Dossier d'habilitation à diriger des recherches

Document 1 : présentation synthétique de mes activités

Fonctions, réécriture et preuves : terminaison et certification

Frédéric Blanqui (INRIA)

http://www-rocq.inria.fr/~blanqui/

4 novembre 2010
(mis-à-jour le 28 mars 2012)

Table des matières

Dans ce document, je présente une synthèse de mon travail de recherche et d'encadrement depuis l'obtention de mon doctorat en septembre 2001 [Bla01b]. Il présente en particulier une introduction à mon domaine de recherche, une liste exhaustive de mes travaux de recherche et de mes développements, commentée et classée par sujet, et se termine par un ensemble de perspectives de recherche.

J'essaie d'expliquer de manière informelle les résultats que j'ai obtenus et comment je les obtenus. Pour plus de détails, je renvoie le lecteur vers la monographie [Bla12] qui accompagne ce dossier et les articles où ces résultats sont présentés. J'explique également ce qui a stimulé ces recherches et ces développements logiciels et à quelles suites ils ont donné lieu dont, en particulier, les travaux de thèse que j'ai co-encadrés.

Toutes mes publications et développements logiciels sont accessibles librement depuis l'URL `http://www-rocq.inria.fr/~blanqui/` (ou `http://hal.inria.fr/` pour les publications).

1 Parcours professionnel

Après la soutenance de ma thèse en septembre 2001 à l'Université Paris-Sud, j'ai été jusqu'en août 2002 au laboratoire d'informatique de l'Université de Cambridge en Angleterre grâce à une bourse INRIA. Là-bas, avec Larry Paulson, je me suis initié aux problèmes posés par les protocoles cryptographiques, leur formalisation et la preuve de leur correction dans l'assistant à la preuve Isabelle [NPW02].

J'ai ensuite poursuivi mon postdoc au laboratoire d'informatique de l'École Polytechnique jusqu'en septembre 2003. Là-bas, j'ai notamment pris part au projet RNTL Averroes et développé un prototype étendant l'assistant à la preuve Coq [Coq10] avec de la réécriture.

En octobre 2003, j'ai pris mes fonctions de Chargé de Recherche de l'INRIA au LORIA à Nancy dans l'équipe-projet Prothéo de Claude Kirchner. Là-bas, durant 5 ans, j'ai pris part à différents projets (ACISI Modulogic, ARC Quotient, etc), organisé plusieurs workshops (Coq et réécriture, Nancy-Saarbrücken, etc) et co-encadré les thèses de Colin Riba[1] [Rib07a] et Pierre-Yves Strub[2] [Str08], ainsi que la thèse de Cody Roux[3] [Rou11] à partir d'octobre 2007.

En septembre 2008, j'ai quitté Nancy pour aller, en tant qu'expatrié INRIA à Pékin en Chine, aider à la formation avec Jean-Pierre Jouannaud et Vania Joloboff d'une nouvelle équipe-projet au sein du LIAMA, le laboratoire franco-chinois de recherche en informatique, automatique et mathématiques appliquées, co-financé par l'INRIA, le CNRS et l'Académie des Sciences Chinoises. Cette nouvelle équipe-projet, baptisée Formes, a pour objectif le développement d'outils de simulation et de vérification de systèmes embarqués en coopération avec des collègues chinois de l'Université de Tsinghua.

1. `http://perso.ens-lyon.fr/colin.riba/`
2. `http://pierre-yves.strub.nu/`
3. `http://pareo.loria.fr/dokuwiki/doku.php?id=codyroux`

2 Enseignement

Voir `http://www-rocq.inria.fr/~blanqui/teaching.html`.

Pendant ma thèse, de 1998 à 2001, j'ai été moniteur à l'IUT d'informatique de l'Université Paris-Sud. J'y ai en particulier fait des TDs de programmation en Cobol et en Ada. J'ai également donné quelques TDs de programmation en Caml à l'Université Paris-Sud. Enfin, j'ai aussi été interrogateur en mathématiques dans les classes préparatoires des lycées Paul Valéry, Condorcet et Saint Louis à Paris.

De 2002 à 2007, je n'ai pas fait d'enseignement.

J'ai repris l'enseignement avec mon arrivée en Chine à l'Université de Tsinghua en 2008 :

- Mars 2008 : Cours d'introduction au λ-calcul typé et à la logique (transparents disponibles sur ma page web).
- Novembre 2008 - Février 2009 : Cours d'introduction à Coq.
- Mars 2009 : Cours sur la méta-théorie du Système F.
- Août 2009 : TDs de Coq à l'école d'été sur Coq et la preuve de programmes organisée par l'équipe Formes.
- Août 2010 : Cours sur la vérification et l'inférence de types, et la terminaison de la β-réduction dans le λ-calcul simplement typé (transparents disponibles sur ma page web), et TDs de Coq à l'école d'été sur Coq et la théorie des types organisée par l'équipe Formes.

3 Encadrement

Voir `http://www-rocq.inria.fr/~blanqui/students.html`.

3.1 Encadrements de stages

- Mars 2003 - Septembre 2003 : Co-encadrement avec Jean-Pierre Jouannaud du stage de master de Pierre-Yves Strub (Paris VII) sur l'extension du calcul des constructions avec des procédures de décision [Str03].
- Février 2004 - Juin 2004 : Stage de master de Sébastien Hinderer (Nancy I) sur la certification de preuves de terminaison basée sur des interprétations polynomiales dans $\mathbb{Z}$ [Hin04].
- Juin 2009 - Juillet 2009 : Stage d'Antoine Taveneaux (ENS Lyon) sur la complexité des systèmes de réécriture.
- Juin 2009 - Juillet 2009 : Stage de Julien Bureaux (ENS Paris) sur la certification de la terminaison de programmes Haskell.
- Mai 2010 - Octobre 2010 : Encadrement d'un stage de niveau master de Kim-Quyen Ly (Vietnam) sur la certification de preuves de terminaison basées sur des interprétations polynomiales dans $\mathbb{Q}$.

3.2 Co-encadrements de thèses

- Avril 2004 - Décembre 2007 : Co-encadrement avec Claude Kirchner du travail de thèse de Colin Riba sur la confluence et la terminaison des systèmes de réécriture d'ordre supérieur [Rib07a]. Colin est maintenant maître de conférence à l'ENS Lyon sur une chaire CNRS. Voir les Sections 7.1.3 et 7.2.3 pour plus de détails.
- Février 2005 - Juillet 2008 : Co-encadrement avec Jean-Pierre Jouannaud du travail de thèse de Pierre-Yves Strub sur l'extension du calcul des constructions avec des procédures de décision [Str08]. Pierre-Yves est maintenant ingénieur de recherche au laboratoire commun INRIA-Microsoft. Voir la Section 7.1.5 pour plus de détails.
- Octobre 2007 - Février 2011 : Co-encadrement avec Claude Kirchner et Gilles Dowek du travail de thèse de Cody Roux sur la terminaison des systèmes de réécriture d'ordre supérieur [Rou11]. Voir la Section 7.2.3 pour plus de détails.
- Novembre 2009 - Septembre 2011 : Co-encadrement avec Jean-François Monin et Vania Joloboff du travail de thèse de Xiaomu Shi sur la certification d'un simulateur de processeur ARM. Voir la Section 8.5 pour plus de détails.
- Novembre 2010 - : Co-encadrement avec Jean-François Monin du travail de thèse de Kim-Quyen Ly sur la certification de preuves de terminaison. Voir la Section 8.3 pour plus de détails.

4 Workshops, écoles d'été et projets

4.1 Organisation de workshops

Voir `http://www-rocq.inria.fr/~blanqui/workshops.html`.

- 1st Workshop on Coq and rewriting, École Polytechnique, 27 juin 2003 (environ 30 participants).
- 1st Nancy-Saarbrücken workshop on logic, proofs and programs, Nancy, 17-18 juin 2004 (36 participants).
- 2nd Workshop on Coq and rewriting, École Polytechnique, 22-24 septembre 2004 (52 participants).
- QSL Workshop on functional programming and verification, Nancy, 12 mai 2005 (environ 30 participants).
- 3rd Nancy-Saarbrücken Workshop on logic, proofs and programs, Nancy, 13-14 octobre 2005 (39 participants).
- 1st Workshop on the certification of termination proofs, Nancy, 11-12 mai 2007 (8 participants).

4.2 Co-organisation d'écoles

- 1st Asian-Pacific Summer School on Formal Methods [4], 24-31 août 2009 (environ 55 participants).
- 2nd Asian-Pacific Summer School on Formal Methods [5], 20-27 août 2010 (environ 55 participants).

4.3 Participation à des projets

Voir `http://www-rocq.inria.fr/~blanqui/projects.html`.

- Entre 2002 et 2005, j'ai participé au projet RNTL AVERROES [6] sur l'analyse et la vérification de systèmes embarqués (voir Section 8.1).
- Entre 2003 et 2006, j'ai participé au projet MODULOGIC [7] de l'ACI Sécurité Informatique sur la conception d'une plate-forme pour développer des logiciels certifiés [DHDG04]. C'est de là qu'est issu le projet QUOTIENT que j'ai ensuite monté avec Pierre Weis pour développer le logiciel Moca.

4.4 Coordination de projets

Voir `http://www-rocq.inria.fr/~blanqui/projects.html`.

- En 2006, j'ai bénéficié d'une bourse jeune chercheur du Conseil Régional de Lorraine pour travailler sur la certification de preuves de terminaison (voir Section 8.3).
- Entre 2007 et 2008, j'ai coordonné avec Andreas Abel (LMU, Münich) un projet franco-allemand de coopération sur la terminaison financé par le Centre de Coopération Universitaire Franco-Bavarois (voir Section 7.2.3).
- Entre 2007 et 2008 également, j'ai coordonné QUOTIENT [8], une action de recherche collaborative (ARC) de l'INRIA avec le CNAM, l'ENSIIE et Paris 6 sur l'utilisation certifiée de types concrets non-libres (voir les Sections 7.3 et 8.4).
- Enfin, entre 2009 et 2011, j'ai coordonné côté français le projet franco-chinois SIVES [9] sur la simulation et la vérification de systèmes embarqués co-financé par l'ANR côté français et par la NSFC côté chinois (voir Section 8.5).

5 Domaines de recherche

Mon travail de recherche s'inscrit dans les domaines de la logique, des systèmes de types, du λ-calcul, de la théorie de la preuve, de la théorie de la réécriture et de la théorie des langages de programmation. Il s'accompagne de

4. `http://formes.asia/cms/coqschool/2009`
5. `http://formes.asia/cms/coqschool/2010`
6. `http://www-verimag.imag.fr/PROJECTS/DCS/AVERROES/`
7. `http://modulogic.inria.fr`
8. `http://quotient.loria.fr`
9. `http://formes.asia/cms/sives`

développements logiciels décrits en Section 8. Ses applications sont les outils de spécification et de preuve formelle, les langages de programmation, les assistants à la preuve, et les outils de certification de programmes et de preuves.

5.1 λ-calcul

Le λ-calcul est un formalisme introduit par Alonzo Church en 1932 pour décrire la notion de fonction de manière intensionnelle [Chu32]. Une λ-*expression* ou λ-*terme* est soit :

- une *variable* x prise dans un ensemble infini de variables $\mathcal{X}$;
- l'*abstraction* d'une λ-expression t par une variable x, notée $\lambda x.t$ (et $x \mapsto t$ en mathématiques) ;
- l'*application* d'une λ-expression t (représentant une fonction) à une autre λ-expression u, notée tu par juxtaposition de t et u (et $t(u)$ en mathématiques).

Comme en mathématiques, dans une abstraction $\lambda x.t$, la variable x choisie n'est pas significative et peut être remplacée par toute autre variable, opération appelée α-*conversion*. Ainsi, ce qu'on considère en fait, ce ne sont pas les λ-termes eux-mêmes mais bien plutôt leurs classes d'équivalence modulo α-conversion.

Lorsqu'une abstraction $\lambda x.t$ est appliquée à une expression u, on peut naturellement la simplifier en remplaçant dans t toutes les occurrences libres de x par u (en renommant les variables liées de t différemment des variables libres de u). On obtient ainsi un terme noté t_x^u. Cette opération s'appelle la β-*réduction* :

$$(\lambda x.t)u \to_\beta t_x^u$$

Ce formalisme peut paraître excessivement simple et sans surprise mais c'est en fait un formalisme extrêmement puissant puisqu'il permet de représenter toutes les fonctions calculables au sens d'Alan Mathison Turing [Kle36, Tur37] ! Et, de fait, ce formalisme est à la base des langages de programmation dits fonctionnels tels que LISP (1958), ML (1973), Scheme (1975), OCaml (1996), etc.

Plus de détails sur le λ-calcul peuvent être trouvés dans [Bar84].

5.2 Logique et systèmes de types

Le λ-calcul fournit également une manière extrêmement puissante de représenter les preuves en déduction naturelle de Gerhard Gentzen [Gen33, Pra65]. C'est ce qu'on appelle la correspondance de Curry-de Bruijn-Howard dont les différents aspects ont été progressivement mis en évidence par Haskell Curry [CF58] d'une part, puis par Nicolaas Govert de Bruijn [dB68] et William Alvin Howard [How80] à la fin des année 60 d'autre part. Dans cette correspondance, une hypothèse A est représentée par une variable x et une preuve de A obtenue par hypothèse est représentée par x :

$$\frac{}{\Gamma, x : A, \Delta \vdash x : A}$$

De plus, si b représente une preuve de B sous l'hypothèse A représentée par la variable x, alors $\lambda x.b$ représente la preuve de $A \Rightarrow B$ obtenue par la règle d'introduction de l'implication :

$$\frac{\Gamma, x : A, \Delta \vdash b : B}{\Gamma, \Delta \vdash \lambda x.b : A \Rightarrow B}$$

Maintenant, si t représente une preuve de $A \Rightarrow B$ et a représente une preuve de A, alors l'application ta représente la preuve de B obtenue par la règle d'élimination de l'implication aussi appelée *modus ponens* :

$$\frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash ta : B}$$

Vérifier la correction pour une proposition P d'une preuve représentée par un terme t revient alors à vérifier que t est bien un terme de type P, c'est-à-dire, $\vdash t : P$ où $\vdash$ est la relation définie inductivement par les règles ci-dessus.

De plus, il apparaît également que l'élimination des coupures, une importante propriété en logique [Sch77], correspond à la β-réduction :

$$\frac{\dfrac{\dfrac{\pi}{\Gamma, x : A \vdash b : B}}{\Gamma \vdash \lambda x.b : A \Rightarrow B} \quad \dfrac{\alpha}{\Gamma \vdash a : A}}{\Gamma \vdash (\lambda x.b)a : B} \qquad \frac{\pi^{\alpha}_{A}}{\Gamma \vdash b^{a}_{x} : B}$$

Donc, la terminaison de la β-réduction implique la terminaison de l'élimination des coupures.

On voit également à travers cette correspondance que la notion de type simple introduite par Alonzo Church en 1940 [Chu40] pour réparer l'incohérence du λ-calcul d'un point de vue logique [KR35] et la notion de type de données utilisée en programmation sont des cas particuliers du calcul des prédicats. Les types `int`, `float`, `int`$\rightarrow$`int` peuvent être vus comme des propositions dont les valeurs $0, 1, \ldots$ sont autant de preuves. Inversement, une formule propositionnelle $A \Rightarrow B$ peut être vue comme le type des fonctions de A dans B, A et B étant à leur tour vus comme des types.

Et cela peut se généraliser au calcul des prédicats dont les formules permettent de donner des types extrêmement précis aux programmes, voir d'en spécifier complètement le comportement. De manière générale, un programme peut avoir le type $\forall x, P(x) \Rightarrow \exists y.Q(x, y)$ signifiant que, pour toute valeur d'entrée x vérifiant la pré-condition $P(x)$, une valeur y est fournie vérifiant la post-condition $Q(x, y)$.

On voit alors que les types peuvent dépendre de valeurs. Cela conduit à considérer les types équivalents modulo l'équivalence entre les valeurs. Par exemple, $P(2+2)$ and $P(4)$ sont généralement considérés équivalents. Ainsi, toute preuve de $P(4)$ est également une preuve de $P(2+2)$.

L'équivalence (en fait, la congruence car la relation doit être stable par contexte) entre types/propositions que l'on choisie est d'une grande importance théorique et pratique. Si elle est trop petite, alors l'utilisateur doit faire beaucoup de manipulations syntaxiques et les preuves deviennent fastidieuses et hors de portée d'un non-spécialiste, ce qui ne permet pas de passer à l'échelle. Si l'équivalence est trop grosse, alors la vérification de la correction d'une preuve peut devenir indécidable [Our05]. On a donc tout intérêt à étendre cette équivalence autant que possible tout en préservant bien sûr la décidabilité de la vérification de type.

En 1998, Gilles Dowek, Thérèse Hardin et Claude Kirchner ont introduit une extension (conservatrice) de la déduction naturelle de Gentzen par une équivalence sur les propositions [DHK98, DHK03]. Gilles Dowek a ensuite étudié, avec Benjamin Werner notamment, certaines propriétés de cette *déduction (naturelle) modulo* : en particulier, l'élimination des coupures [DW98, DW03] et le fait que certaines théories axiomatiques peuvent s'exprimer comme des théories sans axiome modulo un ensemble de règles de réécriture [DHK01, Dow01b, DW05, DM07].

En 1988, Stefano Berardi [Ber88] et Jan Terlow [Ter89] indépendamment ont défini un système de types, paramétré par un ensemble de sortes et de relations entre celles-ci spécifiant quelles quantifications sont possibles, qui généralise nombre des systèmes de types introduits précédemment. Dans ces *systèmes de types purs* (PTS), l'équivalence entre types est la β-équivalence.

Un exemple important de PTS est le calcul des constructions (CC) de Thierry Coquand et Gérard Huet [Coq85, CH88]. Ce calcul contient deux sortes : $\star$ pour typer les prédicats (*e.g.* $\leq : \mathbb{N} \to \mathbb{N} \to \star$) et $\Box$ pour typer les classes de prédicats (*e.g.* $\mathbb{N} \to \mathbb{N} \to \star : \Box$). De plus, il permet toutes les quantifications possibles entre valeurs et types :

- une valeur peut dépendre d'une valeur : c'est le λ-calcul simplement typé de Church [Chu40] ;
- une valeur peut dépendre d'un type : c'est le polymorphisme ou la logique d'ordre supérieur [Gir72, Rey74] ;
- un type peut dépendre d'une valeur : ce sont les types *dépendants* [dB68] ;
- enfin, un type peut dépendre d'un type : ce sont les constructeurs de types.

CC est le système de types purs le plus simple permettant de représenter les preuves de la logique d'ordre supérieur. Mais cela va en fait plus loin car, dans ce système, il n'y a pas de distinction entre les valeurs comme les entiers, les listes, etc. et les preuves : les preuves sont des objets de première classe. Ainsi, une valeur peut-être définie à partir d'une preuve et *vice versa*. Cela permet notamment de définir des sous-ensembles et donc des fonctions "partielles", en spécifiant leur domaine de définition. Par exemple, un élément de $\{x \in \mathbb{N} \mid x$ est premier$\}$ est une paire (x, p) où x est de type $\mathbb{N}$ et p une preuve que x est premier.

Plus de détails sur le λ-calcul typé peuvent être trouvé dans [GLT88, Bar92, Kri97].

Dans la suite, nous noterons par :

- $\mathrm{FV}(t)$ l'ensemble des variables libres de t.

– $\lambda x : T.u$ ou $[x : T]u$ l'abstraction de l'expression u par la variable x de type T (comme la fonction $x \in T \mapsto u$ en mathématiques).
– $\forall x : T.U$ ou $(x : T)U$ le produit dépendant de T et de U (comme l'ensemble des fonctions $t \in T \mapsto u \in U_x^t$ en mathématiques).

Nous disons également qu'un terme est *linéaire* s'il ne contient pas deux occurrences de la même variable.

5.3 Réécriture

Nous avons vu tout l'intérêt du λ-calcul. Cependant, il ne permet pas toujours d'exprimer les structures de données et les fonctions que l'on souhaite de manière simple et efficace. De plus, toute extension de la $\beta\eta$-équivalence est nécessairement incohérente, un résultat dû à Corrado Böhm [Böh68].

