graphe correspondent à une règle de DPR. Ensuite on tire des transitions entre chacun des sommets si la règle correspondante au sommet d'arriver peut suivre la règle du sommet de départ.*

Sommets : DP

Arc de $u \to v$ à $u' \to v'$ si et seulement si il existe σ telle que $v\sigma \underset{R}{\to}^* u'\sigma$

Exemple 7.2 *Par exemple pour l'arithmétique de Peano le graphe est : (figure 2) :*

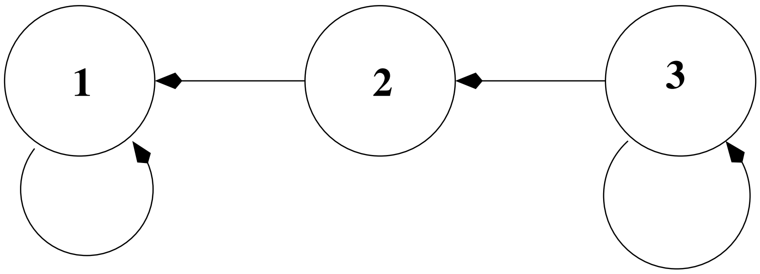


FIG. 2 – Graphe de dépendance

On calcule des composantes fortement connexes $C_1, \ldots, C_k$ du graphe.

Théorème 7.3 (Arts & Giesl, 1998) *Si pour toute composante connexe C_i il existe un ordre de réduction $>_i$ tel que :*
- *$l \geq_i r$ pour les règles de R*
- *$u >_i v$ pour les paires de C_i*

alors R termine.

7.1 Formalisation Coq du critère

On peut extraire les plus grandes parties connexes du graphe, et ainsi créer les systèmes DPRG.

Définition 7.4 *Les systèmes DPRG sont les systèmes qui rassemblent les règles de DPR qui sont rassemblés dans une même partie connexe du graphe de dépendance. Ainsi DPRG s'écrit de la même façon que DPR en enlevant les constructeurs qui ne font pas partis de la partie connexe du graphe que l'on traite.*

Exemple 7.5 *Pour l'arithmétique de Peano il y a 2 DPRG :*

```
Inductive DPRG1 : term-> term-> Prop :=
| DPRG11 : forall x y x1 y1: term, Retoile x (s x1)-> Retoile y y1 ->
 DPRG1 (plus x y)(plus x1 y1).

Inductive DPRG2 : term-> term-> Prop :=
| DPRG23 : forall x y x1 y1 : term, Retoile x (s x1) -> Retoile y y1->
 DPRG2 (mult x y) (mult x1 y1).
```

7.2 Le passage de la terminaison de DPRG à la terminaison du système de paires de dépendances.

Lemme 7.6 *Si chaque DPRGi termine alors le système DPR termine.*

Preuve 7.7 *La preuve de ce lemme s'effectue à l'aide d'induction sur les n différents systèmes DPRG :*

```
Theorem TermineDPR : termine DPR.
unfold termine.
intro.
```

Ensuite on fait les inductions sur tous les systèmes DPRG. Pour i étant compris entre 1 et n :

```
apply SN_ind with (R:=DPRGi)(P:= fun x =>
(forall y : term, DPRG1 x y -> SN DPR y) -> ...
 ->(forall y : term, DPRG(i-1) x y -> SN DPR y)
-> SN R x.
clear x.
intros x H ;clear H.
```

Ensuite on introduit les hypothèses d'induction puis on regarde chaque cas de DPR.

```
intros IDPRG1 IDPRG2 ... IDPRGn.
split;intros;inversion H;subst.
```

Cela à fait apparaître autant de sous buts qu'il existe de règles du système DPR. Pour prouver ces sous buts il faut distinguer 2 cas :

1. *Si le sous but est issu d'une règle qui n'appartient à aucun des DPRG et si cette règle n'a pas d'arêtes sortante dans le graphe de dépendance. Dans ce cas il suffit de regarder ce qui peut suivre cette règle. Ce qui s'écrit en coq :*

   ```
   split;intros y F;inversion F;subst;clear F.
   ```

2. *Si le sous but est issu d'une règle qui n'appartient à aucun des DPRG et si cette règle a une arête sortante dans le graphe de dépendance. Dans ce cas il faut faire une nouvelle induction sur les systèmes sortant de cette règle. Ce qui s'écrit en coq :*