La théorie de la réécriture répare ce problème en étendant le λ-calcul avec :

– des *symboles de fonction* $\mathsf{f}, \mathsf{g}, \ldots$ pris dans un ensemble $\mathcal{F}$,

et en étendant la β-réduction par un ensemble $\mathcal{R}$ de règles de réécriture définissant les symboles de $\mathcal{F}$. Une *règle de réécriture* est une paire (l, r) de λ-termes, notée $l \to r$, dont le membre gauche est de la forme $\mathsf{f} l_1 \ldots l_n$:

$$\mathsf{f} l_1 \ldots l_n \to r$$

La relation de réduction $\to_{\mathcal{R}}$ engendrée par un ensemble $\mathcal{R}$ de règles est la plus petite relation stable par contexte et substitution contenant $\mathcal{R}$. Ainsi, réécrire un terme t avec une règle $l \to r$ consiste à remplacer un sous-terme de t de la forme $l\sigma$ où σ est une substitution (fonction des variables dans les termes) par $r\sigma$.

Historiquement, la réécriture du premier ordre, *i.e.* sans abstraction ni β-réduction, a été introduite par Donald E. Knuth et Peter B. Bendix en 1967 comme outil pour décider l'égalité de deux expressions dans certaines théories équationnelles [KB70]. Mais elle permet aussi de représenter toute fonction calculable au sens de Turing.

Pour la réécriture d'ordre supérieur, plusieurs approches existent. La première est celle des *systèmes de réduction combinatoires* (CRS) de Jan Willem Klop introduite en 1980 [Klo80]. Une autre est celle des *systèmes de réduction d'expressions* (ERS) de Zurab Khasidashvili introduite en 1990 [Kha90, KvO95, GKK05]. Enfin, en 1991, Tobias Nipkow a introduit ses *systèmes de réécriture d'ordre supérieur* (HRS) [Nip91, MN98]. Cette dernière s'applique aux λ-termes en forme β-normale et η-longue et utilise le filtrage modulo $\beta\eta$ [Dow01a].

Le filtrage modulo $\beta\eta$ a été montré décidable par Colin Stirling [Sti06, Sti09] en 2006 seulement répondant ainsi positivement à une conjecture de Gérard Huet de 1976 qui avait montré que l'unification modulo $\beta\eta$ était indécidable [Hue76], et alors que le filtrage modulo β seulement a été montré indécidable par Ralph Lauder en 2003 [Loa03]. Mais sa complexité est non élémentaire [Sta79, Wie99]. Cependant, lorsqu'on restreint les membres gauches de règles aux motifs introduits par Dale Miller [Mil89], qui correspondent d'ailleurs exactement aux

conditions de Klop sur les CRS, le filtrage (resp. l'unification) devient similaire au filtrage (resp. l'unification) du premier ordre modulo α-conversion.

Ainsi, dans le cas du λ-calcul simplement typé, CRS et HRS sont assez proches. Voir [KvOvR93, Bla00] pour une comparaison entre les deux approches. D'ailleurs, dans [vO94, vR96], Vincent van Oostrom and Femke van Raamsdonk développent un cadre général pour la réécriture d'ordre supérieur, les HORS, paramétré par un "calcul de substitution" et montrent que de nombreux systèmes comme les CRS et les HRS correspondent à des calculs de substitution particuliers. Cependant, du fait que les HRS ne s'appliquent qu'à des λ-termes simplement typés en forme β-normale η-longue, il semble difficile de les étendre à des systèmes de type plus complexes dans lesquels la terminaison de la β-réduction n'est pas garantie *a priori* (c'est par exemple le cas avec de la réécriture au niveau des types) ou dans lesquels la définition ou l'existence d'une notion de forme η-longue est problématique [Bar98a, Bar99b].

Plus de détails sur la théorie de la réécriture peuvent être trouvés dans [DJ90, BN98, Ohl02] pour la réécriture du premier ordre, et dans [TeR03] pour la réécriture d'ordre supérieur.

De manière générale, étant donné une relation R, nous disons :

- t se *réécrit* en u si tRu,
- t est en *forme normale* s'il n'existe pas d'élément u tel que t puisse se réécrire en u,
- R *normalise* si tout élément admet une forme normale,
- R *termine* s'il n'existe pas de séquence infinie de réécriture $t_1Rt_2R\ldots$,
- R est *finiment branchante* si aucun élément ne peut se réécrire en un nombre infini d'éléments,
- R *conflue* si l'ordre des réécritures n'est pas important : $\overline{R}^*R^* \subseteq R^*\overline{R}^*$, où $\overline{R}$ est l'*inverse* de R ($t\overline{R}u$ si uRt), R^* est la clôture réflexive et transitive de R, et la composition des relations est notée par juxtaposition ($tRSv$ s'il existe u tel que tRu et uSv).

6 Travail de thèse

Afin de mieux apprécié le chemin parcouru depuis mon travail de thèse, je rappelle ci-après quelles en étaient les motivations et quels en sont les principaux résultats.

Nous avons vu l'importance d'avoir une équivalence au niveau des types aussi grosse que possible tout en préservant la décidabilité de la vérification de type. Nous avons aussi vu l'intérêt de la réécriture pour définir des fonctions et des prédicats plus simplement, et pour montrer l'égalité de deux expressions dans certaines théories.

Je me suis donc intéressé à étendre l'équivalence des systèmes de types purs avec de la réécriture tout en préservant la décidabilité de la vérification de type. Or, pour cela, il suffit que la réécriture termine (*i.e.* il n'y a pas de séquence infinie de réécriture) et conflue (*i.e.* l'ordre des étapes de réécriture n'a pas d'importance).

Les principaux résultats que j'ai obtenus, qui généralisent nombre de travaux antérieurs sur ce sujet dont notamment ceux de Val Breazu-Tannen [BTG91], Jean-Pierre Jouannaud et Mitsuhiro Okada [JO97a], Gilles Barthe [BG95b], Franco Barbanera, Maribel Fernández et Herman Geuvers [BFG97], sont :

1. L'étude méta-théorique des systèmes de types obtenus en remplaçant dans les systèmes de types purs (PTS) [Bar92] la β-équivalence par différents types de relations plus ou moins abstraites : systèmes de types modulo, systèmes de type à réduction, systèmes de types algébriques. En particulier, dans les systèmes de types algébriques, l'équivalence au niveau des types inclut la β-réduction ainsi qu'un ensemble arbitraire de règles de réécriture d'ordre supérieur (*i.e.* autorisant la manipulation de variables fonctionnelles) permettant de définir fonctions et prédicats.
2. Un critère de terminaison pour le calcul des constructions algébriques (CAC), cas particulier important de système de types algébriques car il inclut à la fois les types dépendants et le polymorphisme.
3. L'application de ce critère à deux autres systèmes logiques :
   - Si un ensemble de règles de réécriture satisfait le critère de terminaison, alors l'élimination des coupures en *déduction naturelle modulo* cet ensemble de règles termine, la déduction naturelle modulo étant une extension conservatrice de la déduction naturelle de Gentzen introduite en 1998 par Gilles Dowek, Thérèse Hardin et Claude Kirchner [DHK98, DHK03, DW98, DW03], où les règles de la déduction naturelle s'applique modulo une relation d'équivalence sur les propositions.
   - Le critère de terminaison implique la terminaison d'un sous-ensemble important du calcul des constructions inductives de Christine Paulin et Benjamin Werner [PM93, Wer94], qui est le calcul à la base de l'assistant à la preuve Coq [Coq10].

La première partie de la thèse n'a pas encore donné lieu à publication. Quand au travail de recherche sur le critère de terminaison, il a donné lieu pendant la thèse aux publications suivantes :

[BJO99] Première version du critère pour le calcul des constructions (avec une preuve incorrecte qui sera ensuite corrigée).

[BJO02] Preuve détaillée (et correcte cette fois-ci) dans le cas du λ-calcul simplement typé.

[Bla01a] Extension de [BJO99] à la réécriture au niveau type, *i.e.* où les types eux-mêmes peuvent être définis par des règles de réécriture.

[Bla00] Extension à la réécriture avec filtrage modulo $\alpha\beta\eta$-conversion et application aux systèmes de réécriture combinatoires de Jan Willem Klop (CRS) [Klo80] et aux systèmes de réécriture d'ordre supérieur de Tobias Nipkow (HRS) [Nip91].

7 Travaux théoriques

Depuis lors, mon travail de recherche s'est articulé autour de trois axes interdépendants :

1. L'étude méta-théorique du calcul des constructions modulo.
2. L'étude de critères de terminaison pour des fonctions définies par des règles de réécriture d'ordre supérieur dans le calcul des constructions algébriques ou l'un de ses sous-ensembles (le λ-calcul simplement typé en particulier).
3. La certification automatique de preuves de terminaison pour des systèmes de réécriture du premier ordre.

Chacun de ces axes de recherches a également donné lieu à des développements logiciels décrits dans la section suivante. Le troisième axe étant essentiellement du développement, il est décrit dans la section suivante, tandis que les deux premiers axes sont décrits ci-après.

7.1 Méta-théorie du calcul des constructions modulo

7.1.1 Réduction, typage et cohérence logique

Journaux : [Bla05b]

Dans [Bla05b], j'ai simplifié et complété mon travail de thèse dans plusieurs directions :

1. **Simplification de la preuve de correction du critère de terminaison.**
   Dans ma thèse, la preuve de correction du critère de terminaison suit celle du calcul des constructions de Thierry Coquand et Jean Gallier basée sur l'utilisation de modèles *à la* Kripke [CG90]. Elle considère ainsi des ensembles de paires (Γ, t) où Γ est un environnement de typage (séquence ordonnée de paires (x, T) où x est une variable et T un type) et t un terme typable dans l'environnement Γ.

   Dans [Bla05b], je donne une preuve simplifiée de la correction du critère en suivant la preuve de terminaison du calcul des constructions de Herman Geuvers [Geu94]. Celle-ci ne considère pas d'environnements de typage et donc des termes potentiellement mal typés voir non typables. Cela permet d'alléger les notations, d'éliminer la nécessité de montrer certaines propriétés liées à l'utilisation d'environnements de typage (monotonie par rapport à Γ), et ainsi de diviser la taille de la preuve par deux.
2. **Conditions de cohérence logique.**
   Je donne aussi des conditions générales suffisantes pour garantir la cohérence logique (Théorème 17). Je montre en effet qu'il ne peut y avoir de preuve de $\bot = (A : \star)A$ (terme signifiant que toute proposition A est prouvable) dans l'environnement vide si, pour tout symbole de fonction f, quelle que soit les règles de réécriture définissant f :
   – soit le type de f est de la forme $(\vec{x} : \vec{T})\mathsf{C}\vec{v}$ où C est une constante de type,

– soit de la forme $(\vec{x} : \vec{T})T_i$ (f est alors une espèce de projection),

Je montre également qu'il ne peut y avoir de preuve de $\bot$ en forme normal si, grosso modo, f est complètement défini, c'est-à-dire, si f est de type $(x_1 : T_1) \dots (x_n : T_n)U$, $x_n \notin \mathrm{FV}(U)$ et, pour toute substitution γ telle que, pour tout i, $x_i\gamma$ est en forme normal et $A : \star \vdash x_i\gamma : T_i\gamma$, alors $f\vec{x}\gamma$ est réductible. Ce problème sera par la suite étudié beaucoup plus en détail par Daria Walukiewicz et Jacek Chrząszcz dans [WCC08].

3. **Conditions pour que la β-réduction préserve le typage.**
Une condition essentielle à toutes les propriétés importantes du calcul (décidabilité du typage, terminaison, cohérence) est que la β-réduction et la réécriture préservent le typage. Cette propriété élémentaire est généralement facile à vérifier dans les systèmes de types sans relation de conversion au niveau des types ou dont la relation de conversion est transitive. Quand cette relation de conversion est la joinabilité pour une certaine relation de réduction (*i.e.* le fait d'avoir un réduit commun), la transitivité se réduit à la confluence de cette relation, une propriété satisfaite par tout système de réécriture orthogonal [Ros73, Klo80], ce qui est en particulier le cas pour la β-réduction. Mais que faire dans le cas où on se sait pas établir la confluence *a priori* ?

   Dans [BFG97], Franco Barbanera, Maribel Fernández et Herman Geuvers montrent que la β-réduction et la réécriture, avec des symboles de fonction simplement typés seulement, préservent le typage s'il n'y a pas de réécriture au niveau type et si, pour chaque règle de réécriture $l \to r$, il existe un environnement de typage Γ pour les variables libres de l et un type T tels que $\Gamma \vdash l : T$ and $\Gamma \vdash r : T$.

   Dans [Bla05b] (Théorème 16), je simplifie et étend leur preuve aux symboles de fonctions typés dans CAC lui-même et à la réécriture au niveau type. Je montre en particulier que la β-réduction préserve le typage si aucun membre droit de règle de réécriture contient $\Box$ ou un terme de la forme $(\vec{x} : \vec{T})\star$, et si chaque membre droit de règle de réécriture au niveau type est de la forme $\mathsf{f}\vec{t}$. Le problème des règles de réécriture au niveau type dont le membre droit est de la forme x, $(x : T)U$ ou $[x : T]U$ reste ouvert.

   Je donne également des conditions générales pour assurer la préservation du typage par réécriture (Théorème 5). En particulier, je remarque qu'il n'est pas nécessaire que le membre gauche d'une règle $l \to r$ soit typable (condition $\Gamma \vdash l : T$ dans [BFG97]). Ce qui est important est que, si $l\theta$ est une instance typable de l, alors $r\theta$ est également typable avec le même type dans le même environnement. Cette simple remarque a des conséquences importantes.

   En effet, avec des types polymorphes ou dépendants, imposer que le membre gauche d'une règle soit typable impose des contraintes telles que le membre gauche d'une règle aura toutes les chances d'être non linéaire, *i.e.* contenir plusieurs occurrences de la même variable. Par exemple, considérons la concaténation de listes polymorphes $\mathsf{app} : (A : \star)\mathsf{list}A \to \mathsf{list}A \to \mathsf{list}A$ définie par les règles suivantes :

$$\begin{array}{rcl} \mathsf{app}\, A\, (\mathsf{nil}\, A')\, l' & \to & l' \\ \mathsf{app}\, A\, (\mathsf{cons}\, A'\, x\, l)\, l' & \to & \mathsf{cons}\, A\, x\, (\mathsf{app}\, A\, l\, l') \end{array}$$

où $\mathsf{nil} : (A : \star)\mathsf{list}A$ est la liste vide et $\mathsf{cons} : (A : \star)A \to \mathsf{list}A \to \mathsf{list}A$ permet de rajouter un élément en tête d'une liste. Pour que les membres gauches soient typables, il faudrait prendre $A' = A$. Cependant, si une instance d'un membre gauche est typable, alors A' est nécessairement équivalent à A. On peut donc prendre $A' \neq A$ sans danger.

Avoir des membres gauches de règles linéaires permet de réécrire les termes beaucoup plus efficacement car nous n'avons alors pas à tester l'égalité des sous-termes que le filtrage associe à chaque occurrence d'une variable.

De plus, étant donné que la confluence est modulaire pour les systèmes de réécriture dont les membres gauches sont linéaires, *i.e.* est une propriété préservée par union disjointe [Mül92, vO94], cela simplifie grandement la preuve de confluence et donc de préservation du typage.

7.1.2 Types inductifs

Journaux : [Bla05c]
Conférences : [Bla03a]

Dans mon travail de thèse, j'avais montré que le calcul des constructions inductives pouvaient être aisément traduit dans le calcul des constructions algébriques. Cependant, les règles définissant les récurseurs ou principes d'induction ne satisfaisaient pas toutes le critère de terminaison. Dans [Bla03a, Bla05c], je montre qu'il y a essentiellement deux classes de types posant un problème :

- **Les "gros" types inductifs.** Un type inductif est *gros* s'il a un constructeur prenant en argument un type ou un prédicat qui n'est pas un paramètre du type. C'est le cas par exemple d'un prédicat disant qu'un ensemble d'éléments de type A, représenté par un terme de type $A \to \star$, est fini (exemple dû à Christine Paulin). Avec certains gros types inductifs, l'élimination forte, *i.e.* la définition de prédicats par induction sur un tel type, permet de construire des termes non terminants et conduit à des incohérences [Coq86]. Par contre, l'élimination faible, *i.e.* la définition d'objets par induction, ne devrait pas poser de problème.
- **Les types inductifs "non linéaires".** Un type inductif est *non linéaire* s'il prend plusieurs types ou prédicats en arguments et a un constructeur dans le type duquel plusieurs de ces types sont égaux. C'est le cas par exemple de la définition inductive de l'égalité proposée par Conor McBride (appelée "égalité de John Major" par cet auteur) [McB99].

Dans ma thèse ou dans [Bla05b], pour montrer la correction du critère de terminaison, j'utilise une interprétation des types "basée sur l'introduction" selon la terminologie de Ralph Matthes [Mat98], *i.e.* ne tenant compte que des constructeurs. Mais il est aussi possible d'utiliser une interprétation duale "basée sur l'élimination", *i.e.* ne tenant compte que des destructeurs. Les deux approches ne sont pas équivalentes et conduisent à des conditions de terminaison différentes. En particulier, comme je l'ai montré dans ma thèse, l'interprétation des types basée sur l'introduction se généralise aisément à la réécriture, y compris avec des constructeurs non libres.

Dans [Bla03a, Bla05c], je montre que l'interprétation basée sur l'élimination permet de montrer la terminaison de l'élimination faible sur les types inductifs gros ou non linéaires. Par contre, je ne sais pas encore comment traiter l'élimination forte sur les types inductifs non-linéaires. Heureusement, l'intérêt pratique d'une telle élimination forte semble limité.

Je montre également que le critère de terminaison et sa preuve de correction peuvent être aisément étendus aux classes de types inductifs suivantes qui ne sont pas prises en compte dans [PM93, Wer94] et ne sont pas autorisées dans le système Coq :

- **Types positifs non stricts.** La positivité est une condition syntaxique simple permettant de garantir que l'ensemble des valeurs de ce type est non vide et aussi que la définition de fonctions par induction sur ce type préserve la terminaison comme l'a montré Paul Francis Mendler [Men87].

  Pour expliquer cette condition, je vais me restreindre au λ-calcul simplement typé. Dans ce cas, un type inductif I défini par des constructeurs $\mathsf{c}_1 : T_{1,1} \to \ldots \to T_{1,n_1} \to I, \ldots, \mathsf{c}_m : T_{m,1} \to \ldots \to T_{m,n_m} \to I$ est *positif* si I n'apparaît jamais à gauche d'un nombre impair de $\to$ dans chacun des $T_{i,j}$. Il est *strictement* positif si, pour tout i et j, $T_{i,j}$ ne contient pas I ou est de la forme $U_1 \to \ldots \to U_p \to I$ avec I n'apparaissant dans aucun U_k.

  Par exemple, le type $\mathbb{N}$ des entiers naturels unaires peut être défini par les constructeurs $\mathsf{0} : \mathbb{N}$ et $\mathsf{s} : \mathbb{N} \to \mathbb{N}$. C'est le plus petit ensemble tel que $\mathsf{0} \in \mathbb{N}$ et, pour tout n, $n \in \mathbb{N} \Rightarrow \mathsf{s}n \in \mathbb{N}$. Dit autrement, $\mathbb{N}$ est le plus petit point fixe de la fonction $S \mapsto S \cup \{\mathsf{0}\} \cup \{\mathsf{s}n \mid n \in S\}$. La positivité de $\mathbb{N}$ assure que cette fonction est monotone et donc admet un plus petit point fixe [Tar55].

  Le critère de terminaison donné dans ma thèse ne permet pas de montrer la terminaison de fonctions définies par induction sur des types positifs non stricts. Dans [Bla03a, Bla05c], je montre que l'approche basée sur l'élimination permet de montrer la terminaison de telles fonctions.
- **Types inductif-récursifs.** Il y a de nombreux exemples où il est utile de définir simultanément un type inductif et une fonction définie par induction sur ce même type comme l'a montré Peter Dybjer [Dyb00]. Il en va ainsi du type des listes ne contenant pas deux fois le même élément. Pour rajouter un élément à une telle liste, il est en effet nécessaire de montrer que cet élément n'apparaît pas dans la liste. Dans [Bla03a, Bla05c], j'étends la définition de positivité pour pouvoir traiter de tels types.