   ```
   assert (exists x0 : term , ... exists xn : term ,
   (f x0 x1 ... xn) = ( f ST1 ST2 ... STn).
   exists (ST1);.... exists (STn);auto.
   generalize H2.
   apply SN_ind with (R:=DPRG_1)(P:= fun x =>
   (exists x0 : term,... exists xn : term ,
   (f x0 ... xn ) = x) -> SN DPR x).
   intros x F IDPRG' Q;inversion_clear Q;inversion_clear H3;subst;clear F.
   split;intros y F ;inversion F.
   apply IDPRG'.
   constructor;auto.
   exists x_n+1;...;exists x_2n+1;auto.
   apply TermineDPRG_1.
   ```

   ou n est l'arité de f, ST sont les sous termes directs du termes dont on cherche la forte normalisation.

3. *Si le sous but est issu d'une règle appartenant à DPRGi Dans ce cas il suffit d'utiliser l'hypothèse d'induction équivalente :*

   ```
   apply IDPRi.
   constructor;auto.
   ```

Une fois tous ces cas prouvés il ne reste plus qu'à prouver la terminaison des systèmes DPRG. Soit en coq :

```
apply TermineDPRG1.
.
.
.
apply TermineDPRGn.
Qed.
```

7.3 Validation

La génération de la preuve de

$$termineDPRG1- > ...termineDPGRn- > termineDPR$$

a été codée en CiME. Mais comporte encore quelques problèmes, en effet certains tests ne passent pas. Cela est du à 2 problèmes différents :

1. d'une part un problème d'implémentation. En effet un problème subsiste lorsqu'un sommet du graphe de dépendance, n'appartenant pas à une des composantes connexe de celui ci, possède une arête sortante.
2. d'autre part, un problème plus fondamental. L'absence d'arête entre deux sommets est due à l'inexistence de substitution σ vérifiant

$$v\sigma \underset{R}{\rightarrow}^{*} u'\sigma$$

inexistence qui a été trouvée par CiME. Or l'existence d'une telle substitution est un problème indécidable, et CiME a en fait effectué une approximation à base de test d'unification. Or l'implantation actuelle ne fournit pas de trace de ce test, qui serait nécessaire.

8 critère de graphe de dépendance affiné

Pour prouver la terminaison de DPRG on peut utiliser deux méthodes : on peut bien sur utiliser la méthode vu en 5. Mais il arrive parfois qu'aucun ordre qui fasse décroître strictement toutes les règles de DPRG.

Théorème 8.1 (collectif) *Si pour toute partie fortement connexe C_i il existe un ordre de réduction $>_i$ tel que :*

- $l \geq_i r$ *pour les règles de* R
- $u \geq_i v$ *pour les paires de* C_i *et* $u >_i v$ *pour au moins une de ces paires.*

alors R termine.

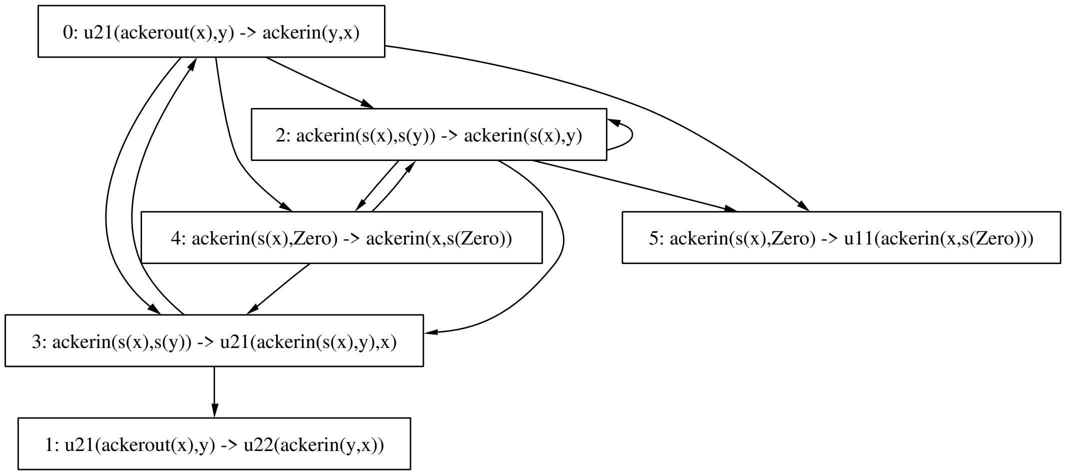
Remarquer la subtilité : on parle ici de partie connexe, pas de composante (qui sont les parties connexes maximales). Dans la pratique, pour appliquer ce critère, CiME procède de la façon suivante : pour chaque composante connexe, il recherche un ordre qui fait décroître au sens large les paires et strictement au moins une, puis considère le sous-graphe des paires qui décroissent seulement au sens large, et recommence le processus récursivement (y compris le calcul des composantes connexes).