7.1.3 Confluence de la réécriture conditionnelle d'ordre supérieur

Journaux : [BKR10]
Conférences : [BKR06]

Ce travail a été réalisé en collaboration avec Colin Riba et Claude Kirchner. Il constitue la première partie de la thèse de Colin Riba [Rib07a] dont j'ai coencadré le travail avec Claude Kirchner.

Suite à mon premier travail sur l'utilisation d'annotations de taille pour montrer la terminaison de fonctions [Bla04d] (voir Section 7.2.3), il m'est apparu nécessaire de chercher à étendre mes travaux sur la terminaison de la réécriture non conditionnelle à la réécriture conditionnelle, afin de pouvoir simuler fidèlement la construction `if then else` des langages de programmation où l'ordre d'évaluation des arguments est important. Dans l'étude de la terminaison, il est alors essentiel de prendre en compte la valeur de la condition.

Une règle de réécriture conditionnelle est une expression de la forme [DOS88] :

$$l \to r \Leftarrow l_1 = r_1 \wedge \ldots \wedge l_n = r_n$$

Réécrire un terme t avec une telle règle consiste, comme dans le cas non conditionnel, à remplacer un sous-terme de t de la forme $l\sigma$ où σ est une substitution par $r\sigma$. La différence est que ce remplacement ne peut maintenant se faire que si, pour tout i, $l_i\sigma \downarrow_{\mathcal{R}} r_i\sigma$, où $t \downarrow_{\mathcal{R}} u$ si t et u ont un réduit commun par $\to_{\mathcal{R}}$ (relation de joinabilité). Ainsi, la relation de réécriture engendrée par un ensemble de règles de réécriture conditionnelle est un point fixe.

En fait, il est possible de définir différentes relations de réécriture selon la façon dont on interprète $=$. On obtient ainsi par exemple [DOS88] :

- la réécriture conditionnelle *standard* si $=$ est interprété par $\downarrow_{\mathcal{R}}$;
- la réécriture conditionnelle *semi-équationnelle* ou *naturelle* si $=$ est interprété par $\leftrightarrow^*_{\mathcal{R}}$.

Et ces différentes relations n'ont pas les mêmes propriétés. Par exemple, si la relation de réécriture conditionnelle standard conflue, il en va de même de la relation conditionnelle semi-équationnelle. Par contre, la réciproque est fausse en général.

Un cas particulier important de réécriture conditionnelle standard est quand les termes $r_1, \ldots, r_n$ sont des termes clos (*i.e.* sans variable libre) en forme normale pour $\to_{\mathcal{R}}$, auquel cas tester une condition $l_i = r_i$ revient à vérifier si la forme normale de $l_i\sigma$ est r_i. Cependant, avec la réécriture conditionnelle, savoir si un terme est réductible ou en forme normale est généralement indécidable.

On voit également qu'avec la β-rédaction, deux relations de réécriture conditionnelle peuvent être définie selon que l'on autorise la β-réduction dans l'évaluation des conditions ou pas :

- réécriture $\mathcal{R}$-conditionnelle, notée $\to_{\mathcal{R}}$, si $=$ est interprété par $\downarrow_{\mathcal{R}}$,
- réécriture $\beta \cup \mathcal{R}$-conditionnelle, notée $\to_{\mathcal{R}(\beta)}$, si $=$ est interprété par $\downarrow_{\beta\cup\mathcal{R}}$.

Outre que la confluence de la β-réduction et de la réécriture est une propriété éminemment désirable en soi, car elle exprime le déterminisme des calculs, elle permet aussi de montrer la terminaison du calcul des constructions algébriques quand il y a des types ou des prédicats définis par réécriture, comme expliqué en Section 7.1.1. Or, il se trouve que cette propriété n'avait fait l'objet que de quelques travaux :

- Dans [Tak93], Masako Takahashi considère une forme très générale de réécriture conditionnelle où les conditions sont des prédicats quelconques. Il montre dans ce cas qu'il y a confluence si les conditions sont préservées par réduction. Or, si la condition consiste à vérifier si deux termes ont un

réduit commun, vérifier que cette condition est stable par réduction revient à montrer la confluence elle-même. Ce résultat n'est donc pas applicable au type de réécriture conditionnelle que nous considérons.

- Dans [ALS94], Jürgen Avenhaus et Carlos Loría-Sáenz considère de la réécriture d'ordre supérieur *à la* Nipkow qui est une forme particulière de réécriture d'ordre supérieur où une β-normalisation est réalisée avant et après chaque réécriture. Ce type de réécriture ne peut donc fonctionner que dans un système où la β-réduction termine *a priori*, ce qui n'est pas le cas dans le calcul des constructions algébriques avec de la réécriture au niveau des types par exemple.

Il était donc nécessaire d'entreprendre une étude détaillée de la confluence de la réécriture conditionnelle d'ordre supérieur combinée avec la β-réduction dans un cadre non typé de façon à obtenir des résultats qui puissent s'appliquer à n'importe quel système de types. Pour cela, nous nous sommes inspirés des travaux les plus avancés sur la combinaison de la réécriture non conditionnelle et de la β-réduction :

- Dans [Mül92], Fritz Müller montre que, pour tout système de réécriture dont les membres gauches de règles sont algébriques (*i.e.* sans abstraction ni variable appliquée) et linéaires, et les membres droits applicatifs (*i.e.* sans abstraction), $\rightarrow_\beta \cup \rightarrow_\mathcal{R}$ conflue sur l'ensemble des λ-termes si $\rightarrow_\mathcal{R}$ conflue sur l'ensemble des λ-termes applicatifs. Ce résultat sera généralisé à la combinaison de deux systèmes d'ordre supérieur linéaires à gauche quelconques par Vincent van Oostrom [vO94].
- Quand les membres droits des règles sont également algébriques (*i.e.* $\mathcal{R}$ est un système de réécriture du premier ordre), Dan Dougherty montre dans [Dou92] que la condition de linéarité n'est pas nécessaire si l'on se restreint à un ensemble *stable* de termes β-terminants, *i.e.* un ensemble clos par réduction et sous-terme et dont aucun élément ne contient un sous-terme de la forme $\mathsf{f}\vec{t}$ avec $\mid \vec{t} \mid > \alpha_\mathsf{f}$ où α_f est l'arité de f (forme faible de typage). Cela généralise les travaux antérieurs de Val Breazu-Tannen et Jean Gallier sur la combinaison de la réécriture du premier ordre et de la β-réduction dans certains λ-calculs typés [BT88, BTG94].

Résultats de modularité. Nous avons généralisé ces deux résultats de la manière suivante :

- [Mül92] : Nous montrons que la réécriture $\mathcal{R}$-conditionnelle combinée avec β conflue si la réécriture $\mathcal{R}$-conditionnelle conflue et si $\mathcal{R}$ est un ensemble de règles $l \rightarrow r \Leftarrow l_1 = r_1 \wedge \ldots \wedge l_n = r_n$ telles que l est algébrique et linéaire, r est applicatif et, pour tout i, r_i est applicatif et clos.
- [Dou92] : Nous montrons que la réécriture $\mathcal{R}$-conditionnelle combinée avec β conflue sur l'ensemble des termes ayant une forme β-normale respectant l'arité des symboles si la réécriture $\mathcal{R}$-conditionnelle conflue et si $\mathcal{R}$ est un ensemble de règles dont tous les éléments sont algébriques.

Systèmes orthonormaux. Nous nous sommes également intéressés au cas où la confluence de β et de la réécriture conditionnelle ne peut être déduite de celle de la réécriture. Dans le cas non-conditionnel, Barry K. Rosen a montré

que tout système du premier ordre *orthogonal*, *i.e.* linéaire gauche et sans paire critique, confluе [Ros73]. Ce résultat a ensuite été généralisé aux systèmes parallèlement clos par Gérard Huet [Hue80], aux systèmes de réduction combinatoires (CRS) par Jan Willem Klop [Klo80] et enfin aux systèmes de réécriture d'ordre supérieur (HORS) parallèlement clos par Vincent van Oostrom [vO97].

Qu'en est-il dans le cas conditionnel ? Dans ce cas, une paire critique est une équation conditionnelle obtenue par la superposition de deux règles et la conjonction de leurs conditions. Alors, un système est orthogonal s'il est linéaire gauche et aucune de ses paires critiques n'est *faisable*, *i.e.* la conjonction des conditions est insatisfaisable. Jan Bergstra et Jan Willem Klop ont alors montré que tout système semi-équationnelle ou normal qui est orthogonal confluе, mais que ce n'était pas toujours le cas des systèmes standards [BK86]. De plus, vérifier l'insatisfaisabilité des paires critiques paraît aussi difficile que de montrer la confluence.

Nous avons alors introduit une sous-classe décidable des systèmes normaux orthogonaux : les systèmes *orthonormaux*. Un système est orthonormal s'il est linéaire gauche ; pour chaque règle $l \to r \Leftarrow l_1 = r_1 \wedge \ldots \wedge l_n = r_n$, les termes r_i sont clos, en forme β-normale et ne contiennent aucun symbole défini ; et pour chaque paire critique $l = r \Leftarrow l_1 = r_1 \wedge \ldots \wedge l_n = r_n$, il existe i et j tels que $l_i = l_j$ et $r_i \neq r_j$.

7.1.4 Confluence de la réécriture d'ordre supérieur modulo

Conférences : [Bla03b]

Un autre type de réécriture est également extrêmement utile : la réécriture modulo certaines théories équationnelles non orientables telles que l'associativité et la commutativité, comme c'est le cas de l'addition et de la multiplication sur les entiers par exemple [LB77].

D'un point de vue pratique, cela permet de rendre implicite par le système l'utilisation de tactique comme `ring` dans Coq, qui permet de montrer l'égalité d'expressions polynômiales dans un anneau commutatif.

Le problème de la confluence et de la terminaison d'une relation modulo une autre relation a fait l'objet de nombreuses études dont les plus importantes sont celle de Gérard Huet en 1980 [Hue80] et celle de Jean-Pierre Jouannaud et Hélène Kirchner en 1986 [JK86].

Cependant, aucune étude ne concernait la confluence de la β-réduction combinée à de la réécriture modulo un ensemble d'équations $\mathcal{E}$. C'est ce que j'ai entrepris dans [Bla03b] (cet article sera également discuté dans la Section 7.2.2 dédiée à la terminaison de la réécriture modulo $\mathcal{E}$).

Pour $\mathcal{E}$, je considère un ensemble symétrique d'équations $l = r$ (*i.e.* $l = r \in \mathcal{E}$ ssi $r = l \in \mathcal{E}$) telles que l et r sont des termes algébriques (*i.e.* sans abstraction ni variable appliquée) de la forme $\mathtt{f}\vec{t}$ (*i.e.* non réduits à une variable) ayant les mêmes variables exactement (*i.e.* $\mathrm{FV}(l) = \mathrm{FV}(r)$). Cela inclut des équations comme $x+y = y+x$ (commutativité), $x+(y+z) = (x+y)+z$ (associativité) ou $(x \times (y+z) = (x \times y)+(x \times z)$ (distributivité), mais exclut des équations comme

$x + 0 = x$ (élément neutre), $x \times 0 = x$ (élément absorbant) ou $x + (-x) = 0$ (inverse), ce qui n'est pas très grave puisque, étant orientables, elles peuvent être incluses dans l'ensemble des règles $\mathcal{R}$.

Je note par $\sim$ la relation d'équivalence engendrée par $\mathcal{E}$, et je dis qu'une relation R :

- *confluе modulo* $\sim$ *sur les classes d'équivalence de* $\sim$ si $\overline{R}^* \sim R^* \subseteq R^* \sim \overline{R}^*$,
- *confluе localement modulo* $\sim$ si $\overline{R}R \subseteq R^* \sim \overline{R}^*$,
- *est localement cohérente modulo* $\sim$ si $\leftarrow_{\mathcal{E}} R \subseteq R^* \sim \overline{R}^*$ [JK86].

Alors je montre les résultats suivants :

- Théorème 11 : La vérification de type modulo $\rightarrow_\beta \cup \rightarrow_{\mathcal{R}} \cup \rightarrow_{\mathcal{E}}$ est décidable si $\rightarrow_\beta \cup \sim \rightarrow_{\mathcal{R}}$ normalise et conflue sur les classes d'équivalence de $\sim$, $\rightarrow_{\mathcal{R}}$ est finiment branchante, $\mathcal{E}$ est linéaire et $\sim$ est décidable.
- Théorème 14 : $\rightarrow_\beta \cup \sim \rightarrow_{\mathcal{R}}$ conflue modulo $\sim$ sur les classes d'équivalence de $\sim$ si $\rightarrow_\beta \cup \sim \rightarrow_{\mathcal{R}}$ termine, $\mathcal{E}$ est linéaire et $\rightarrow_{\mathcal{R}}$ est localement confluente et cohérente modulo $\sim$.
- Théorème 15 : $\rightarrow_\beta \cup \sim \rightarrow_{\mathcal{R}}$ conflue modulo $\sim$ sur les classes d'équivalence de $\sim$ si $\mathcal{E}$ est linéaire, $\mathcal{R}$ est linéaire gauche et $\rightarrow_{\mathcal{R}}$ conflue modulo $\sim$ sur les classes d'équivalences de $\sim$.

7.1.5 Calcul des constructions modulo théories

Conférences : [BJS07, BJS08]

Ce travail a été réalisé en collaboration avec Pierre-Yves Strub et Jean-Pierre Jouannaud. Il constitue la thèse de Pierre-Yves Strub [Str03, Str08] dont j'ai coencadré le travail avec Jean-Pierre Jouannaud.

Nous avons vu en introduction tout l'intérêt qu'il y a dans un système logique à considérer une équivalence entre les propositions. Dans ma thèse et dans les travaux précédemment décrits, j'ai étudié le cas où cette équivalence est engendrée à partir d'une théorie équationnelle donnée, la réécriture fournissant une procédure particulière de décision de cette théorie.

Mais on peut envisager d'aller plus loin en permettant à cette équivalence d'être dynamique et de dépendre, au cours d'une preuve, des hypothèses de l'environnement de typage courant qui peuvent être introduites par l'utilisateur par la règle d'introduction de l'implication ou de la quantification universelle. Se pose alors le problème de savoir quelles hypothèses peuvent être prises en compte et comment une telle équivalence (combinée avec β ou de la réécriture) peut être décidée.

Dans [PS94], Erik Poll et Paula Severi considère une extension des systèmes de types purs où l'équivalence sur les types est augmentée des hypothèses de la forme $x = t$ où x est une variable et t un terme quelconque (ne contenant pas x par construction).

Dans [Ste02, Ste05a, Ste05b], Mark-Oliver Stehr introduit le "calcul des constructions ouvert" combinant de manière consistante systèmes de types purs fonctionnels, logique équationnelle avec prédicats d'appartenance [BJM00] et

logique de réécriture (conditionnelle). Il introduit notamment trois modalités différentes pour les types/propositions dont l'égalité : les propositions vraies structurellement du fait de leur représentation (cela rejoint nos travaux sur les types quotients en Section 7.3), les propositions vraies calculatoirement et les propositions vraies par déduction/inférence à la Prolog ("assertional"). Il fournit une sémantique ensembliste booléenne montrant la consistance de ce système, travail rejoignant celui d'Alexandre Miquel et Benjamin Werner pour le calcul des constructions [MW02]. Cependant, sans restriction supplémentaire, un tel calcul ne peut pas être décidable.

Dans [Str03], nous considérons une extension du calcul des constructions où l'équivalence sur les types est augmentée des hypothèses équationnelles du premier ordre non quantifiées, *i.e.* de toutes les hypothèses de la forme $t = u$ où t et u sont des termes du premier ordre. Mais seules quelques propriétés métathéoriques sont étudiées. Pierre-Yves a implanté ce calcul en Maude [CDE+05] en réutilisant des développements de Mark-Oliver Stehr et José Meseguer [SM99].

Dans un environnement de typage donné, les variables sont comme des constantes et donc les équations comme des équations closes. Or, toute théorie équationnelle close est décidable [Ack54]. Une procédure de décision utilisant la réécriture peut être obtenue en *complétant* l'ensemble d'équations par la procédure de Donald E. Knuth et Peter B. Bendix [KB70] en orientant les équations par un ordre bien fondé *total sur les termes clos* quelconque [Lan75]. Mais l'approche la plus efficace utilise la notion de clôture de congruence [NO77, NO80] dont le meilleur algorithme est due à Peter J. Downey, Ravi Sethi et Robert Endre Tarjan [DST80].

Dans [BJS07], nous considérons le calcul des constructions étendu avec les entiers naturels et le principe d'induction correspondant modulo, non seulement les hypothèses équationnelles, mais également modulo une théorie équationnelle décidable. Ainsi, dans un environnement de typage Γ, on inclut dans l'équivalence toute conséquence équationnelle de $\mathbb{T} \cup \mathcal{E}(\Gamma)$ où $\mathbb{T}$ est une théorie équationnelle décidable (*e.g.* l'arithmétique linéaire) et $\mathcal{E}(\Gamma)$ l'ensemble des hypothèses équationnelles de Γ.

Pour ce calcul, nous montrons que la β-réduction est compatible avec le typage à condition que, lorsqu'on fait une coupure sur une hypothèse équationnelle $u = v$ dans un environnement Γ, l'équation $u = v$ soit *équationnellement* déductible de $\mathbb{T} \cup \mathcal{E}(\Gamma)$. Autrement dit, lorsqu'on applique une fonction de type $\forall x : u = v.P$ à un terme p et que l'hypothèse $u = v$ est utilisée dans le typage de t, alors p doit être une preuve par réflexivité. Il est possible d'implanter cela en utilisant par exemple le système des définitions opaques/transparentes de Coq.

Nous montrons également que le calcul est consistant en réutilisant une technique due à Gilles Barthe [Bar98b]. Enfin, pour la décidabilité, nous proposons d'utiliser un algorithme inspiré des techniques de combinaison de théories équationnelles [SS89, BS92].

Dans sa thèse [Str08], Pierre-Yves Strub considère le cas du calcul des constructions inductives avec élimination forte, à la base du système Coq, et étudie en détail les propriétés de ce système. L'article [BJS08] est issu de son travail de thèse.

Par la suite, dans [Str10b], Pierre-Yves Strub simplifie et généralise encore un peu plus ce calcul et en prouve formellement les propriétés en Coq en partant de la formalisation du calcul des constructions inductives de Bruno Barras [Bar99a]. De plus, il a développé une extension de Coq basé sur ce travail et dans laquelle il est possible d'utiliser des procédures de décision externes [Str10a].

7.2 Terminaison de la réécriture d'ordre supérieur

7.2.1 Clôture de calculabilité

Conférences : [Bla07]

Tous mes travaux sur la terminaison s'appuient sur une notion essentielle, la clôture de calculabilité (ou réductibilité), dont je retrace l'historique et montre l'application à la réécriture (avec filtrage d'ordre supérieur) et à la preuve de bonne fondation de l'ordre récursif sur les chemins dans [Bla07].

Cette notion a été introduite par Jean-Pierre Jouannaud et Mitsuhiro Okada dans un manuscrit non publié de 1997 [JO97b] qui a servi de base à mon travail de master [Bla98]. Elle apparaît ensuite dans [BJO99, JR99].

Dans [JO91, JO97a], Jean-Pierre Jouannaud et Mitsuhiro Okada ont introduit une notion de "spécification algébrique d'ordre supérieur exécutable" basée sur la combinaison de la β-réduction et d'un ensemble $\mathcal{R}$ de règles de réécriture d'ordre supérieur dont les membres gauches sont algébriques, *i.e.* ne contiennent ni abstraction ni variable appliquée, ce qui évite d'avoir à utiliser du filtrage d'ordre supérieur. Un tel système n'est rien d'autre en fait qu'un cas particulier de système de réduction combinatoire (CRS) typé [Bla00]. Ensuite, ils montrent qu'un tel système termine si, d'une part, les règles du premier ordre non-dupliquantes[10] terminent et si, d'autre part, les autres règles appartiennent à un schéma de règles, dit "schéma général", qui généralise la récursion primitive.

Pour montrer la correction du schéma général, ils utilisent la technique de William W. Tait et Jean-Yves Girard des prédicats de calculabilité (ou réductibilité) [Tai67, Gir71, Tai75, Gir72, GLT88, Gal90]. Dans ce cas, pour montrer que l'application d'un symbole de fonction f à des arguments calculables $\vec{t}$ est elle-même calculable, il suffit en effet de montrer que, pour toute règle de réécriture $\mathsf{f}\vec{l} \to r$ et substitution σ tels que $\vec{t} = \vec{l}\sigma$, $r\sigma$ est calculable.

L'approche par clôture de calculabilité consiste alors à supposer que r appartient à un ensemble, la clôture de calculabilité de $\mathsf{f}\vec{l}$, notée $\mathrm{CC}(\mathsf{f}\vec{l})$ par la suite, contenant les termes $\vec{l}$ et clos par des opérations préservant la calculabilité, *i.e.* telles que $r\sigma$ est calculable si les termes $\vec{l}\sigma$ sont calculables, comme c'est le cas des opérations suivantes [Bla07] :

1. $\{\vec{l}\} \subseteq \mathrm{CC}(\mathsf{f}\vec{l})$,
2. $r_i \in \mathrm{CC}(\mathsf{f}\vec{l})$ si $\mathsf{g}\vec{r} \in \mathrm{CC}(\mathsf{f}\vec{l})$ et le type de $\mathsf{g}\vec{r}$ apparaît uniquement positivement dans le type de r_i (voir la Section 7.1.2 sur les types inductifs),

10. $l \to r$ est *non-dupliquante* si aucune variable a plus d'occurrences dans r que dans l

3. $\mathsf{g}\vec{r} \in \mathrm{CC}(\mathsf{f}\vec{l})$ si $\mathsf{f} >_{\mathcal{F}} \mathsf{g}$ et $\vec{r} \in \mathrm{CC}(\mathsf{f}\vec{l})$, où $>_{\mathcal{F}}$ est un pré-ordre bien fondé sur les symboles de fonction (précédence),
4. $\mathsf{g}\vec{r} \in \mathrm{CC}(\mathsf{f}\vec{l})$ si $\mathsf{f} =_{\mathcal{F}} \mathsf{g}$, $\vec{r} \in \mathrm{CC}(\mathsf{f}\vec{l})$ et $\vec{t}\ R(\to_{\mathcal{R}}^{+} \cup \rhd)\ \vec{r}$, où $\lhd$ est l'ordre sous-terme et R un opérateur préservant la bonne fondation comme l'extension lexicographique ou multi-ensemble,
5. $t \in \mathrm{CC}(\mathsf{f}\vec{l})$ si $u \in \mathrm{CC}(\mathsf{f}\vec{l})$ et $u \to_{\mathcal{R}}^{+} t$,
6. $tu \in \mathrm{CC}(\mathsf{f}\vec{l})$ si $t \in \mathrm{CC}(\mathsf{f}\vec{l})$ et $u \in \mathrm{CC}(\mathsf{f}\vec{l})$,
7. $\lambda x.t \in \mathrm{CC}(\mathsf{f}\vec{l})$ si $t \in \mathrm{CC}(\mathsf{f}\vec{l})$,
8. $x \in \mathrm{CC}(\mathsf{f}\vec{l})$ si $x \notin \mathrm{FV}(\mathsf{f}\vec{l})$.

Mais d'autres opérations peuvent être envisagées et mon travail de recherche sur ce sujet a consisté à étendre cette clôture de différente manière, ce que je détaille dans les sections suivantes.

7.2.2 Réécriture modulo théories

Conférences : [Bla03b]

Dans [Bla03b], je montre que la clôture de calculabilité est également adaptée à la réécriture modulo un ensemble $\mathcal{E}$ d'équations du type décrit en Section 7.1.4, *i.e.* la relation $\sim\!\!\to_{\mathcal{R}}$.

La clôture de calculabilité d'un terme $\mathsf{f}\vec{l}$ a la propriété suivante : si les termes $\vec{l}$ sont calculables, alors tout terme $t \in \mathrm{CC}(\mathsf{f}\vec{l})$ est calculable. De plus, pour montrer la calculabilité d'un terme de la forme $\mathsf{f}\vec{l}$, il suffit de montrer la calculabilité de tous ses réduits de tête.

Dans le cas de la réécriture modulo, nous avons $\mathsf{f}\vec{l} \to_{\mathcal{E}}^{*} \mathsf{g}\vec{m} \to_{\mathcal{R}} r$. Il suffit alors de s'assurer que les termes $\vec{m}$ sont calculables pour pouvoir se ramener au cas non modulo. Or, cela peut justement se faire grâce à la clôture elle-même si, pour toute équation $\mathsf{f}\vec{l} \to \mathsf{g}\vec{m} \in \mathcal{E}$, les termes $\vec{m}$ appartiennent à la clôture de $\mathsf{f}\vec{l}$. Cela est toujours vérifié pour des équations telles que la commutativité puisque, par définition, $\{\vec{l}\} \subseteq \mathrm{CC}(\mathsf{f}\vec{l})$. Pour l'associativité, c'est par exemple le cas pour les symboles du premier ordre. Par exemple, avec $+ : \mathbb{N} \Rightarrow \mathbb{N} \Rightarrow \mathbb{N}$, nous avons $\{x, +yz\} \subseteq \mathrm{CC}(+(+xy)z)$ car, si $+xy$ et z sont calculables, alors x est calculable en tant que sous-terme de $+xy$, et $+yz$ est calculable car y est un argument plus petit que $+xy$ (schéma général).

J'ai également étendu le résultat de Jean-Pierre Jouannaud et Mitsuhiro Okada sur la combinaison hiérarchique de réécriture du premier ordre et d'ordre supérieur [JO91, JO97a] au cas de la réécriture modulo : la combinaison d'un système du premier ordre non-dupliquant et terminant modulo certaines équations et d'un système d'ordre supérieur lui aussi modulo vérifiant la clôture de calculabilité (anciennement, le schéma général) termine si les équations de la forme décrite en Section 7.1.4 sont linéaires et les classes d'équivalence sont finies.

7.2.3 Annotations de taille

Conférences : [Bla04d, Bla05a, BR06, BR09]
Rapports : [Bla10]

Certains de ces travaux ont été faits en collaboration avec Colin Riba et Cody Roux dont j'ai co-encadré les thèses [Rib07a, Rou11] avec Claude Kirchner (et Gilles Dowek pour Cody).

En 1996 paraissent deux travaux introduisant indépendamment la notion de type avec annotation de taille : un article de John Hughes, Lars Pareto et Amr Sabri à POPL [HPS96], considérant le λ-calcul simplement typé, et la thèse de doctorat de Carlos Eduardo Giménez [Gim96], considérant le calcul des constructions. L'idée est simple : annoter les constantes de type par des expressions bornant la taille des termes. Par exemple, $\mathbb{L}^i$ dénote le type des listes de taille inférieure ou égale à i. Par "taille", ces auteurs entendent une notion dérivée de la sémantique des types inductifs qui, dans le cas d'un terme clos, correspond à la hauteur de l'arbre représentant sa *forme normale*. Pour une liste, il s'agit de la longueur. Les règles de typage permettent alors de déduire de l'information non seulement sur le type d'une expression mais également sur sa taille, et cette information peut naturellement être utilisée pour montrer la terminaison d'une fonction.

Ces travaux furent ensuite approfondis et étendus par différents auteurs dont Hongwei Xi, Andreas Abel et Gilles Barthe [Xi01, Xi02, Abe02, Abe03, Abe04, BFG+04, Abe08].

Un autre travail proche est celui de Christoph Zenger en 1997 [Zen97], qui considère une extension du système de types de J. Roger Hindley and Robin Milner (langage ML) et le problème d'inférence de type correspondant.

Comparer la taille des arguments plutôt que leur structure permet de montrer la terminaison de davantage de fonctions car, comparer la taille de deux termes revient d'une certaine manière à comparer la taille de toutes les formes normales de leurs instances closes. Ainsi, avec la fonction $\mathsf{pivot} : \mathbb{N} \Rightarrow \mathbb{L} \Rightarrow \mathbb{L} \times \mathbb{L}$ définie ci-après qui sépare les éléments d'une liste en deux sous-listes, les éléments plus petits que le pivot d'une part, et les éléments plus grands ou égaux au pivot d'autre part :

$$\begin{array}{rcl} \mathsf{pivot}\, x\, \mathsf{nil} & \to & (\mathsf{nil}, \mathsf{nil}) \\ \mathsf{pivot}\, x\, (\mathsf{cons}\, y\, \ell) & \to & (\mathsf{cons}\, y\, (\mathsf{fst}\, (\mathsf{pivot}\, \ell)), \mathsf{snd}\, (\mathsf{pivot}\, \ell)) \Leftarrow y < x \\ \mathsf{pivot}\, x\, (\mathsf{cons}\, y\, \ell) & \to & (\mathsf{fst}\, (\mathsf{pivot}\, \ell), \mathsf{cons}\, y\, (\mathsf{snd}\, (\mathsf{pivot}\, \ell))) \Leftarrow y \geq x \end{array}$$

la "taille" de $(\mathsf{fst}\, (\mathsf{pivot}\, \ell))$ ou de $(\mathsf{snd}\, (\mathsf{pivot}\, \ell))$ est inférieure ou égale à la taille de ℓ, bien que ℓ soit structurellement plus petit que $(\mathsf{fst}\, (\mathsf{pivot}\, \ell))$ ou $(\mathsf{snd}\, (\mathsf{pivot}\, \ell))$. Et cela est une conséquence inductive de la définition de pivot.

Dans [Bla04d], j'étends l'approche de Hughes-Pareto-Sabri et Giménez au calcul des constructions algébriques et donc, en particulier, à la réécriture, c'est-à-dire à des fonctions définies par des règles de réécriture, alors que les auteurs

précédents considèrent des fonctions définies par point fixe et filtrage constructeur.

Pour cela, et contrairement aux travaux précédents, j'introduis une notion abstraite d'algèbre de taille et définis une extension du calcul des constructions où les constantes de type sont annotées par des éléments de cette algèbre, sans préciser laquelle à priori. En effet, de nombreuses propriétés méta-théoriques ne dépendent absolument pas du type d'annotations considérées. Comme nous sommes amener à comparer ces annotations de taille et que celles-ci représentent des bornes supérieures, il est naturel de considérer qu'une algèbre de taille est un ensemble ordonné et d'étendre le système de type avec une relation de sous-typage engendré par l'ordre sur les tailles. Par exemple, $\mathbb{L}^i$ doit être un sous-type de $\mathbb{L}^{i+1}$.

Le critère de terminaison basé sur la comparaison des annotations de taille dérivées du typage n'est rien d'autre qu'une nouvelle extension de la clôture de calculabilité. Pour montrer la terminaison de la réécriture par la méthode de Tait-Girard [GLT88], il convient en effet de montrer la calculabilité de tout terme de la forme $\mathsf{f}\vec{l}$, et cela peut se faire en raisonnant par induction sur $\vec{l}$ avec n'importe quel ordre bien fondé compatible avec la réduction. Or, il se trouve que la taille d'un terme constitue justement un tel ordre.

A la fin de [Bla04d], je donne également deux exemples intéressants qui ne satisfont pas le critère de terminaison décrit dans le papier et qui donneront lieu à d'autres travaux ensuite. Le premier exemple est celui d'un système de réécriture permettant de normaliser les expressions conditionnelles [BM79] dont la terminaison peut être montrée en utilisant une notion de taille autre que la hauteur [Pau86]. Le deuxième exemple est la fonction 91 de Mc Carthy [MP69] dont la preuve de terminaison nécessite de traiter de manière particulière la construction `if then else`. De plus, je donne un exemple qui, bien qu'il satisfasse le critère de terminaison, n'est pas complètement satisfaisant d'un autre point de vue. Cet exemple est l'algorithme de tri *quicksort* qui utilise la fonction pivot précédente. Avec le système d'annotations décrit dans le papier, il est possible de montrer la terminaison de *quicksort*. Par contre, il n'est pas possible de déduire que le résultat de *quicksort* a même longueur que son argument. Ces deux derniers exemples conduiront aux travaux de thèse de Colin Riba sur la terminaison et la confluence de la réécriture conditionnelle [BKR06, BKR10, BR06, Rib07a, BR09].

Dans [Bla05a], je prouve la décidabilité du critère de terminaison décrit dans [Bla04d]. Tout d'abord, pour montrer la transitivité de la conversion de type, je réutilise une méthode introduite par Gang Chen dans sa thèse sur le calcul des constructions avec sous-typage [Che98] (cela apparaîtra d'abord dans une version longue de [Bla04d] puis dans [Bla10]). Ensuite, je fournis un algorithme générique basé sur une procédure de résolution de contraintes d'ordre dans l'algèbre de taille dont je montre la correction et la complétude dans le cas où tout ensemble de contraintes satisfaisable admet une solution la plus générale (à renommage près des variables). Enfin, je montre que la résolution de contraintes d'ordre dans l'algèbre de taille utilisée dans [Bla04d, Abe04, BFG+04]

(algèbre "successeur") est décidable. Cela se fait en deux étapes. Premièrement, la propagation des tailles indéfinies en utilisant l'algorithme de Richard Bellman et L. R. Ford [For56, Bel58]. On obtient alors un système d'équations linéaires à deux inconnues pour lesquels il existe des algorithmes efficaces, *e.g.* [LMR90].

Dans [Bla10], je détaille et généralise les deux articles de conférence précédents. En particulier, je montre qu'il est possible de prendre comme notion de taille toute fonction monotone et strictement "extensive", c'est-à-dire, telle que $f(x) > x$, répondant ainsi positivement à une idée émise dans [Bla04d]. Par ailleurs, je montre qu'avec l'algèbre de taille "successeur" utilisée dans [Bla04d, Abe04, BFG$^+$04], il n'est pas possible de considérer certains membres gauches de règles non-linéaires ou de hauteur supérieur à deux (voir Section 6.1 dans [Bla10] ; cela est également mentionné dans [BR09]).

Dans [BR06], Colin et moi étendons [Bla04d, Bla05a] dans plusieurs directions :

1. Nous considérons des types avec quantifications universelles et existentielles ainsi que des contraintes sur les variables de types. Dans ce cas, le type $\mathbb{L}^i$ ne représente plus les listes de longueur inférieure ou égale à i mais *exactement* celles de longueur i, l'ancienne sémantique pouvant être récupérée en utilisant une quantification existentielle contrainte $(\exists j(j \leq i)\mathbb{L}^j)$. Pour montrer la correction du critère de terminaison, il convient alors d'étendre à la quantification existentielle l'interprétation des types par des candidats de réductibilité à la Girard. Nous montrons alors qu'une classe particulière de types est stable par union si la réduction conflue. Plus tard, Colin approfondira cette question et étudiera en détail les propriétés de l'union dans les ensembles saturés de William W. Tait [Tai75], les candidats de réductibilité [Gir72, GLT88, Gal90] et les ensembles bi-orthogonaux de Jean-Yves Girard [Gir87] dans [Rib07a, Rib07b, Rib07c, Rib08, Rib09].
2. Le calcul est également étendu avec une construction `let` permettant le partage non seulement d'une valeur mais, surtout, de son information de taille. De cette façon, nous pouvons déduire que *quicksort* préserve bien la taille, *i.e.* lui donner le type $\forall\alpha\mathbb{L}^\alpha \Rightarrow \mathbb{L}^\alpha$, car il est désormais possible de typer la fonction de pivot de la manière suivante :

$$\mathsf{pivot} : \mathbb{N} \Rightarrow \forall\alpha\mathbb{L}^\alpha \Rightarrow \exists\beta\gamma(\alpha = \beta + \gamma)\mathbb{L}^\beta \times \mathbb{L}^\gamma$$

3. Comme Hongwei Xi dans [Xi02], nous traitons de manière particulière la construction `if then else`, ce qui nous permet de montrer la terminaison de fonctions comme la fonction 91 de Mc Carthy [MP69]. Cependant, contrairement à Xi qui ne s'intéresse qu'à la normalisation faible de programmes ML, nous nous intéressons à la normalisation forte (*i.e.* en autorisant la réécriture sous les λ).
4. Enfin, nous étendons le critère de terminaison à la réécriture conditionnelle avec des conditions de la forme $l = \mathsf{true}$ ou $l = \mathsf{false}$, où true et false sont

les deux constructeurs du type des booléens. Il s'agit du premier critère de terminaison pour la réécriture conditionnelle d'ordre supérieur ! Le critère étend la clôture de calculabilité aux appels récursifs dont les arguments sont plus petits en supposant que soient satisfaites les contraintes engendrées par les conditions de réécriture. Au passage, il est intéressant de noter que le système de type avec contraintes fournit une formulation plus compacte et élégante de la clôture de calculabilité en tant que sous-système de typage.

Enfin, dans [BR09], Cody et moi avons étudié quel lien il pouvait y avoir entre les annotations de taille et l'*étiquetage sémantique* introduit pour les systèmes de réécriture du premier ordre par Hans Zantema dans [Zan95] et étendu à l'ordre supérieur par Makoto Hamana [Ham07]. L'étiquetage sémantique consiste à annoter les symboles de fonction par la sémantique de leurs arguments dans un modèle du système de réécriture, *i.e.* un ensemble ordonné $(D, \geq_D)$ où les symboles de fonctions sont interprétés par des fonctions *monotones*, et tel que $[\![l]\!]\nu \geq_D [\![r]\!]\nu$ pour toute règle $l \to r$ et valuation $\nu : \mathcal{X} \to D$. Bien que cela conduit généralement à un ensemble infini de règles de réécriture, sa terminaison peut se révéler plus facile. En fait, en théorie, à tout système qui termine, on peut associer un modèle avec lequel le système étiqueté est inclus dans un ordre récursif sur les chemins [Der79].

Ainsi, nous avons montré que les annotations de taille constitue un modèle d'étiquetage particulier et que le critère de terminaison basé sur les annotations de taille permet de montrer la terminaison du système étiqueté. L'étiquetage sémantique constitue donc un cadre plus général dans lequel exprimer et étendre le critère de terminaison basé sur les annotations de taille :

1. Il est tout à fait possible de considérer des notions de taille (l'interprétation des constructeurs) autres que la hauteur, comme cela est également montré dans [Bla10].
2. Alors que le critère de terminaison basé sur les annotations de taille ne s'applique qu'aux systèmes constructeurs, l'étiquetage sémantique s'applique à tout système de réécriture, y compris ceux avec filtrage sur symboles définis comme dans la règle d'associativité $x + (y + z) \to (x + y) + z$.

Enfin, dans la deuxième partie de sa thèse [Rou11], Cody cherche à utiliser les annotations de taille pour obtenir une approximation du graphe de dépendance (Section 7.2.5).

7.2.4 Ordre récursif sur les chemins

Conférences : [BJR06, BJR07, BJR08]
Rapports : [Bla06b]

Certains de ces travaux ont été faits en collaboration avec Jean-Pierre Jouannaud et Albert Rubio.

L'ordre récursif sur les chemins, $>_{\text{rpo}}$, est un ordre de réduction (*i.e.* bien fondé et stable par substitution et contexte) décidable inventé par Nachum Dershowitz en 1979 [Der79, Der82]. Il étend un ordre sur les symboles $>_{\mathcal{F}}$ (précédence) en un ordre sur les termes (en fait, il s'agit de *pré-ordres*, *i.e.* de relations non nécessairement antisymétriques). Il est défini inductivement par les règles suivantes :

1. $\mathsf{f}\vec{l} >_{\text{rpo}} t$ s'il existe i tel que $l_i \geq_{\text{rpo}} t$,
2. $\mathsf{f}\vec{l} >_{\text{rpo}} \mathsf{g}\vec{r}$ si $\mathsf{f} >_{\mathcal{F}} \mathsf{g}$ et $\mathsf{f}\vec{l} >_{\text{rpo}} \vec{r}$,
3. $\mathsf{f}\vec{l} >_{\text{rpo}} \mathsf{g}\vec{r}$ si $\mathsf{f} =_{\mathcal{F}} \mathsf{g}$, $\mathsf{f}\vec{l} >_{\text{rpo}} \vec{r}$ et $\vec{l}\, R(>_{\text{rpo}})\, \vec{r}$, où R est un opérateur préservant la bonne fondation comme l'extension lexicographique ou multi-ensemble.