Exemple 8.2 *Le système suivant est issu de la traduction [25] du programme logique*

```
ackermann(0,N,s(N)).
ackermann(s(M),0,Val) :- ackermann(M,s(0),Val).
ackermann(s(M),s(N),Val) :-
             ackermann(s(M),N,Val1), ackermann(M,Val1,Val).
```

$$\begin{array}{rcl}
ackerin\ 0\ x_0 & \rightarrow & ackerout\ (s\ x_0) \\
ackerin\ (s\ x_0)\ 0 & \rightarrow & u11\ (ackerin\ x_0\ (s\ 0)) \\
u11\ (ackerout\ x_0) & \rightarrow & ackerout\ x_0 \\
ackerin\ (s\ x_0)\ (s\ x_1) & \rightarrow & u21\ (ackerin\ (s\ x_0)\ x_1)\ x_0 \\
u21\ (ackerout\ x_0)\ x_1 & \rightarrow & u22\ (ackerin\ x_1\ x_0) \\
u22\ (ackerout\ x_0) & \rightarrow & ackerout\ x_0
\end{array}$$

Le graphe de dépendance est le suivant



La trace texte de la preuve de CiME est

```
the dependency graph has one strongly connected component:
  0: u21(ackerout(V_0),V_1) ; ackerin(V_1,V_0)
  2: ackerin(s(V_0),s(V_1)) ; ackerin(s(V_0),V_1)
  3: ackerin(s(V_0),s(V_1)) ; u21(ackerin(s(V_0),V_1),V_0)
  4: ackerin(s(V_0),Zero) ; ackerin(V_0,s(Zero))
  0 -> 2, 0 -> 3, 0 -> 4, 2 -> 2, 2 -> 3, 2 -> 4, 3 -> 0, 4 -> 2, 4 -> 3,
  On this component, with the interpretation
    [Zero] = 0;
    [s](X0) = X0 + 1;
    [ackerout](X0) = 0;
    [u22](X0) = 0;
    [u11](X0) = 0;
    [ackerin](X0,X1) = X0;
    [u21](X0,X1) = X1;

  the subgraph of pairs that are only weakly decreasing
  has one strongly connected component:
    2: ackerin(s(V_0),s(V_1)) ; ackerin(s(V_0),V_1)  2 -> 2,
    On this component, the interpretation
    [Zero] = 0;
    [s](X0) = X0 + 1;
    [ackerout](X0) = 0;
    [u22](X0) = 0;
    [u11](X0) = 0;
    [ackerin](X0,X1) = X1;
    [u21](X0,X1) = 0;
     strictly decreases on every cycle
```

la première interprétation fait décroître strictement les paires 3 et 4, et seulement largement les paires 0 et 2. Le sous-graphe $\{0,2\}$ *contient alors seulement la composante* *connexe $\{2\}$, qui est traitée par la 2e interprétation. On constate bien que chaque partie connexe est prise en compte par l'une ou l'autre des interprétations, suivant qu'elle contient l'une des paires 3 ou 4 ou non.*

Pour ce critère affiné, j'ai effectué une preuve Coq pour cet exemple particulier. L'implantation ne fonctionne bien sur pas encore, puisque celle du critère de graphe non affiné non plus.