Plusieurs tentatives ont été faites pour généraliser $>_{\text{rpo}}$ à l'ordre supérieur [LSS92, LP95, JR96] jusqu'à la définition de l'ordre récursif sur les chemins à l'ordre supérieur, $>_{\text{horpo}}$, de Jean-Pierre Jouannaud et Albert Rubio en 1999 [JR99]. Alors que la preuve de bonne fondation de $>_{\text{rpo}}$ donnée par Nachum Dershowitz repose sur le théorème de Joseph Bernard Kruskal [Kru60], celle de $>_{\text{horpo}}$ repose sur la technique de Tait-Girard pour le λ-calcul [GLT88] (ce qui du même coup donne une nouvelle preuve de bonne fondation de $>_{\text{rpo}}$), comme c'est également le cas du schéma général [JO91] et de la clôture de calculabilité [BJO99]. Par la suite, alors même qu'ils partagent une même technique, les deux critères de terminaison ont continué à être développé indépendamment l'un de l'autre. $>_{\text{horpo}}$ a notamment été développé par ses auteurs [BR01, JR07, JR06] et étendu au calcul des constructions par Daria Walukiewicz [WC03a, WC03b]. Pourtant, la clôture de calculabilité elle-même est utilisée pour renforcer l'expressivité de $>_{\text{horpo}}$ [JR99].

Dans [Bla06b], j'étudie donc les liens qu'il peut y avoir entre ces deux critères de terminaison, en partant de la constatation simple que la clôture de calculabilité elle-même fourni un ordre :

$$\mathsf{f}\vec{l} >_{\text{CC}} t \text{ si } t \in \text{CC}(\mathsf{f}\vec{l})$$

Ainsi, la définition de la clôture de calculabilité donnée en Section 7.2.1 peut se réécrire comme la définition d'un ordre de la manière suivante :

1. $\mathsf{f}\vec{l} >_{\text{CC}} \vec{l}$,
2. $\mathsf{f}\vec{l} >_{\text{CC}} r_i$ si $\mathsf{f}\vec{l} >_{\text{CC}} \mathsf{g}\vec{r}$ et le type de $\mathsf{g}\vec{r}$ apparaît uniquement positivement dans le type de r_i,
3. $\mathsf{f}\vec{l} >_{\text{CC}} \mathsf{g}\vec{r}$ si $\mathsf{f} >_{\mathcal{F}} \mathsf{g}$ et $\mathsf{f}\vec{l} >_{\text{CC}} \vec{r}$,
4. $\mathsf{f}\vec{l} >_{\text{CC}} \mathsf{g}\vec{r}$ si $\mathsf{f} =_{\mathcal{F}} \mathsf{g}$, $\mathsf{f}\vec{l} >_{\text{CC}} \vec{r}$ et $\vec{l}\, R(\to_{\mathcal{R}}^{+} \cup \rhd)\, \vec{r}$,
5. $\mathsf{f}\vec{l} >_{\text{CC}} t$ si $\mathsf{f}\vec{l} >_{\text{CC}} u$ et $u \to_{\mathcal{R}}^{+} t$,
6. $\mathsf{f}\vec{l} >_{\text{CC}} tu$ si $\mathsf{f}\vec{l} >_{\text{CC}} t$ et $\mathsf{f}\vec{l} >_{\text{CC}} u$,
7. $\mathsf{f}\vec{l} >_{\text{CC}} \lambda x.t$ si $\mathsf{f}\vec{l} >_{\text{CC}} t$,
8. $\mathsf{f}\vec{l} >_{\text{CC}} x$ si $x \notin \text{FV}(\mathsf{f}\vec{l})$.

On voit alors clairement la similarité avec $>_{\text{rpo}}$ ou $>_{\text{horpo}}$, et ce qui l'en distingue également (la condition de positivité est toujours vérifié au premier ordre) : l'ordre utilisé pour comparer les arguments de deux symboles de fonction équivalents n'est pas $>_{\text{CC}}$ lui-même ; autrement dit, l'ordre $>_{\text{CC}}$ n'est pas récursif.

Cependant, la fonctionnelle qui à la relation $\mathcal{R}$ associe l'ordre $>_{\text{CC}}(\mathcal{R})$ est monotone et admet donc un point fixe [Tar55] qui, lui, est récursif. J'appelle ce point fixe l'ordre de calculabilité récursif d'ordre supérieur faible et le note $>_{\text{whorco}}$. Cet ordre est dit *faible* car il n'est pas stable par contexte. Cela est dû au fait que la clôture de calculabilité n'est définie que pour les termes de la forme $f\vec{l}$. Mais ce n'est pas nécessairement un désavantage, bien au contraire ! D'une part, il est généralement plus difficile de construire un ordre stable par contexte. D'autre part, lorsqu'on utilise la technique des paires de dépendance, qui est désormais la technique de base pour montrer la terminaison des systèmes du premier ordre, il n'est pas nécessaire d'utiliser un ordre stable par contexte pour orienter les paires de dépendance (voir la Section 7.2.5).

Je montre alors que $>_{\text{horpo}}$ est inclus dans la clôture transitive de $>_{\text{horco}}$, la clôture par contexte de $>_{\text{whorco}}$ (la clôture transitive n'est pas nécessaire si on modifie légèrement la définition de $>_{\text{horpo}}$). Ainsi, la bonne fondation de $>_{\text{horpo}}$ peut se réduire à la correction de la clôture de calculabilité, et toute extension de la clôture de calculabilité conduit immédiatement à une extension de $>_{\text{horpo}}$. La clôture de calculabilité constitue donc une manière d'étendre $>_{\text{horpo}}$, en particulier à des systèmes de types complexes. D'un autre côté, la décidabilité de l'ordre obtenu par point fixe paraît moins évidente et n'a pas encore été étudiée.

Il est également intéressant de noter que, si l'on se restreint aux termes du premier ordre, alors $>_{\text{whorco}}$ est en fait exactement égal à $>_{\text{rpo}}$! Dans ce cas, $>_{\text{whorco}}$ est donc stable par contexte et transitif.

Après cela, dans [BJR06, BJR07, BJR08], j'ai collaboré avec Jean-Pierre Jouannaud et Albert Rubio sur l'extension de $>_{\text{horpo}}$ de façon à y inclure la notion de positivité présentée en Section 7.1.2. En effet, contrairement à ce qui se passe au premier ordre, l'ordre sous-terme ne préserve pas la calculabilité à l'ordre supérieur. Par contre, il préserve la calculabilité si on le restreint par des conditions de positivité (qui sont toujours satisfaites au premier ordre).

7.2.5 Paires de dépendance

Journaux : [KISB09, SKB11]
Workshops : [Bla06a]
Rapports : [SKB10]

Certains de ces travaux ont été faits en collaboration avec Keiichirou Kusakari et Masahiko Sakai.

En 1997, Thomas Arts and Jürgen Giesl ont montré que la terminaison d'un système de réécriture du premier ordre peut se réduire à l'étude du *graphe de*

dépendance de ses *paires de dépendance* [AG97, AG00]. Étant donné un ensemble $\mathcal{R}$ de règles de réécriture, une *paire de dépendance* n'est rien d'autre qu'une paire $(\mathsf{f}\vec{l}, \mathsf{g}\vec{r})$ où $\mathsf{f}\vec{l}$ est le membre gauche d'une règle $\mathsf{f}\vec{l} \to r \in \mathcal{R}$ et $\mathsf{g}\vec{r}$ un sous-terme de r telle que g soit un symbole défini, *i.e.* pour lequel il existe une règle de $\mathcal{R}$ de la forme $\mathsf{g}\vec{m} \to s$. Alors, $\to_{\mathcal{R}}$ termine si et seulement si la relation $\overset{\neq\varepsilon}{\to}{}^*_{\mathcal{R}} \overset{\varepsilon}{\to}_{\mathcal{P}}$ termine, où $\mathcal{P}$ est l'ensemble des paires de dépendance de $\mathcal{R}$, $\overset{\varepsilon}{\to}_{\mathcal{R}}$ la réécriture en tête et $\overset{\neq\varepsilon}{\to}_{\mathcal{R}}$ la réécriture à des positions autres qu'en tête. Le *graphe de dépendance* de $\mathcal{R}$ est alors le graphe dont les noeuds sont les paires de dépendance et tel qu'il y a un arc entre $(\mathsf{f}\vec{l}, \mathsf{g}\vec{r})$ et $(\mathsf{g}\vec{m}, \mathsf{h}\vec{s})$ s'il existe deux substitutions σ et θ telles que $\mathsf{g}\vec{r}\sigma \overset{\neq\varepsilon}{\to}{}^*_{\mathcal{R}} \mathsf{h}\vec{m}\theta$. Ainsi, un chemin dans le graphe représente une séquence "d'appels de fonctions" : $\to_{\mathcal{R}}$ termine si et seulement s'il n'existe aucun chemin infini dans le graphe de dépendance. Cette approche de la terminaison est maintenant devenue la base de tous les prouveurs automatiques de terminaison pour les systèmes du premier ordre [GTSKF06, HM07, Thi07].

C'est pourquoi plusieurs auteurs ont cherché à étendre la notion de paire de dépendance à la réécriture d'ordre supérieur, dont en particulier Keiichirou Kusakari et Masahiko Sakai [SWS01, SK01, SK05] pour les HRS (réécriture sur formes β-normales η-longues) [Nip91].

Dans [Bla06a], je considère le cas des systèmes de réduction combinatoires de Jan Willem Klop (CRS) [Klo80] sans me restreindre aux formes β-normales η-longues. Je défini une notion de paires de dépendance similaire à celle du premier ordre pour laquelle je généralise le théorème des paires de dépendance ($\to_\beta \cup \to_{\mathcal{R}}$ termine si $\to_\beta \cup \overset{\neq\varepsilon}{\to}{}^*_{\mathcal{R}} \overset{\varepsilon}{\to}_{\mathcal{P}}$ termine) au cas où les membres gauches de règles sont des motifs à la Miller [Mil89, Bla00] et les paires de dépendance préservent le typage et les variables, *i.e.* pour toute paire (l, r), l et r ont le même type et $\mathrm{FV}(r) \subseteq \mathrm{FV}(l)$.

La preuve donnée au premier ordre par Thomas Arts et Jürgen Giesl utilise le tiers exclus et l'axiome du choix en montrant qu'à partir de toute séquence infinie de $\to_{\mathcal{R}}$ on obtient une séquence infinie de $\overset{\neq\varepsilon}{\to}{}^*_{\mathcal{R}} \overset{\varepsilon}{\to}_{\mathcal{P}}$. Au contraire, dans [Bla06a], je donne une preuve intuitionniste en montrant la bonne fondation de $\to_\beta \cup \to_{\mathcal{R}}$ par induction bien fondée sur $\to_\beta \cup \overset{\neq\varepsilon}{\to}{}^*_{\mathcal{R}} \overset{\varepsilon}{\to}_{\mathcal{P}}$. Et il se trouve que cette preuve est similaire à la preuve de correction de la clôture de calculabilité ! J'ai formalisé ensuite cette preuve dans Coq dans le cas du premier ordre (voir Section 8.3).

J'ai également étendu à l'ordre supérieur le théorème de Arts et Giesl (étendu aux "paires de réduction" par Keiichirou Kusakari *et al* in [KNT99]) disant que $\to_{\mathcal{R}}$ termine s'il existe une paire de relations $(\geq, >)$ telles que $\geq$ soit un pré-ordre stable par substitution et contexte, $>$ soit une relation bien fondée et stable par substitution (mais pas nécessairement par contexte !), $\geq > \subseteq >$, $\mathcal{R} \subseteq \geq$ et $\mathcal{P} \subseteq >$ (seules les paires de dépendance doivent être orientées strictement). Là encore, la preuve est similaire à la preuve de correction de la clôture de calculabilité si, pour définir la calculabilité, on prend la relation $\to_{\mathcal{P}}$ au lieu de la relation $\to_{\mathcal{R}}$! On voit qu'ainsi l'ordre $>_{\mathrm{whorco}}$ non stable par contexte défini dans [Bla06b] est tout à fait intéressant.

Ensuite, dans [KISB09, SKB10, SKB11], j'ai collaboré avec Keiichirou Kusakari et Masahiko Sakai afin d'améliorer leur théorème des paires de dépendance pour les HRS en utilisant la notion de clôture de calculabilité. Cela permet en effet d'élargir considérablement la classe de systèmes auxquels peut s'appliquer leur théorème. Cependant, que ce soit dans les CRS ou les HRS, le traitement des paires ne préservant pas le typage ou les variables continuent de poser problème car, même si un théorème de paires de dépendance peut être établi dans ces cas-là, aucun ordre ne permet actuellement de les orienter.

Enfin, dans la deuxième partie de sa thèse [Rou11], Cody cherche à utiliser les annotations de taille (Section 7.2.3) afin d'obtenir une approximation du graphe de dépendance.

7.3 Fonctions de constructions pour types quotients

Conférences : [BHW07]

Ce travail a été fait en collaboration avec Thérèse Hardin et Pierre Weis, et a fait l'objet d'une implantation décrite en Section 8.4.

Dans [BHW07], nous nous intéressons au problème de programmation suivant. Nous considérons un type de données *concret* (ou inductif), *i.e.* un type défini par un ensemble de *constructeurs*. Sans autre restriction, les valeurs de ce type sont alors tous les termes (clos) que l'on peut former en composant ces constructeurs. Cependant, il est fréquent de vouloir restreindre les valeurs possibles afin, par exemple, de préserver certains invariants. Comment alors garantir qu'aucune valeur violant ces invariants ne sera jamais construite ?

La première solution consiste à utiliser la notion de *module* avec, d'une part une interface déclarant un type de données *abstrait* et un ensemble de *fonctions de construction* permettant de construire des valeurs de ce type et, d'autre part, une implantation de ce type et de ces fonctions [Mor73, LZ74, Gut77]. L'inconvénient de cette solution est qu'elle ne permet pas de définir des fonctions par filtrage sur les constructeurs [GMW77, BQS80, Mil84, Aug85, FM01, Mar92, MRV03] car la définition du type est inconnue de ou non utilisable par l'utilisateur (principe d'abstraction).

Cela a suscité un certain nombre de recherches pour essayer de contourner ce problème dont, en particulier, la notion de *vue* de Philip Wadler [Wad87, BMS+96, Oka98]. Mais la bonne solution est due à F. Warren Burton et Robert D. Cameron en 1993 [BC93]. Dans celle-ci, les constructeurs peuvent être utilisés dans des motifs de filtrage mais ils ne peuvent pas l'être pour construire des valeurs. Là encore, il faut passer par des fonctions de construction. Cette solution a été redécouverte et implantée par Pierre Weis dans OCaml [LDF+10] en 2003 où un tel type est dit *privé* [Wei03].

Dans tous les cas donc, que le type soit déclaré *abstrait* ou *privé*, il est nécessaire de fournir des fonctions de construction garantissant les invariants souhaités. Dans [BHW07], nous nous intéressons au cas où ces invariants sont décrits sous la forme de règles de réécriture telles que l'associativité, la neutralité,

la distributivité, etc. Par exemple, on peut vouloir représenter des ensembles en utilisant le type concret OCaml [LDF+10] suivant :

```
type set = Empty | Singleton of int | Union of set * set
```

Et vouloir qu'aucune valeur de type `set` ne soit de la forme $\texttt{Union}(\texttt{Empty}, s)$, ce qu'on peut réaliser en déclarant `set` abstrait ou privé, et en fournissant les fonctions de construction suivantes :

```
(* file set.mli: interface of module Set *)
type set = private Empty | Singleton of int | Union of set * set
val empty : set
val singleton : int -> set
val union : set -> set -> set

(* file set.ml: implantation of module Set *)
type set = Empty | Singleton of int | Union of set * set
let empty = Empty
let singleton x = Singleton x
let union = function
| Empty, s -> s
| s, Empty -> s
| s, t -> Union (s,t)
```

Ainsi, l'ensemble des valeurs de type `set` qu'un utilisateur du module `Set` peut construire en utilisant les fonctions de constructions `empty`, `singleton` et `union`, est un sous-ensemble particulier de l'ensemble de tous les termes que l'on peut former à partir des constructeurs `Empty`, `Singleton` et `Union`. En effet, deux termes équivalents modulo l'équation :

$$\texttt{Union}(\texttt{Empty}, x) = x$$

sont représentés par la même valeur, et celle-ci peut être obtenue en appliquant la règle de réécriture suivante :

$$\texttt{Union}(\texttt{Empty}, x) \to x$$

De même, vouloir que $\texttt{Union}(t, \texttt{Union}(u, v))$ et $\texttt{Union}(\texttt{Union}(t, u), v)$ soient représentés par la même valeur peut se faire en appliquant la règle d'associativité :

$$\texttt{Union}(\texttt{Union}(t, u), v) \to \texttt{Union}(t, \texttt{Union}(u, v))$$

Dans ce cas, toute valeur de type `set` est en fait un peigne et peut donc être vue comme une liste. On peut alors aller un peu plus loin et faire en sorte que $\texttt{Union}(t, u)$ et $\texttt{Union}(u, t)$ soient représentés par la même valeur. Pour cela, il suffit de se donner un ordre total sur les termes $>$ et d'appliquer la règle conditionnelle suivante :

$$\texttt{Union}(t, u) \to \texttt{Union}(u, t) \Leftarrow t > u$$

dont la théorie équationnelle correspondante est la commutativité.

On peut enfin achever cette représentation des ensembles en faisant en sorte que la fonction de construction pour l'union de deux ensembles soit idempotente :

$$\texttt{Union}(t, u) \to t \Leftarrow t = u$$

Dans ce cas, il est intéressant de noter que, vu que t et u sont nécessairement en forme normale *par construction*, si les valeurs sont maximalement partagées [CF06], alors le test d'égalité peut être réalisé en temps constant en utilisant l'égalité des pointeurs en mémoire plutôt que l'égalité structurelle.

En conclusion, on voit qu'un ensemble de valeurs peut être spécifié par un ensemble d'équations et implanté en utilisant un ensemble de règles conditionnelles terminant et confluant décidant la théorie équationnelle engendrée par ces équations.

Le problème de trouver, pour un ensemble donné d'équations, un ensemble de règles de réécriture terminant et confluant décidant cette théorie est un problème appelé *complétion* bien connu en réécriture du premier ordre depuis les travaux pionniers de Donald E. Knuth et P. B. Bendix en 1967 [KB70]. Cette technique a été utilisée et étendue pour obtenir des procédures de décision pour de nombreuses théories équationnelles, notamment par Jean-Marie Hullot [Hul79, Hul80b, Hul80a] puis Philippe Le Chenadec [Che83, Che85, Che86]. Des implantations actuelles sont fournies par des outils comme CiME [CM96, CMMU04], Waldmeister [GHLS03] ou mkbTT [mkb10].

Nous pourrions donc résoudre le problème consistant à trouver des définitions pour les fonctions de constructions `empty`, `singleton` et `union` en utilisant une fonction de normalisation par réécriture avec filtrage modulo associativité et commutativité telle qu'elle est implantée de manière optimisée dans des interpréteurs ou compilateurs pour des langages basés sur la réécriture tels que Maude [CDE+05] ou Elan [BCD+00].

Mais la réécriture avec filtrage modulo associativité et commutativité est complexe [BKN87]. De plus, du fait des traits hautement impératifs généralement utilisés par souci d'optimisation [Eke96, Mor99, KM01], la preuve de correction d'un tel programme serait extrêmement difficile. A noter cependant dans cette direction la preuve en Coq d'un algorithme fonctionnel de filtrage modulo associativité et commutativité par Évelyne Contejean [Con04]. Enfin, il n'est pas clair avec quelle efficacité une procédure générique comme celle-ci peut exploiter les particularités d'une théorie donnée et le fait que – *par construction* – tous les arguments d'une fonction de construction soient déjà en forme normale, ce qui correspond à une stratégie de réécriture à l'intérieur d'abord ("innermost").

Dans [BHW07], nous avons donc cherché à fournir, pour une classe de théories équationnelles particulières incluant notamment la théorie des groupes commutatifs, une construction *modulaire* des fonctions de constructions sous

forme de programmes fonctionnels simples. Cette construction a été implantée par Pierre Weis et moi-même dans Moca (voir Section 8.4).

Une preuve de correction a été faite en Coq [Coq10] pour la théorie des monoïdes, l'objectif étant à terme de générer la preuve de correction de manière automatique pour toute combinaison de théories de la classe considérée. A ce propos, il est intéressant de noter que, pour faire accepter par Coq les fonctions de constructions générées par Moca, il est nécessaire de fournir une preuve de terminaison qui peut être non triviale. Mon travail sur la certification des preuves de terminaison (Section 8.3) pourrait donc trouver ici une autre application.

Ce travail peut également être rapproché des travaux suivants sur les ensembles quotients et leur définition et manipulation en programmation ou en théorie des types :

- S. Thompson a introduit dans le langage de programmation Miranda [Tho86, Tho90] un mécanisme permettant de normaliser à la création toute valeur par un ensemble de "lois" ou règles de réécriture conditionnelles définies par l'utilisateur. Dans [BHW07] et dans Moca, nous nous intéressons à la génération automatique de "lois" par rapport à une spécification équationnelle donnée.
- La représentation et la manipulation d'ensembles quotient en théorie des types intensionnelle est un problème délicat qui a donné lieu à plusieurs travaux de recherche [BG95a, Hof95b, Hof95a, Bou97]. Dans [Cou01a, Cou01b], Pierre Courtieu considère le cas des ensembles quotients construits sur un type inductif. Il propose qu'à chaque type inductif, on puisse associer une fonction de normalisation qui est appelée à chaque fois qu'une valeur de ce type est analysée. Il s'agit donc là encore de représenter un ensemble quotient par un sous-ensemble de représentants pour chaque classe d'équivalence. Cependant, contrairement aux types abstraits ou privés, des termes non normaux peuvent être créés.

8 Développements logiciels

Dans cette section, je décris mes développements logiciels et, en particulier, les raisons qui m'ont amenées à les développer.

8.1 Coq-R : assistant basé sur la réécriture

Rapports : [Bla04a, Bla04b, Bla04c]

En 2003, dans le cadre du projet RNTL Averroes (Analysis and VErification for the Reliability Of Embedded Systems), j'ai développé une extension expérimentale de l'assistant à la preuve Coq [Coq10] dans laquelle les fonctions et prédicats peuvent être définis à l'aide de règles de réécriture. Les réécritures nécessaires à la vérification des preuves étaient réalisées par la bibliothèque CiME [CMMU04].

Ce prototype est toujours disponible dans l'archive SVN de Coq (tag `recriture`). Mais il n'a donné lieu à aucun autre développement car la conclusion de cette expérience a été plutôt négative. En effet, du fait que les notions de "fixpoint" et de "match" utilisés dans Coq pour définir fonctions et prédicats soient codées en dur dans le noyau, y intégrer de la réécriture sans en modifier les structures de données de base est non seulement inefficace mais ne permet pas d'en tirer tous les bénéfices possibles. Pour cela, il faudrait adapter nombre de tactiques et modifier la librairie standard, ce qui représente un travail considérable.

C'est pourquoi il me parait préférable de développer un noyau complètement nouveau dans lequel symboles de fonction et règles de réécriture sont des notions primitives, tandis que types inductifs et fonctions récursives sont des notions secondaires. Ce travail à la fois théorique et pratique a été récemment entrepris par Mathieu Boespflug sous la direction de Gilles Dowek [Boe10a, Boe10b]. Pour cela, il convient en effet de trouver des structures de données et un algorithme efficaces pour tester l'équivalence modulo la β-réduction et les règles de réécriture *a priori* arbitraires de l'utilisateur. La rapidité avec laquelle un système comme Coq peut vérifier des preuves de plus en plus grosses [GWZ00, GPWZ02, GMR+07] ou gourmandes en calcul [Gon, Gon07, GTW06] est en effet un enjeu de plus en plus important qui a stimulé et continue de stimuler de nombreux travaux et développements [GL02, Gré03, AGST10].

L'approche adoptée par Mathieu dans Dedukti est similaire à celle d'un autre logiciel, Moca [BW08], que je développe avec Pierre Weis depuis 2007. Moca est un générateur de fonctions de construction pour des types de données abstraits (ou privés [Wei03]) dont les valeurs doivent vérifier une théorie équationnelle données par des équations ou des règles entre termes constructeurs [BHW07]. Cette approche consiste à compiler les règles de réécriture vers un langage de haut niveau disposant de filtrage et d'un compilateur efficace (Haskell [PJ03] pour Dedukti, OCaml [LDF+10] pour Moca).

A noter que le développement d'un nouveau noyau n'interdit pas, bien au contraire, de réutiliser les outils déjà développés autour du noyau actuel de Coq ou d'un autre prouveur (tactiques [Del01], bibliothèques, etc.).

8.2 HOT : prouveur de terminaison

En ce qui concerne les critères de terminaison, je n'ai commencé à développer un prototype que récemment. Une première compétition de prouveurs de terminaison pour des systèmes de réécriture d'ordre supérieur a été organisée durant FLOC'10 [TC]. Elle a réuni deux outils : HORPO [Rub10] et Wanda [Kop10] sur une base de données de quelques dizaines de problèmes seulement. Je prévoie de contribuer à enrichir cette base de données et de participer à la prochaine compétition.

8.3 CoLoR : certification de preuves de terminaison

Site Internet : `http://color.inria.fr/`

Journaux : [BK11]
Rapports : [BK09]
Workshops : [BCGD+06]

Ce travail a été réalisé en collaboration avec des stagiaires de master et d'autres chercheurs européens, dont en particulier Adam Koprowski (Université technologique de Eindhoven jusqu'en 2008, Université de Radboud en 2009, et MLstate depuis).

Autres contributeurs : Sidi Ould-Biha (postdoc INRIA), Kim-Quyen Ly (stagiaire INRIA), Sorin Stratulat (Université Paul Verlaine, Metz), Johannes Waldmann (Leipzig HTWK, Germany), Julien Bureaux (stagiaire ENS Paris), Pierre-Yves Strub (postdoc INRIA), Qian Wang (stagiaire Tsinghua University, China), Lianyi Zhang (stagiaire Tsinghua University, China), Hans Zantema (Eindhoven University of Technology, The Netherlands), Jörg Endrullis (Amsterdam Vrije Universiteit, The Netherlands), Stéphane Le Roux (ENS Lyon), Léo Ducas (stagiaire ENS Paris), Solange Coupet-Grimal (Université de Provence Aix-Marseille I), William Delobel (Université de Provence Aix-Marseille I), Sébastien Hinderer (stagiaire UHP Nancy).

Depuis 2004, mes travaux de développements logiciels se sont donc surtout concentrés sur la certification de preuves de terminaison pour les systèmes de réécriture du premier ordre.

Les raisons en sont les suivantes. Comme indiqué précédemment, étendre la version actuelle de Coq avec de la réécriture ne permet pas de bénéficier de tous les avantages que peut procurer la réécriture. Pour cela, il est préférable de développer un nouveau noyau d'assistant basé dès le départ sur la réécriture.

Mais cela implique que certaines propriétés essentielles pour la cohérence logique du système ou la décidabilité de la correction des preuves ne sont plus garanties par construction : non ambiguïté des définitions (confluence), complétude des définitions (aucun cas n'est oublié) et terminaison des définitions. Et, bien sûr, ces propriétés sont indécidables. Il existe cependant des critères pour celles-ci (voir [Kou85, Coq92, SP03, WCC08] pour la complétude) mais, vu leur complexité, la confiance que l'on peut avoir dans un tel système logique en est inévitablement diminuée.

Pour éviter cela, il est nécessaire de certifier les résultats des outils implantant ces critères. Pour cela, plusieurs approches sont possibles :

1. Vérifier la correction du code de l'outil avec un logiciel d'analyse statique et de génération d'obligations de preuve comme Why [FM07]. Cette approche semble bien adaptée pour vérifier des propriétés universelles sur des entiers ou des tableaux utilisant des prédicats qui peuvent être facilement axiomatisés dans un fragment simple de la logique du premier ordre (*e.g.* Prolog), ce qui risque de ne pas être le cas des propriétés pour la terminaison des systèmes de

réécriture. Par ailleurs, Why et les prouveurs qu'il peut utiliser pour décharger les obligations de preuve les plus simples ne sont pas eux-mêmes certifiés, quoique des travaux récents vont dans cette direction [BDD07, CCKL07].

2. Coder un algorithme implantant le critère dans un assistant, prouver sa correction et en extraire le code [PM89, Let04, Haf09, HN10]. L'inconvénient principal de cette approche est l'efficacité car les assistants actuels ne permettent pas d'utiliser des constructions impératives et, du fait que toute fonction doit être accompagnée d'une preuve de terminaison, limitent *de facto* le type de fonctions pouvant être définies (les fonctions dont la terminaison est trop compliquée à montrer seront généralement évitées).
3. Définir une notion de certificat que l'outil peut fournir en même temps que le résultat et qui permet de vérifier sa correction. Dans cette approche, seul le certificat compte. L'outil peut mettre en oeuvre des heuristiques compliquées, faire appel à des outils externes non certifiés, et ainsi avoir de forte chance d'être incorrect. Mais, si le certificat fourni est correct, alors le résultat fourni est correct. Mais cela demande de certifier le programme vérifiant la correction des certificats, ce qui peut se faire par une des deux premières méthodes, et de préférence la seconde dans la mesure où vérifier la correction d'une solution est généralement bien plus simple que de rechercher une solution. Cette approche a deux avantages importants sur les précédentes. D'une part, un outil n'a pas besoin d'être prouvé à chaque changement car seuls ses résultats sont vérifiés. D'autre part, un vérificateur de certificats peut servir à différents outils si la notion de certificats est suffisamment générale.

C'est ainsi que j'ai développé avec l'aide des personnes précédemment citées une bibliothèque Coq [Coq10] de définitions et théorèmes (récents) sur la théorie de la réécriture et en particulier sur les relations de réécriture bien-fondées.

Cette bibliothèque compte désormais environ 70 000 lignes de code Coq, 1500 définitions et 3500 théorèmes, la plupart de ces théorèmes étant des théorèmes très simples servant de règles d'introduction ou d'élimination d'un objet, ou permettant de faire de la réécriture lorsque ce théorème est une égalité ou une équivalence propositionnelle.

On y trouve en particulier des définitions et résultats sur :

- **structures mathématiques :** relations/graphes, semi-anneaux et anneaux ordonnés ou non.
- **structures de données :** listes, vecteurs, matrices, multi-ensembles finis et polynômes à plusieurs variables.
- **structure de termes :** mots, termes algébriques standards avec symboles de fonction d'arité fixe, termes variadiques (termes algébriques avec symboles de fonction d'arité variable), λ-termes simplement typés.
- **techniques de transformation :** conversion de mots en termes algébriques et de termes algébriques en termes variadiques, transformation "miroir" pour les systèmes de réécriture de mots, filtrage des arguments, paires de dépendance, décomposition du graphe de dépendance par unification [AG00] (voir Section 7.2.5), étiquetage sémantique [Zan95] (voir Sec-

tion 7.2.3), étiquetage par la racine et clôture par contextes plats [SM08], et règles utilisables [GTSKF03].
- **techniques de (non-)terminaison :** extensions lexicographique et multi-ensemble, interprétations polynomiales et matricielles [EWZ06], ordre récursif sur les chemins au premier ordre [Der82] et à l'ordre supérieur [JR99], critère sous-terme [HM07], et certification de boucles.

Parallèlement au développement de cette bibliothèque, Adam Koprowski et moi avons développé une notion de certificat de terminaison et le programme Rainbow qui, à partir d'un problème de terminaison et d'un certificat pour ce problème, génère un script Coq contenant une formalisation du problème et de la preuve de terminaison correspondant à ce certificat. Un appel à Coq permet ensuite de vérifier la correction de ce script et donc du certificat.

D'autres équipes de recherche se sont également intéressées à la certification des preuves de terminaison :
- **A3PAT** [11] avec Évelyne Contejean du LRI (Orsay) d'une part, Xavier Urbain, Julien Forest, Pierre Courtieu et Olivier Pons du CNAM (Évry et Paris) d'autre part, Évelyne Contejean et Xavier Urbain développant également le prouveur automatique de terminaison CiME [ECU09]. L'équipe A3PAT utilise une approche similaire à la nôtre, à savoir la génération de scripts Coq, mais ils utilisent leur propre bibliothèque Coq sur la réécriture et la terminaison dénommée Coccinelle. Cependant, pour un certain nombre de notions, ils préfèrent utiliser un encodage direct en Coq ("shallow embedding") plutôt que de définir ces notions préalablement ("deep embedding"). Dans certains cas, cela permet de limiter certains calculs dans Coq qui n'est pas aussi efficace qu'un langage de programmation standard. Mais cela a aussi des inconvénients. Premièrement, les scripts ainsi générés sont plus longs et plus difficiles à comprendre. Cela rend plus difficile le débogage et la maintenance du générateur de scripts. Deuxièmement, outre que de disposer d'une formalisation de ces notions est intéressant en soit, cela ne permet pas d'extraire [PM89, Let02, Let04] de la bibliothèque Coq utilisée un vérificateur de certificats, correct par construction, qui puisse ensuite être compilé et exécuté indépendamment, solution retenue par l'équipe suivante et sur laquelle nous travaillons également.
- **CeTA** [12] avec Christian Sternagel et René Thiemann (Université d'Innsbruck, Autriche), René Thiemann étant un des développeurs du prouveur automatique de terminaison AProVE [13] [GSKT06] et Christian Sternagel un des développeurs du prouveur automatique de terminaison TTT2 [HM07]. Là encore, une bibliothèque dénommée IsaFoR est à la base de ce vérificateur de certificats, mais elle utilise l'assistant à la preuve Isabelle/HOL [NPW02] qui est basé sur la théorie des types simples clas-

11. `http://a3pat.ensiie.fr/`
12. `http://cl-informatik.uibk.ac.at/software/ceta/`
13. `http://aprove.informatik.rwth-aachen.de/`

sique avec axiome du choix [Chu40], alors que Coq est basé sur le calcul des constructions inductives [PM93, Wer94], c'est-à-dire une logique constructive sans axiome *a priori*. Une autre différence importante est que CeTA est entièrement extrait de IsaFoR [BN02, Haf09, HN10], ce qui rend la vérification de certificats bien plus rapide. Cette extraction a été commencée avec CoLoR également. Pour la mener à bien, il convient de définir et certifier Rainbow lui-même en Coq. Ce travail va faire l'objet de la thèse de Kim-Quyen Ly qui va démarrer en novembre 2010.

A3PAT et l'équipe développant AProVE ont chacun développé un format pour les certificats de terminaison. Du coup, en 2009, un format commun, dénommé CPF [Cer10], a été élaboré pour être utilisé dans la compétition internationale de terminaison [TC] qui, depuis 2007, organise une compétition pour les prouveurs certifiés.

En 2007 et 2008, cette compétition évaluait des couples prouveur automatique - vérificateur de certificats sur le nombre de preuves de terminaison trouvées et certifiées par rapport à l'ensemble de la base de données de problèmes de terminaison de la compétition [TPD]. En 2007, c'est le couple TPA-Rainbow (TPA étant développé par Adam Koprowski) qui est arrivé premier devant CiME-A3PAT et TTT2-Rainbow. En 2008, c'est le couple AProVE-Rainbow qui est arrivé premier devant les couples CiME-A3PAT, Matchbox-Rainbow et AProVE-A3PAT sur les systèmes de réécriture de termes, et c'est le couple Matchbox-Rainbow qui est arrivé premier devant les couples AProVE-Rainbow et AProVE-A3PAT sur les systèmes de réécriture de mots.

En 2009, le format CPF a été introduit et le déroulement de la compétition modifié. D'une part, la compétition ne porte désormais que sur un sous-ensemble tiré au hasard de la base de données. D'autre part, la compétition est organisée en deux temps. Premièrement, les prouveurs génèrent les certificats. Deuxièmement, les vérificateurs de certificats sont appelés sur l'ensemble des certificats générés dans la première phase. Le meilleur vérificateur de certificats a ainsi été CeTA devant Rainbow et A3PAT, en particulier du fait de son utilisation dans les certificats du critère sous-terme et de la technique dite des règles utilisables non supportés à l'époque par Rainbow et A3PAT. Rainbow n'a pas participé à la compétition 2010 organisée 6 mois seulement après celle de 2009. L'ensemble des résultats est disponible sur la page web de CoLoR ou sur celle de la compétition.

Autres publications liées à CoLoR :

Thèses : [Kop08]
Journaux : [CGD06b, Kop09, BK11]
Conférences : [CGD06c, Kop06a, Rou07, KW08, KZ08]
Workshops : [dKKvR04, BCGD$^+$06, KZ07]
Rapports : [CGD06a, Kop06b, BK09]
Masters : [dK03, Hin04, Kop04]

Autres publications sur la certification de preuves de terminaison :
A3PAT : [Con04, Hub04, Con07, CCF+07, CFU08a, CFU08b, CPU+10]
CeTA : [TS09, ZSHM10, STWZ09, ST10]

8.4 Moca : générateur de fonctions de constructions

Site Internet : `http://moca.inria.fr/`

Conférences : [BHW07]

Ces développements ont été faits en collaboration avec Pierre Weis.

Moca est un outil permettant de spécifier des types quotients à l'aide d'équations ou de règles de réécriture sur les constructeurs du type de données sous-jacent, et de générer à partir d'une telle spécification un ensemble de fonctions de construction permettant de définir un tel type dans OCaml [LDF+10]. Pour cela, Moca utilise le système de modules d'OCaml [Que84] et le mécanisme des types privés [Wei03] qui a l'avantage sur le mécanisme des types abstraits de permettre la définition de fonctions par filtrage. Moca garantit que chaque classe d'équivalence n'est habitée que par une seule valeur. Il offre également la possibilité de faire du partage maximale, y compris sur des types polymorphes, de manière similaire à [CF06]. Voir Section 7.3 pour plus de détails.

8.5 SimSoC-Cert : simulateur ARM certifié

Site Internet : `http://formes.asia/cms/software/simsoc-cert`

Conférences : [BHJ+11]
Rapports : [BHJ+10]

Ces développements ont été faits en collaboration avec Claude Helmstetter, Jean-François Monin et Xiaomu Shi qui a commencé une thèse avec nous sur ce sujet en novembre 2009.

SimSoC[14] est un simulateur de systèmes sur puce développé par Vania Joloboff et Claude Helmstetter depuis 2007 [JH08, HJX09]. Depuis novembre 2009, nous nous intéressons à la certification en Coq [Coq10] de ce simulateur pour le processeur ARM version 6 [ARM05].

Le manuel de l'ARM version 6 [ARM05] est un document de 1138 pages dont environ 600 sont dédiées à l'encodage et la sémantique de ses 220 instructions. Pour chaque instruction, le manuel fournit son encodage sur 16 ou 32 bits sous la forme d'un tableau, sa syntaxe assembleur, sa sémantique sous la forme de pseudo-code C, et des informations supplémentaires (exceptions notamment !) en langue naturelle. Les informations formelles (encodage, pseudo-code) peuvent assez facilement être extraites pour ensuite être traitées automatiquement. En

14. `http://formes.asia/cms/software/simsoc`

particulier, il est possible de générer automatiquement du code pour un assistant de preuve ou un langage de programmation pour décoder et exécuter les instructions et obtenir ainsi un simulateur. Cela a plusieurs avantages :

- Cela limite le nombre d'erreurs qu'une formalisation à la main risquerait d'introduire.
- De nombreux éléments peuvent être réutilisés pour passer à une nouvelle version de l'ARM ou à un autre type de processeur.

Nous avons alors développé plusieurs générateurs de code : un générateur de code Coq et deux générateurs de code C (un non-optimisé et un optimisé) qui, à partir des informations formelles extraites de la documentation, permettent de générer de véritables simulateurs du processeur ARM.

Le simulateur Coq ainsi obtenu n'est bien sûr pas destiné à être utilisé en pratique car il est naturellement beaucoup trop lent. Il est d'ailleurs nécessaire d'en extraire [Let02, Let04] un simulateur OCaml qu'on puisse compiler et exécuter pour pouvoir déboguer plus facilement le code Coq généré en utilisant le débogueur OCaml [LDF+10]. Par contre, il fournit une description mathématique simple du comportement que doit suivre le processeur qu'il est possible d'utiliser pour vérifier la correction de simulateurs écrits dans des langages plus standards comme le C. Nous prévoyons ainsi de certifier en Coq, si possible automatiquement, la correction du code C également généré à partir du manuel en s'appuyant sur la sémantique du C formalisée dans le projet CompCert [BL09].