8.1 Formalisation Coq

Dans ces cas il nous faut tout d'abord trouver un ordre *mesureDPRG* tel que pour toutes les règles de *DPRG* : $x \to y$ on a $mesureDPGx \geq mesureDPGy$. Donc cette mesure vérifie les lemmes suivants :

Lemme 8.3 *Si x et y deux termes sont en relation par R alors, mesureDPG x $\geq$ mesureDPG y.*

Preuve 8.4 *La preuve se fait directement par induction. Soit en coq :*

```
Lemma decroissance : forall x y : F_term,R x y ->
(mesureDP x) >= mesureDP y.
induction 1;simpl;auto with *.
Qed.
```

Lemme 8.5 *Si x et y deux termes sont en relation par Retoile alors,*

$$mesureDPG\,x \;\geq\; mesureDPG\,y$$

Preuve 8.6 *La preuve se fait directement par induction. Soit en coq :*

```
Lemma decroissance_etoile : forall x y : F_term, Retoile  x y ->
(mesureDP x) >= mesureDP y.
induction 1.
auto.
assert (mesureDP x >= mesureDP y);auto with *.
apply decroissance.
trivial.
Qed.
```

Ensuite on différencie les règles de *DPRG* : $x \to y$ où l'interprétation est large ($mesureDPx = mesureDPy$) à celle où l'interprétation est stricte ($mesureDPx > mesureDPy$).

Pour prouver que DPRG termine il nous faut plusieurs systèmes intermédiaires :

Les systèmes DPRG_i :

Définition 8.7 *Le système DPRG_i est le regroupement des règles de DPRG où l'interprétation est large.*

Exemple 8.8 *Par exemple pour le système d'Ackerman :*
le système DPRG_i est :

```
Inductive DPRG_i : Ackerman_term->Ackerman_term->Prop :=
| DPRGG4 : forall x y x1 y1 : Ackerman_term, Retoile x (s x1) ->
Retoile y (s y1)-> DPRG_i (ackerin x y) (ackerin (s x1) y1)
| DPRGG3 : forall x y x1 y1: Ackerman_term, Retoile x (s x1) ->
Retoile y (s y1) -> DPRG_i (ackerin  x y) (u21 (ackerin (s x1) y1) x1).
```

Lemme 8.9 *Le système DPRG_i termine.*

Preuve 8.10 *Pour prouver ce lemme, il faut utiliser la même méthode que pour prouver le système DPR.*

Lemme 8.11 *Si x et y deux termes en relation par DPRG_i, alors mesureDPGx $\geq$ mesureDPGy.*

Preuve 8.12 *La preuve se fait sur les cas de DPRG_i.*
On fait apparaître tous les cas de DPRG_i.

```
Lemma largeDPRG_i : forall x y : term, DPRG_i x y -> mesureDP x >= mesureDP y.
intros.
inversion H;subst.
```

Puis on prouve chaque cas c'est-à-dire une fois pas constructeur de DPRG_i à l'aide de ceci.

```
generalize (decroissance_etoile H0).
simpl.
omega.
```

Puis on termine par cela.

```
Qed.
```

Le système DPRGG :

Définition 8.13 *Le système DPRGG est le regroupement des règles où l'interprétation est stricte.*

Exemple 8.14 *Par exemple pour le système d'Ackerman le système DPRGG est :*

```
Inductive DPRGG : Ackerman_term->Ackerman_term->Prop :=
| DPRGGG2 : forall x x1  : Ackerman_term, Retoile x (s x1) -> DPRGG
(ackerin x Zero) (ackerin x1 (s Zero))
| DPRGGG6 : forall x y x1 y1: Ackerman_term, Retoile x (ackerout x1)
-> Retoile y y1 -> DPRGG (u21 x y) (ackerin y1 x1).
```

Lemme 8.15 *Le système DPRGG termine.*

Preuve 8.16 *La preuve s'effectue par l'interprétation polynômiale mesureDP comme dans la section 5.*

Lemme 8.17 *Si x et y deux termes en relation par DPRGG, alors (mesureDPG x) > (mesureDPG y).*

Preuve 8.18 *La preuve se fait sur les cas de DPRGG.*
On fait apparaître tous les cas de DPRGG.

```
Lemma strictDPRGG : forall x y : F_term, DPRGG x y -> mesureDPmark x
> mesureDPmark y.
intros.
inversion H;subst.
```

Puis on prouve chaque cas c'est-à-dire une fois par constructeur de DPRGGG à l'aide de ceci.

```
generalize (decroissance_etoile H0).
simpl.
omega.
```

Puis on termine par cela.

```
Qed.
```

Le système DPRGGn :

Définition 8.19 *Le système DPRGGn est le système où un pas de DPRGGn est équivalent à n pas de DPRG_ipour*$0 < i <= k$*.*

Exemple 8.20 *Par exemple pour le système d'Ackerman le système DPRGGn est :*

```
Inductive DPRGGn : nat->F_term -> F_term -> Prop :=
| DPRGGn1 : forall x  y z : F_term,forall n : nat,n <> 0 ->DPRG_1 x y
-> DPRGGn (n-1) y z -> DPRGGn n x z
.
.
.
| DPRGGnk :  forall x  y z : F_term,forall n : nat,n <> 0 ->DPRG_k x y
-> DPRGGn (n-1) y z -> DPRGGn n x z
| DPRGGn0 : forall x : F_term,forall n : nat, n =0 -> DPRGGn n x x.
```

Lemme 8.21 *Si pour tous n différents de 0,x et y deux termes en relation par (DPRGGn n), alors mesureDPGx* $\geq$ *mesureDPGy.*

Preuve 8.22 *La preuve se fait par induction sur DPRGGn, puis par cas sur n. Ce qui s'écrit en coq :*

```
Lemma largeDPRGGn : forall x y : F_term, forall n : nat, n <>0
->DPRGGn n x y -> mesureDPmark x >= mesureDPmark y.
intros.
induction H0.
generalize (LargeDPRG_i H1).
destruct (n-1).
inversion H2.
```

```
elim H3;auto.
subst.
omega.
assert (Datatypes.S n0 <> 0).
auto.
generalize (IHDPRGGn H3).
omega.
elim H;auto.
Qed.
```

Le système DPRGGG :

Définition 8.23 *Le système DPRGGG est le système qui regroupe le système DPRGGn et le système DPRGG. En effet un pas de DPRGGG est équivalent pour un n donné à un pas de (DPRGGnn) suivis d'un pas de DPRGGG.*

Exemple 8.24 *Par exemple pour le système d'Ackerman le système DPRGGG est :*

```
Inductive DPRGGG : Ackerman_term -> Ackerman_term -> Prop :=
| DPRGGG1 : forall x y z: Ackerman_term,(exists n : nat,  DPRGGn n x
y) ->DPRGG y z -> DPRGGG x z.
```

Théorème 8.25 *DPRGGG termine.*

Preuve 8.26 *La preuve se fait par induction sur le n.*

```
Theorem TermineDPRGGG : termine DPRGGG.
apply termine_gt_compat with mesureDPmark.
intros.
inversion H;subst.
inversion H0.
induction x0.
(*cas n =0*)
inversion H2.
elim H3;auto.
generalize (strictDPRGG H1);auto.
(*cas n <>0*)
assert (Datatypes.S x0 <> 0).
auto.
generalize (largeDPRGGn H3 H2).
intro.
generalize (strictDPRGG H1).
omega.
Qed.
```

Une fois tous ces systèmes créer la preuve que DPR termine n'a besoin plus que d'un lemme :

Lemme 8.27 *Si pour tout x, y, z des termes tel que x et y sont en relation par DPRGG et y z sont en relation par DPRGGG alors x et z sont en relation par DPRGGG.*

Preuve 8.28 *La preuve se fait par induction sur DPRGGG. Et elle s'écrit en coq :*

```
Lemma DPRGtoDPRGGG : forall x y z: F_term, DPRGGG y z -> DPRGG x y
-> DPRGGG x z.
intros.
induction H.
inversion H.
assert (x1+1 <>0).
omega.
replace x1 with (x1 + 1-1) in H2.
generalize (DPRGGn1 H3 H0 H2).
intros.
generalize (DPRGGG1).
intros.
apply H5 with y;auto.
exists (x1+1).
auto.
omega.
Qed.
```

Enfin nous pouvons montrer le Théorème suivant :

Théorème 8.29 *DPRG termine.*

Preuve 8.30 *La preuve se fait, grâce aux inductions sur DPRGGG et sur DPRGG, sur cas de DPRG.*
On commence par faire les inductions sur DPRGGG puis sur DPRGG et faire les cas de DPRG.

```
Theorem TermineDPRG : termine DPRG.
unfold termine.
intro.
apply SN_ind with (R:=DPRGGG).
intros.
generalize H0.
clear H0.
clear H.
apply SN_ind with (R:= DPRGG)(P:= fun x0 =>(forall y : term, DPRGGG
x0 y -> SN DPRG y) -> SN DPRG x0).
intros.
split;intros.
inversion H2;subst.
```

Pour chacun des sous buts la règle que l'on traite à pour numéro i et on créer une liste l d'entiers initialisée à liste vide. Puis ensuite deux cas sont possible :