Le simulateur C optimisé ainsi généré a été récemment intégré à SimSoC qui est écrit en C++ et utilise les bibliothèques SystemC [IEE06] et TLM [Ghe05]. Ces performances sont similaires à la version précédente écrite à la main et permet par exemple de simuler l'exécution du système d'exploitation Linux. L'étape suivante sera bien sûr de chercher à certifier les optimisations permettant d'accélérer de manière importante la simulation. Certaines reposent sur des techniques de spécialisation de code. D'autres sur la mise en cache du décodage des instructions. D'autres, plus avancées encore, reposent sur des techniques de décompilation et recompilation à la volée sur la machine hôte (processeur x86).

Quand nous avons commencé ce projet en 2009, il n'existait pas de formalisation récente de l'ARM. Seule une formalisation de l'ARMv3 et certains aspects de l'ARMv4 avaient été formalisés par Anthony Fox dans HOL [Fox01]. Mais, d'une part, ces versions de ARM sont obsolètes et, d'autre part, il s'agit d'un assistant différent de Coq (à noter cependant le travail récent de Chantal Keller et Benjamin Werner sur la traduction de HOL-Light [Har10] en Coq [KW10]). Cependant, en 2010, Anthony Fox et Magnus Myreen ont publié un article [FM10] présentant une formalisation de l'ARMv7 dans HOL [Gor88, HOL10]. Il s'agit donc d'une version plus récente de l'ARM dont la formalisation a été écrite à la main alors que nous cherchons à la générer automatiquement depuis la documentation. Cependant, ces deux chercheurs ont fait de nombreux tests afin de vérifier la correction de leur formalisation. Il sera donc intéressant d'essayer de comparer les deux formalisations et de réutiliser leurs jeux de tests afin de vérifier la correction de nos simulateurs.

9 Perspectives

Dans cette section, j'indique certains problèmes ou pistes de recherche en relation avec mes travaux ou mes développements logiciels qui me paraissent intéressants d'étudier ou de suivre.

9.1 Calcul des constructions modulo

- **Hypothèses non équationnelles.** Dans notre travail sur l'extension de l'équivalence sur les types (Section 7.1.5), nous n'avons considéré jusqu'à maintenant que les hypothèses équationnelles. Mais il ne devrait pas poser de problèmes d'extraire de l'environnement de typage bien d'autres propositions. On peut par exemple prendre en compte des hypothèses non équationnelles comme $x \leq 2$. En effet, si on peut extraire $x \leq 2$ d'une part, et $x \geq 2$ d'autre part, alors il est naturel de considérer x comme étant équivalent à 2. On peut également extraire des hypothèses non équationnelles comme $x \leq 2 \wedge x \geq 2$ ou $x \leq 2 \vee x \geq 2$.
- **Réécriture au niveau type.** Les conditions de terminaison actuelles pour la réécriture au niveau type interdisent d'avoir des paires critiques entre les règles de réécriture au niveau type et les autres règles. En fait, ces conditions ne sont pas vraiment nécessaires à la terminaison proprement dite mais permettent, d'une part, de définir l'interprétation des types de laquelle se déduit la terminaison et, d'autre part, de s'assurer que cette interprétation est stable par réduction. Or, lorsqu'un prédicat est complètement défini par un ensemble de règles, rajouter une nouvelle règle qui n'est qu'une conséquence inductive des autres règles ne devrait pas poser de problème. Il serait intéressant de vérifier le bien fondé de cette idée en s'appuyant sur le récent travail de Daria Walukiewicz et Jacek Chrząszcz [WCC10].
- **Univers.** Le calcul des constructions étendu (ECC) de Zhaohui Luo [Luo90] est une extension du calcul des constructions avec une hiérarchie cumulative d'univers prédicatifs $\Box = \Box_0 : \Box_1 : \ldots$ au-dessus de $\star$ qui sert de base au système Coq. On peut naturellement chercher à étendre la clôture de calculabilité [Bla05b, Bla10] ou le calcul des constructions modulo théorie [Str08, Str10b] à ce système. Ainsi, Qian Wang, qui a débuté une thèse avec Jean-Pierre Jouannaud en septembre 2009, a commencé à étudier l'extension du travail de Pierre-Yves Strub à ECC [BJSW11]. Une des difficulté réside dans le fait que la cumulativité introduit une notion de sous-typage qui rend l'étude méta-théorique du système bien plus délicate. Cela pourrait être simplifié en s'appuyant sur de récents travaux [BJP10, Wes11].
- **Modules et sous-typage.** Les modules sont un outil important dans les langages de programmation et les assistants de preuve pour organiser de gros développements, factoriser du code en utilisant des modules paramétrés ou foncteurs, permettre d'abstraire certains détails d'implantation et de compiler les modules séparément [Mor73, LZ74, Que84].

Pour Coq, voir les travaux de Judicaël Courant [Cou98, Cou07] et Jacek Chrząszcz [Chr03a, Chr03b, Chr04]. Un module pouvant être vu comme une structure ou un type inductif à un constructeur, on peut se demander dans quelle mesure le calcul des constructions algébriques peut servir à décrire un calcul de modules, en particulier comme objets de première classe. Premièrement, un calcul de module utilise une notion de sous-typage. Il convient donc d'introduire une telle notion en s'appuyant par exemple sur les travaux de Gang Chen [Che98] (déjà utilisés pour traiter le sous-typage lié aux annotations de taille en Section 7.2.3). Deuxièmement, un module qui contient un type ou un prédicat correspond alors à un gros type inductif pour lequel l'élimination forte n'est généralement pas autorisée [Coq86] (voir Section 7.1.2). La clôture de calculabilité peut-elle malgré tout être étendue à cette classe particulière de gros types inductifs [Cou02] ?

– **Modules et réécriture.** Je trouve insatisfaisante la manière dont est actuellement envisagée la combinaison des notions de module et de réécriture ou, plus généralement, de sémantique opérationnelle ou d'équivalence [Chr03a, Chr03b, Chr04]. Par exemple, lorsqu'on définit dans Coq [Coq10] un type de module contenant un type inductif Coq, non seulement le type, ses constructeurs et le principe d'induction structurelle ou récurseur généré automatiquement par Coq sont exportés, mais aussi la sémantique opérationnelle de ce récurseur (ι-réduction). De plus, un foncteur prenant en argument un tel type de module ne peut être appliqué qu'à un module contenant exactement le même type inductif. Pour être plus général, il conviendrait dans le type de module de ne pas déclarer un type inductif mais des symboles pour le type, ses constructeurs, le principe d'induction associé et les équations satisfaites par celui-ci. Mais, alors, il n'est plus possible de définir des fonctions par induction sur ce type...

  Un autre exemple est donnée par la théorie des groupes. Lorsqu'on définit un foncteur qui à tout groupe associe un ensemble de théorèmes de la théorie des groupes, l'équivalence au niveau type n'est pas augmentée par l'équivalence (pourtant décidable) engendrée par les équations de la théorie des groupes. Ainsi, toute égalité relevant de la théorie des groupes doit être justifiée et, même s'il est possible de définir une tactique pour prouver ce type d'égalités, l'intervention de l'utilisateur est nécessaire et engendre des termes de preuve éventuellement gros.

  Dans ces deux cas, il est pourtant possible au système, avec l'aide d'un outil de complétion à la Knuth-Bendix [KB70] guidé par l'utilisateur, d'étendre l'équivalence sur les types en rajoutant des règles de réduction ou une procédure pour décider la théorie équationnelle engendrée par les propositions extraites de l'environnement de typage, y compris donc les équations universellement quantifiées comme celles de la théorie des groupes. On voit ainsi se dessiner les contours d'une théorie unifiant les travaux sur les modules, la réécriture, l'intégration de procédures de décision, les types quotients, ... qu'on pourrait par exemple appeler le calcul des constructions équationnelles, sous-ensemble décidable du calcul des

constructions ouvert de Mark-Oliver Stehr [Ste02, Ste05a].

9.2 Confluence

- **Préservation du typage par la β-réduction** dans le calcul des constructions algébriques avec réécriture au niveau type en absence de preuve de confluence. Dans [Bla05b], je donne une condition sur les règles au niveau type qui exclut beaucoup trop de systèmes (voir Section 7.1.1). Les techniques utilisées dans l'étude de la modularité de la confluence [Toy87, KMTdV94, Jou06, JT08] peuvent-elles servir à améliorer ces conditions ? Autrement, la préservation du typage par β-réduction pourrait-elle être déduite de l'équivalence avec un système où la réécriture est typée (*i.e.* avec jugements d'égalité) [Gog94, BM96] ?
- **Préservation de la confluence par curryfication** pour la réécriture conditionnelle (d'ordre supérieur). Dans [Rib07a], Colin n'a réussi à étendre qu'aux systèmes de réécriture conditionnelle non-effondrants le résultat de Stefan Kahrs pour la récriture inconditionnelle du premier ordre, à savoir que si un système conflue sur les termes algébriques alors sa version curryfiée conflue également sur les termes applicatifs [Kah95]. La difficulté vient du fait que la décurryfication ne commute pas avec la mise en contexte.

9.3 Terminaison

- **Réécriture du premier ordre polymorphe.** Depuis les travaux de Val Breazu-Tannen et Jean Gallier [BTG89, BTG91], Mitsuhiro Okada [Oka89] et Dan Dougherty [Dou91, Dou92], il est connu que la terminaison est préservée lorsqu'on combine un λ-calcul β-terminant avec un système de réécriture du premier ordre standard, *i.e.* dont toutes les variables sont interprétées par des objets. Malheureusement, et aussi surprenant que cela puisse paraître, aucun résultat connu ne semble s'appliquer directement si l'on paramètre un tel système de réécriture avec des variables de types. C'est le cas par exemple si l'on passe de :

$$\mathsf{app}\,\ell\,(\mathsf{cons}\,x\,\ell')\to\mathsf{cons}\,x\,(\mathsf{app}\,\ell\,\ell')$$

où $\mathsf{app}:\mathbb{L}\Rightarrow\mathbb{L}\Rightarrow\mathbb{L}$ est la concaténation sur les listes *d'entier naturels* à :

$$\mathsf{app}\,A\,\ell\,(\mathsf{cons}\,A'\,x\,\ell')\to\mathsf{cons}\,A\,x\,(\mathsf{app}\,A\,\ell\,\ell')$$

où $\mathsf{app}:\forall A:\star.\mathbb{L}A\Rightarrow\mathbb{L}A\Rightarrow\mathbb{L}A$ est la concaténation sur les listes *polymorphes*.
- **Inférence des annotations de taille.** Dans [Bla10, BR06], nous avons développé une procédure permettant de vérifier la terminaison de fonctions définies par un ensemble de règles de réécriture conditionnelles qui suppose connus les types annotés des fonctions et les mesures permettant de montrer la décroissance des arguments. Développer des heuristiques

pour trouver ces types annotés et ces mesures permettraient de transformer cette procédure de vérification de la terminaison en une procédure de recherche de preuve de terminaison. Pour ce qui est de l'inférence des types annotés, on peut s'appuyer sur les travaux de Wei-Ngan Chin and Siau-Cheng Khoo [CK01a].

- **Annotations de taille et types intersection.** Dans une série d'articles [Ber05a, Ber05b, Spi05, CS06, CS07, Abe07, Ber08], Ulrich Berger d'une part, Arnaud Spiwack et Thierry Coquand d'autre part, ont développé des critères sémantiques de terminaison pour la réécriture basés sur l'utilisation de types intersection [Sal78, CD78, Hin82, BCDC83, vB92, vB95]. Il serait intéressant de comparer leur approche avec, d'une part, celle antérieure de Steffen van Bakel et Maribel Fernández [vB93, vBF97] et, d'autre part, avec celle basée sur l'utilisation d'annotations de taille (voir Section 7.2.3).
- **Paires de dépendance à l'ordre supérieur.** Comme je l'ai mentionné en Section 7.2.5, les versions actuelles du théorème des paires de dépendance à l'ordre supérieur ne traitent pas de manière satisfaisante les variables liées et les types. Cela est-il améliorable ? Les travaux de Neil D. Jones sur l'application du "size-change principle" (SCP) de Chin Soon Lee, Neil D. Jones et Amir M. Ben-Amram [LJBA01] au λ-calcul [JB04, SJ05, JB08], ceux de Jürgen Giesl montrant le lien entre SCP et paires de dépendance [TG03, TG05] et comment la technique des paires de dépendance peut être utilisée pour montrer la terminaison de programmes Haskell [GSSKT06, GRSK$^+$10], et enfin l'application par David Wahlstedt de SCP au λ-calcul avec types dépendants et réécriture au niveau type [Wah07], semblent de ce point de vue très intéressants.
- **HORPO avec ordre sur les types.** Dans quelle mesure HORPO peut être étendu à des systèmes de types plus riches que les types simples. Daria Walukiewicz a déjà étendu la version de HORPO de LICS'99 au calcul des constructions inductives sans élimination forte [WC03a]. Cependant, les versions plus récentes de HORPO [BJR08] utilisent un ordre sur les types, ce qui est problématique. L'approche que j'ai développée dans [Bla06b] pourrait être utile. Jianqi Li, en postdoc avec Jean-Pierre Jouannaud a commencé a travaillé sur cette question [JL10].
- **Candidats de réductibilité.** Colin Riba a étudié dans [Rib07a, Rib07b, Rib07c, Rib08, Rib09] les propriétés par rapport à l'union des ensembles saturés de Tait [Tai67], des candidats de réductibilité [GLT88] et des ensembles bi-orthogonaux de Girard [Gir87]. Denis Cousineau a étudié la complétude des candidats de réductibilité pour la terminaison [Cou09]. Enfin, dans [Dow06, Dow09], Gilles Dowek introduit la notion d'algèbre de valeurs de vérité et montre que les candidats de réductibilité forment une telle algèbre. Ces travaux récents montrent qu'il y a encore de nombreuses choses à dire à propos des ensembles de termes tels que les candidats de réductibilité et comment les adapter à la terminaison de la réécriture.
- **Application au ρ-calcul et à la super déduction.** La super déduction [Wac05, BHK07] est une extension conservatrice de la déduction

naturelle introduite par Benjamin Wack en 2005 consistant à remplacer les axiomes par de nouvelles règles de déduction [Kle52]. Les preuves de la super déduction et l'élimination des coupures peuvent se représenter dans le ρ-calcul introduit en 1998 par Horatiu Cirstea et Claude Kirchner [CK99, Cir00, CK01b] qui est proche du λ-calcul avec patterns introduit par Simon Peyton-Jones en 1987 puis étudié notamment par Vincent van Oostrom [PJ87, vO90, KvOdV08]. Étant donné que le ρ-calcul peut être encodé sous forme de système de réduction combinatoire [BK07], nous pouvons cherché à appliquer les critères de terminaison développés pour les systèmes de réduction combinatoire au ρ-calcul et donc à l'élimination des coupures en super déduction.

9.4 Complexité

– **Annotations de taille et complexité.** Les types avec annotations de taille permettent de déduire des informations sur la taille des valeurs dénotées par des termes dont les symboles de fonctions sont définis par des règles de réécriture. Il est alors assez naturel de se demander dans quelle mesure il est également possible de déduire des informations sur la complexité de ces fonctions. Cela pourrait fournir une procédure de vérification, voire d'inférence, de la complexité. Pour cela, on devrait pouvoir s'appuyer sur les travaux de Jean-Yves Marion et ses co-auteurs [Mar03] ainsi que des travaux plus récents de Georg Moser et ses co-auteurs [AM08, AM10]. J'ai d'ailleurs encadré un stage exploratoire sur ce sujet en 2009 (Antoine Taveneaux, ENS Lyon). A noter également qu'il existe une compétition de prouveurs de complexité [TC] dont les résultats ne font pas encore l'objet d'une certification...

9.5 Certification de preuves de terminaison

En relation avec les travaux décrits en Section 8.3.

– **Formaliser de nouveaux critères de terminaison.** L'idéal serait bien sûr que les auteurs de ces critères les formalisent eux-mêmes...
– **Expérimenter l'utilisation des classes de types** de Matthieu Sozeau [Soz08a, Soz08b, Soz09] pour éviter d'avoir à définir nombre de modules de manière explicite.
– **Définir et raisonner sur des séquences infinies de réécriture.** Les critères qui ne semblent pas avoir de preuves constructives nécessitent de définir et raisonner sur des séquences infinies de réécriture. C'est ce que Sidi Ould-Biha et moi-même avons dû faire récemment pour formaliser en Coq le critère sous-terme et la technique des règles utilisables [HM07]. Une bibliothèque générale sur ce type de séquences avec des fonctions de construction, des théorèmes et des tactiques faciliteraient ce type de preuve. Il serait également intéressant de comparer la formalisation faite en Coq avec celle faite dans Isabelle/HOL [ST10].

9.6 Fonctions de construction pour types quotients

En relation avec les travaux décrits en Section 7.3 et 8.4.

- **Certifier la correction des fonctions générées** par Moca.
- **Générer les invariants satisfaits par un type à relations.** Afin de pouvoir certifier des programmes utilisant un type à relations définis avec Moca avec des outils comme Who [KF09] ou Focalize [HPWD09] ou en Coq directement, il convient de générer les propriétés satisfaites par les valeurs de ce type.
- **Complétude des définitions.** Afin de détecter une source possible d'erreur, le compilateur OCaml émet une alarme si un cas a été oublié dans la définition d'une fonction par filtrage sur un type inductif. Avec un type privé général, de telles alarmes peuvent être incorrectes. Dans le cas d'un type à relations définis avec Moca, il est possible de faire une analyse plus fine des définitions afin d'éviter de telles alarmes [Kou85].
- **Liens avec la réécriture avec filtrage modulo AC.** Dans Moca, la commutativité est d'une certaine manière traitée en considérant un ensemble infini de règles conditionnelles basées sur un ordre total. Il serait intéressant d'étudier les liens entre ce type de réécriture conditionnelle et la réécriture avec filtrage modulo AC [PS81], notamment en terme de complexité pour le calcul de formes normales.
- **Liens avec la complétion de Knuth-Bendix.** Si on voit les fonctions OCaml générées par Moca comme un système de réécriture avec priorités [BBKW89], on peut se demander s'il ne serait pas possible de déduire ces fonctions d'une procédure de complétion à la Knuth-Bendix [KB70].

9.7 Simulateur de processeur certifié

En relation avec les travaux décrits en Section 8.5.

- **Certifier en Coq la correction des fonctions C générées**, si possible automatiquement, en s'appuyant sur la sémantique du C formalisée dans le projet CompCert [BL09]. On pourra ensuite chercher à certifier les optimisations utilisées dans SimSoC (évaluations partielles, mise en cache, décompilation et recompilation sur le processeur hôte à la volée).
- **Certifier le code simulant l'unité de gestion de la mémoire** (MMU) qui est indispensable au démarrage de Linux par exemple.
- **Optimiser le décodeur.** L'analyse du codage des instructions doit permettre d'optimiser la fonction de décodage.
- **Application à d'autres types de processeurs** comme PowerPC, MIPS, SuperH ou x86.

10 Conclusion

J'ai présenté dans ce document une synthèse de mes activités de recherche et développement depuis 2002.

D'un point de vue pratique, mes travaux cherchent à rendre les assistants à la démonstration basés sur le λ-calcul plus puissants et plus facilement accessibles aux non-spécialistes, conditions indispensables pour passer à l'échelle et permettre leur adoption par une plus large communauté : mathématiciens souhaitant vérifier la preuve d'un théorème ou informaticiens souhaitant vérifier la correction d'un programme. Et cela soulève de nombreux problèmes théoriques tout à fait passionnants.

Je me suis intéressé en particulier à la réécriture qui permet de définir fonctions et prédicats de manière simple, et d'utiliser certaines égalités comme règles de simplification. Étendre l'équivalence au niveau type/proposition avec de la réécriture ou des procédures de décision permet de moins perdre du temps dans des détails techniques sans importance. Enfin, utiliser des outils externes pour vérifier la terminaison ou d'autres propriétés importantes ne met pas en péril la sécurité d'un système si ces outils sont capables de fournir des certificats dont la correction peut être vérifiée formellement.

Mes travaux plus récents sur la génération de fonctions de constructions ou le développement d'un simulateur ARM certifié s'inscrivent également dans cette logique visant à fournir des outils certifiés d'aide au développement de programmes et de preuves, champ d'étude très vivant qui compte plusieurs succès importants ces dernières années : une preuve formelle constructive du théorème fondamental de l'algèbre [GWZ00], une plate-forme JavaCard certifiée [BCDdS02], une preuve formelle du théorème des quatre couleurs [Gon], un compilateur C certifié [Ler09], un système d'exploitation certifié [KEH+09], la preuve formelle en cours [15] de la conjecture de Kepler [Hal05, HHM+05], ...