1. *Si la règle de DPRG que l'on traite appartient à DPRGGG alors on utilise l'hypothèse d'induction sur DPRGGG comme ceci :*

   ```
   apply H1.
   eapply DPRGGG1.
   ```

   Ensuite pour j parcourant l, on écrit :

   ```
   exists (taille de l).
   eapply DPRGGn1;auto.
   apply DPRGGj.
   ```

   Puis une fois la liste vide obtenue on termine par :

   ```
   exists 0.
   eapply DPRGGn0;auto.
   constructor;auto.
   ```

2. *Si la règle que l'on traite appartient à DPRGG alors il faut distinguer 2 autre cas :*
   (a) *Si la règle que l'on traite n'appartient pas à un cycle du graphe de dépendance de DPRGG alors on ajoute i en tête de la liste l puis on regarde les cas qui peuvent suivre cette règle. On écrit donc en coq :*

   ```
   split;intros;inversion H5;subst.
   ```

   (b) *Si la règle que l'on traite appartient à un cycle du graphe de dépendance de DPRGG alors on utilise l'hypothèse d'induction sur DPRGG.*

   ```
   apply H0.
   apply DPRGGi;auto.
   intros.
   eapply H1.
   eapply DPRGG+;eauto.
   apply DPRGGi;auto.
   ```

Enfin on termine la preuve grâce à la terminaison des systèmes DPRGGG et DPRGG. Ce qu'on écrit en coq :

```
apply TermineDPRGG.
apply TermineDPRGGG.
Qed.
```

9 Conclusion

Le travail réalisé au cours de ce stage a consisté à proposer un schéma de preuve des critères des paires de dépendance et du graphe de dépendance, et de les implémenter dans CiME. CiME peut donc maintenant générer automatiquement une preuve de terminaison sous la forme d'un fichier Coq.

Le problème rencontré par cette méthode est qu'il faut générer un fichier de preuve pour chaque systèmes de réécriture, fichier dont la taille croit rapidement avec la taille du système. Sur ce point, il aurait été plus intéressant d'utiliser la première approche mentionnée dans l'introduction, à savoir de faire une fois pour toute une preuve générique des critères utilisés. Maintenant que nous avons dégagé un schéma général de preuve (en particulier à l'aide de la relation RST de réduction dans les sous termes), une telle preuve générique semble plus abordable en prenant comme point de départ la formalisation des systèmes de réécriture de Sebastien Hinderer dans son stage de DEA [17].

Parmi les extensions possibles de ce travail, on peut signaler que de nouveaux critères de terminaison ont été proposés récemment, qui sont encore plus puissant que ceux traités dans ce stage [32, 33, 31]. En particulier ces critères ne demandent plus d'orienter au sens large toutes les règles du système il serait donc intéressant de savoir produire des traces pour de tels preuves.

Références

[1] T. Arts and J. Giesl. Automatically proving termination where simplification orderings fail. In M. Bidoit and M. Dauchet, editors, *Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, Lille, France, Apr. 1997. Springer-Verlag.

[2] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236 :133–178, 2000.

[3] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[4] A. Ben Cherifa and P. Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9 :137–159, 1987.

[5] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development*. Springer-Verlag, 2004. To appear.

[6] C. Borralleras and A. Rubio. Termptation. In Rubio [28], pages 61–63. Technical Report DSIC II/15/03, Universidad Politécnica de Valencia, Spain.

[7] E. Contejean, C. Marché, B. Monate, and X. Urbain. Proving termination of rewriting with C*i*ME. In Rubio [28], pages 71–73. `http://cime.lri.fr`.

[8] M. Davis. Unsolvable problems. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 567–594. North-Holland, 1977.

[9] J. E. Dawson and R. Goré. A general theorem on termination of rewriting. In J. Marcinkowski and A. Tarlecki, editors, *Proceedings of the Annual Conference of the European Association for Computer Science Logic*, Lecture Notes in Computer Science, Karpacz, Poland, Sept. 2004. Springer-Verlag.

[10] N. Dershowitz. A note on simplification orderings. *Information Processing Letters*, 9(5) :212–215, Nov. 1979.

[11] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1) :69–115, Feb. 1987.

[12] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. North-Holland, 1990.

[13] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Commun. ACM*, 22(8) :465–476, Aug. 1979.

[14] O. Fissore, I. Gnaedig, and H. Kirchner. CARIBOO : An Induction Based Proof Tool for Termination with Strategies. In *Proceedings of the Fourth International Conference on Principles and Practice of Declarative Programming (PPDP)*, Pittsburgh, USA, October 2002. ACM Press. `http://www.loria.fr/~fissore/CARIBOO`.

[15] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. AProVE : A system for proving termination. In Rubio [28]. `http://www-i2.informatik.rwth-aachen.de/AProVE`.

[16] J. Goubault-Larrecq. Well-founded recursive relations. In L. Fribourg, editor, *Proceedings of the Annual Conference of the European Association for Computer Science Logic*, volume 2142 of *Lecture Notes in Computer Science*, pages 484–497, Paris, France, Sept. 2001. Springer-Verlag.

[17] S. Hinderer. Certification des preuves de terminaison par interprétations polynomiales. Mémoire de DEA, Université Henri Poincaré, 2004.

[18] N. Hirokawa and A. Middeldorp. Tsukuba termination tool. In R. Nieuwenhuis, editor, *14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *Lecture Notes in Computer Science*, pages 311–320, Valencia, Spain, June 2003. Springer-Verlag.

[19] S. Kamin and J.-J. Lévy. Two generalizations of the recursive path ordering. Available as a report of the department of computer science, University of Illinois at Urbana-Champaign, 1980.

[20] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[21] D. S. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Mathematics Department, Louisiana Tech. Univ., 1979. Available at `http://perso.ens-lyon.fr/pierre.lescanne/not_accessible.html`.

[22] S. Lucas. Mu-term : A tool for proving termination of context-sensitive rewriting. In V. van Oostrom, editor, *15th International Conference on Rewriting Techniques and Applications*, volume 3091 of *Lecture Notes in Computer Science*, pages 200–209, Aachen, Germany, June 2004. Springer-Verlag.

[23] Z. Manna and S. Ness. On the termination of markov algorithms. In *Proc. third Hawaii International Conference on Systems Sciences*, pages 789–792, Honolulu, HI, 1970. `http://perso.ens-lyon.fr/pierre.lescanne/not_accessible.html`.

[24] C. Marché and X. Urbain. Termination of associative-commutative rewriting by dependency pairs. In T. Nipkow, editor, *9th International Conference on Rewriting Techniques and Applications*, volume 1379 of *Lecture Notes in Computer Science*, pages 241–255, Tsukuba, Japan, Apr. 1998. Springer-Verlag.

[25] E. Ohlebusch, C. Claves, and C. Marché. TALP : A tool for the termination analysis of logic programs. In L. Bachmair, editor, *11th International Conference on Rewriting Techniques and Applications*, volume 1833 of *Lecture Notes in Computer Science*, pages 270–273, Norwich, UK, July 2000. Springer-Verlag. Available at `http://bibiserv.techfak.uni-bielefeld.de/talp/`.

[26] C. Paulin-Mohring. Inductive definitions in the system COQ. In *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 328–345. Springer-Verlag, 1993.

[27] D. A. Plaisted. A recursively defined ordering for proving termination of term rewriting systems. Unpublished report R-78-943, University of Illinois, Urbana, IL, 1978.

[28] A. Rubio, editor. *Extended Abstracts of the 6th International Workshop on Termination, WST'03*, June 2003. Technical Report DSIC II/15/03, Universidad Politécnica de Valencia, Spain.

[29] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003. `http://www.cs.vu.nl/~terese/`.

[30] The Coq Development Team. *The Coq Proof Assistant Reference Manual – Version V8.0*, Apr. 2004. `http://coq.inria.fr`.

[31] R. Thiemann, J. Giesl, and P. Schneider-Kamp. Improved modular termination proofs using dependency pairs. In *Proceedings of the 2nd International Joint Conference on Automated Reasoning*, volume 3097 of *Lecture Notes in Artificial Intelligence*, pages 75–90, Cork, Ireland, 2004. Springer-Verlag.

[32] X. Urbain. Automated incremental termination proofs for hierarchically defined term rewriting systems. In R. Goré, A. Leitsch, and T. Nipkow, editors, *International Joint Conference on Automated Reasoning*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 485–498, Siena, Italy, June 2001. Springer-Verlag.

[33] X. Urbain. Modular and incremental automated termination proofs. *Journal of Automated Reasoning*, 2004.