11 Co-auteurs

- Thérèse Hardin, professeur à l'Université Paris VI [BHW07]
- Claude Helmstetter, docteur de l'Institut National Polytechnique de Grenoble [BHJ+10, BHJ+11]
- Jean-Pierre Jouannaud, directeur de recherche INRIA, anciennement professeur à l'Université Paris-Sud et à l'Ecole Polytechnique [BJO99, BJO02, BJR08, BJR06, BJR07, BJS07, BJS08]
- Claude Kirchner, directeur de recherche INRIA [BKR06, BKR10]
- Adam Koprowski, docteur de l'Université de Technologie de Eindhoven (Hollande), ingénieur de recherche à MLState à Paris [BCGD+06, BK09, BK11]
- Keiichirou Kusakari, professeur associé à l'Université de Nagoya (Japon) [KISB09, SKB10, SKB11]
- Vania Joloboff, directeur de recherche INRIA [BHJ+10, BHJ+11]
- Jean-François Monin, professeur à l'Université Joseph Fourier (Grenoble) [BHJ+10, BHJ+11]
- Mitsuhiro Okada, professeur à l'Université Keio à Tokyo (Japon) [BJO99, BJO02]

15. `http://code.google.com/p/flyspeck/`

- Colin Riba, maître de conférence à l'ENS Lyon (chaire CNRS) [BKR06, BKR10, BR06]
- Cody Roux, postdoc INRIA [BR09]
- Albert Rubio, professeur à l'Université Polytechnique de Catalogne à Barcelone (Espagne) [BJR08, BJR06, BJR07]
- Masahiko Sakai, professeur à l'Université de Nagoya (Japon) [KISB09, SKB10]
- Pierre-Yves Strub, docteur de l'Ecole Polytechnique, ingénieur de recherche au laboratoire commun INRIA-Microsoft [BJS07, BJS08]
- Pierre Weis, directeur de recherche INRIA à Rocquencourt [BHW07]

Références

[Abe02] A. Abel. Termination checking with types. Technical Report 0201, Ludwig Maximilians Universität, München, Germany, 2002.

[Abe03] A. Abel. Termination and productivity checking with continuous types. In *Proceedings of the 6th International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science 2701, 2003.

[Abe04] A. Abel. Termination checking with types. *Theoretical Informatics and Applications*, 38(4) :277–319, 2004.

[Abe07] A. Abel. Syntactical normalization for intersection types with term rewriting rules. In *Proceedings of the 4th International Workshop on Higher-Order Rewriting*, 2007.

[Abe08] A. Abel. Semi-continuous sized types and termination. *Logical Methods in Computer Science*, 4(2) :1–33, 2008.

[Ack54] W. Ackermann. *Solvable Cases of the Decision Problem.* North-Holland, 1954.

[AG97] T. Arts and J. Giesl. Automatically proving termination where simplification orderings fail. In *Proceedings of the 7th International Joint Conference on Theory and Practice of Software Development*, Lecture Notes in Computer Science 1214, 1997.

[AG00] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236 :133–178, 2000.

[AGST10] M. Armand, B. Grégoire, A. Spiwack, and L. Théry. Introducting machine integers and arrays to type theory and its application to sat verification. In *Proceedings of the International Conference on Interactive Theorem Proving*, Lecture Notes in Computer Science 6172, 2010.

[ALS94] J. Avenhaus and C. Loría-Sáenz. Higher order conditional rewriting and narrowing. In *Proceedings of the 1st International Conference on Constraints in Computational Logics*, Lecture Notes in Computer Science 845, 1994.

[AM08] M. Avanzini and G. Moser. Complexity analysis by rewriting. In *Proceedings of the 9th Fuji International Symposium on Functional and Logic Programming*, Lecture Notes in Computer Science 4989, 2008.

[AM10] M. Avanzini and G. Moser. Closing the gap between runtime complexity and polytime computability. In *Proceedings of the 21st International Conference on Rewriting Techniques and Applications*, Leibniz International Proceedings in Informatics 6, 2010.

[ARM05] ARM. *ARM Architecture Reference Manual - ARM DDI 0100I*, 2005.

[Aug85] L. Augustsson. Compiling pattern matching. In *Proceedings of the Conference on Functional Programming Languages and Computer Architecture*, Lecture Notes in Computer Science 201, 1985.

[Bar84] H. Barendregt. *The Lambda Calculus : Its Syntax and Semantics*. North-Holland, 2nd edition, 1984.

[Bar92] H. Barendregt. Lambda calculi with types. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of logic in computer science*, volume 2. Oxford University Press, 1992.

[Bar98a] G. Barthe. Existence and uniqueness of normal forms in pure type systems with $\beta\eta$-conversion. In *Proceedings of the 12th International Conference on Computer Science Logic*, Lecture Notes in Computer Science 1584, 1998.

[Bar98b] G. Barthe. The relevance of proof-irrelevance. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 1443, 1998.

[Bar99a] B. Barras. *Auto-validation d'un système de preuves avec familles inductives*. PhD thesis, Université Paris VII, France, 1999.

[Bar99b] G. Barthe. Expanding the cube. In *Proceedings of the 2nd International Conference on Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science 1578, 1999.

[BBKW89] J. C. M. Baeten, J. A. Bergstra, J. W. Klop, and W. P. Weijland. Term-rewriting systems with rule priorities. *Theoretical Computer Science*, 67 :282–301, 1989.

[BC93] F. W. Burton and R. D. Cameron. Pattern matching with abstract data types. *Journal of Functional Programming*, 3(2) :171–190, 1993.

[BCD+00] P. Borovanský, H. Cirstea, H. Dubois, C. Kirchner, H. Kirchner, P.-E. Moreau, C. Ringeissen, and M. Vittek. *ELAN User Manual*. INRIA Nancy, France, 2000. `http://elan.loria.fr/`.

[BCDC83] H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4) :931–940, 1983.

[BCDdS02] G. Barthe, P. Courtieu, G. Dufay, and S. de Sousa. Tool-assisted specification and verification of the JavaCard platform. In *Proceedings of the 9th International Conference on Algebraic Methodology and Software Technology*, Lecture Notes in Computer Science 2422, 2002.

[BCGD+06] F. Blanqui, S. Coupet-Grimal, W. Delobel, S. Hinderer, and A. Koprowski. CoLoR : a Coq Library on Rewriting and termination. In *8th International Workshop on Termination*, 2006. `http://color.inria.fr/`.

[BDD07] R. Bonichon, D. Delahaye, and D. Doligez. Zenon : an extensible automated theorem prover producing checkable proofs. In *Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science 4790, 2007.

[Bel58] R. Bellman. On a routing problem. *Quaterly Applied Mathematics*, XVI(1) :87–90, 1958.

[Ber88] S. Berardi. Towards a mathematical analysis of the Coquand-Huet calculus of constructions and the other systems in Barendregt's cube. Technical report, Carnegie-Mellon, Pittsburgh, USA, 1988.

[Ber05a] U. Berger. Continuous semantics for strong normalization. In *Proceedings of the 1st Conference on Computability in Europe* , Lecture Notes in Computer Science 3526, 2005.

[Ber05b] U. Berger. Strong normalization for applied lambda calculi. *Logical Methods in Computer Science*, 1(2) :1–14, 2005.

[Ber08] U. Berger. A domain model characterising strong normalisation. *Annals of Pure and Applied Logic*, 156(1) :39–50, 2008.

[BFG97] F. Barbanera, M. Fernández, and H. Geuvers. Modularity of strong normalization in the algebraic-λ-cube. *Journal of Functional Programming*, 7(6) :613–660, 1997.

[BFG+04] G. Barthe, M. J. Frade, E. Giménez, L. Pinto, and T. Uustalu. Type-based termination of recursive definitions. *Mathematical Structures in Computer Science*, 14(1) :97–141, 2004.

[BG95a] G. Barthe and H. Geuvers. Congruence types. In *Proceedings of the 9th International Conference on Computer Science Logic*, Lecture Notes in Computer Science 1092, 1995.

[BG95b] G. Barthe and H. Geuvers. Modular properties of algebraic type systems. In *Proceedings of the 2nd International Workshop on Higher-Order Algebra, Logic and Term Rewriting*, Lecture Notes in Computer Science 1074, 1995.

[BHJ+10] F. Blanqui, C. Helmstetter, V. Joloboff, J.-F. Monin, and X. Shi. SimSoC-Cert : a certified simulator for systems on chip, 2010. Draft.

[BHJ+11] F. Blanqui, C. Helmstetter, V. Joloboff, J.-F. Monin, and X. Shi. Designing a CPU model : from a pseudo-formal document to fast code. In *Proceedings of the 3rd Workshop on Rapid Simulation and Performance Evaluation : Methods and Tools*, 2011.

[BHK07] P. Brauner, C. Houtmann, and C. Kirchner. Principles of superdeduction. In *Proceedings of the 22nd IEEE Symposium on Logic in Computer Science*, 2007.

[BHW07] F. Blanqui, T. Hardin, and P. Weis. On the implementation of construction functions for non-free concrete data types. In *Proceedings of the 16th European Symposium on Programming*, Lecture Notes in Computer Science 4421, 2007.

[BJM00] A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoretical Computer Science*, 236 :35–132, 2000.

[BJO99] F. Blanqui, J.-P. Jouannaud, and M. Okada. The calculus of algebraic constructions. In *Proceedings of the 10th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 1631, 1999.

[BJO02] F. Blanqui, J.-P. Jouannaud, and M. Okada. Inductive-data-type systems. *Theoretical Computer Science*, 272 :41–68, 2002.

[BJP10] J.-P. Bernardy, P. Jansson, and R. Paterson. Parametricity and dependent types. In *Proceedings of the 15th ACM International Conference on Functional Programming*, SIGPLAN Notices 45(9), 2010.

[BJR06] F. Blanqui, J.-P. Jouannaud, and A. Rubio. Higher-order termination : from Kruskal to computability. In *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science 4246, 2006. Invited paper.

[BJR07] F. Blanqui, J.-P. Jouannaud, and A. Rubio. HORPO with computability closure : A reconstruction. In *Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science 4790, 2007.

[BJR08] F. Blanqui, J.-P. Jouannaud, and A. Rubio. The computability path ordering : the end of a quest. In *Proceedings of the 22nd International Conference on Computer Science Logic*, Lecture Notes in Computer Science 5213, 2008. Invited paper.

[BJS07] F. Blanqui, J.-P. Jouannaud, and P.-Y. Strub. Building decision procedures in the calculus of inductive constructions. In *Proceedings of the 21th International Conference on Computer Science Logic*, Lecture Notes in Computer Science 4646, 2007.

[BJS08] F. Blanqui, J.-P. Jouannaud, and P.-Y. Strub. From formal proofs to mathematical proofs : A safe, incremental way for building in first-order decision procedures. In *Proceedings of the 5th International Conference on Theoretical Computer Science*, International Federation for Information Processing 273, 2008.

[BJSW11] B. Barras, J.-P. Jouannaud, P.-Y. Strub, and Q. Wang. CoqMTU : a higher-order type theory with a predicative hierachy of universes parameterized by a decidable first-order theory. In *Proceedings of the 26th IEEE Symposium on Logic in Computer Science*, 2011.

[BK86] A. Bergstra and J. W. Klop. Conditional rewrite rules : Confluency and termination. *Journal of Computer and System Sciences*, 32(3) :323–362, 1986.

[BK07] C. Bertolissi and C. Kirchner. The rewriting calculus as a combinatory reduction system. In *Proceedings of the 10th International Conference on Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science 4423, 2007.

[BK09] F. Blanqui and A. Koprowski. Automated verification of termination certificates. Technical Report 6949, INRIA Rocquencourt, France, 2009.

[BK11] F. Blanqui and A. Koprowski. CoLoR : a Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates. *Mathematical Structures in Computer Science*, 21(4) :827–859, 2011.

[BKN87] D. Benanav, D. Kapur, and P. Narendran. Complexity of matching problems. *Journal of Symbolic Computation*, 3(1-2) :203–216, 1987.

[BKR06] F. Blanqui, C. Kirchner, and C. Riba. On the confluence of λ-calculus with conditional rewriting. In *Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science 3921, 2006.

[BKR10] F. Blanqui, C. Kirchner, and C. Riba. On the confluence of lambda-calculus with conditional rewriting. *Theoretical Computer Science*, 411(37) :3301–3327, 2010.

[BL09] S. Blazy and X. Leroy. Mechanized semantics for the Clight subset of the C language. *Journal of Automated Reasoning*, 43(3) :263–288, 2009.

[Bla98] F. Blanqui. The calculus of algebraic and inductive constructions. Master's thesis, Université Paris VII, France, 1998.

[Bla00] F. Blanqui. Termination and confluence of higher-order rewrite systems. In *Proceedings of the 11th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 1833, 2000.

[Bla01a] F. Blanqui. Definitions by rewriting in the calculus of constructions (extended abstract). In *Proceedings of the 16th IEEE Symposium on Logic in Computer Science*, 2001.

[Bla01b] F. Blanqui. *Théorie des Types et Récriture*. PhD thesis, Université Paris XI, Orsay, France, 2001. Available in English as "Type Theory and Rewriting".

[Bla03a] F. Blanqui. Inductive types in the calculus of algebraic constructions. In *Proceedings of the 6th International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science 2701, 2003.

[Bla03b] F. Blanqui. Rewriting modulo in deduction modulo. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 2706, 2003.

[Bla04a] F. Blanqui. Définition de la classe de réécriture à intégrer. Projet Averroes, lot 5.1, fourniture 1, 2004.

[Bla04b] F. Blanqui. Proposition d'architecture du moteur de test de conversion. Projet Averroes, lot 5.1, fourniture 2, 2004.

[Bla04c] F. Blanqui. Prototype d'extension du système Coq. Projet Averroes, lot 5.1, fourniture 3, 2004.

[Bla04d] F. Blanqui. A type-based termination criterion for dependently-typed higher-order rewrite systems. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 3091, 2004.

[Bla05a] F. Blanqui. Decidability of type-checking in the calculus of algebraic constructions with size annotations. In *Proceedings of the 19th International Conference on Computer Science Logic*, Lecture Notes in Computer Science 3634, 2005.

[Bla05b] F. Blanqui. Definitions by rewriting in the calculus of constructions. *Mathematical Structures in Computer Science*, 15(1) :37–92, 2005.

[Bla05c] F. Blanqui. Inductive types in the calculus of algebraic constructions. *Fundamenta Informaticae*, 65(1-2) :61–86, 2005.

[Bla06a] F. Blanqui. Higher-order dependency pairs. In *8th International Workshop on Termination*, 2006.

[Bla06b] F. Blanqui. (HO)RPO revisited. Technical Report 5972, INRIA, 2006.

[Bla07] F. Blanqui. Computability closure : Ten years later. In *Rewriting, Computation and Proof – Essays Dedicated to Jean-Pierre Jouannaud on the Occasion of His 60th Birthday*, volume 4600 of *Lecture Notes in Computer Science*, 2007.

[Bla10] F. Blanqui. A size-based termination criterion for dependently-typed higher-order rule-based programs. `http://www-rocq.inria.fr/~blanqui/`, 2010. Draft for a journal. 62 pages.

[Bla12] F. Blanqui. Terminaison des systèmes de réécriture d'ordre supérieur basée sur la notion de clôture de calculabilité. `http://www-rocq.inria.fr/~blanqui/divers/hdr-mono.pdf`, 2012. Habilitation à diriger des recherches.

[BM79] R. Boyer and J. Moore. *A computational logic*. Academic Press, 1979.

[BM96] G. Barthe and P.-A. Melliès. On the subject reduction property for algebraic type systems. In *Proceedings of the 10th International Conference on Computer Science Logic*, Lecture Notes in Computer Science 1258, 1996.

[BMS+96] F. W. Burton, E. Meijer, P. Sansom, S. Thompson, and P. Wadler. Views : An extension to Haskell pattern matching. `http://www.haskell.org/extensions/views.html`, 1996.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[BN02] S. Berghofer and T. Nipkow. Executing higher order logic. In *Proceedings of the International Workshop on Types for Proofs and Programs*, Lecture Notes in Computer Science 2646, 2002.

[Boe10a] M. Boespflug. Conversion by evaluation. In *Proceedings of the 12th International Symposium on Practical Aspects of Declarative Languages*, Lecture Notes in Computer Science 5937, 2010.

[Boe10b] M. Boespflug. Dedukti version 1.1.3. `http://www.lix.polytechnique.fr/dedukti/`, 2010.

[Böh68] C. Böhm. Alcune proprietà delle forme $\beta\eta$-normali nel λk-calcolo. Technical Report 696, Istituto Nazionale per le Applicazioni del Calcolo, Roma, Italy, 1968.

[Bou97] S. Boutin. *Réflexions sur les quotients*. PhD thesis, Université Paris 7, France, 1997.

[BQS80] R. Burstall, D. Mac Queen, and D. Sannella. HOPE : an experimental applicative language. Technical Report CSR-62-80, Edimburg University, UK, 1980.

[BR01] C. Borralleras and A. Rubio. A monotonic higher-order semantic path ordering. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science 2250, 2001.

[BR06] F. Blanqui and C. Riba. Combining typing and size constraints for checking the termination of higher-order conditional rewrite systems. In *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science 4246, 2006.

[BR09] F. Blanqui and C. Roux. On the relation between sized-types based termination and semantic labelling. In *Proceedings of the*

23rd International Conference on Computer Science Logic, Lecture Notes in Computer Science 5771, 2009.

[BS92] F. Baader and K. Schulz. Unification in the union of disjoint equational theories : Combining decision procedures. In *Proceedings of the 11th International Conference on Automated Deduction*, Lecture Notes in Computer Science 607, 1992.

[BT88] V. Breazu-Tannen. Combining algebra and higher-order types. In *Proceedings of the 3rd IEEE Symposium on Logic in Computer Science*, 1988.

[BTG89] V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic strong normalization. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 372, 1989.

[BTG91] V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic strong normalization. *Theoretical Computer Science*, 83(1) :3–28, 1991.

[BTG94] V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic confluence. *Information and Computation*, 114(1) :1–29, 1994.

[BW08] F. Blanqui and P. Weis. Moca, a construction function generator for abstract data types. `http://moca.inria.fr/`, 2008.

[CCF+07] E. Contejean, P. Courtieu, J. Forest, O. Pons, and X. Urbain. Certification of automated termination proofs. In *Proceedings of the 7th International Workshop on Frontiers of Combining Systems*, Lecture Notes in Computer Science 4720, 2007.

[CCKL07] S. Conchon, E. Contejean, J. Kanig, and S. Lescuyer. Lightweight integration of the Ergo theorem prover inside a proof assistant. In *Proceedings of the 2nd Workshop on Automated Formal Methods*, 2007.

[CD78] M. Coppo and M. Dezani. A new type-assignment for λ-terms. *Archive for Mathematical Logic*, 19 :139–156, 1978.

[CDE+05] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *Maude Manual (Version 2.2)*. Computer Science Laboratory, SRI International and Department of Computer Science, University of Illinois at Urbana-Champaign, USA, 2005. `http://maude.cs.uiuc.edu/`.

[Cer10] Certification problem format. `http://cl-informatik.uibk.ac.at/software/cpf/`, 2010.

[CF58] H. B. Curry and R. Feys. *Combinatory Logic*. North-Holland, 1958.

[CF06] S. Conchon and J.-C. Filliâtre. Type-safe modular hash-consing. In *Proceedings of the ACM SIGPLAN Workshop on ML*, 2006.

[CFU08a] E. Contejean, J. Forest, and X. Urbain. Deep-embedded unification. Technical Report 1547, CEDRIC, France, 2008.

[CFU08b] P. Courtieu, J. Forest, and X. Urbain. Certifying a termination criterion based on graphs, without graphs. In *Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science 5170, 2008.

[CG90] T. Coquand and J. Gallier. A proof of strong normalization for the theory of constructions using a Kripke-like interpretation. Technical Report MS-CIS-90-44, University of Pennsylvania, 1990.

[CGD06a] S. Coupet-Grimal and W. Delobel. A constructive axiomatization of the recursive path ordering. Technical Report 28, LIF, Université de la Méditerranée, France, 2006.

[CGD06b] S. Coupet-Grimal and W. Delobel. An effective proof of the well-foundedness of the multiset path ordering. *Applicable Algebra in Engineering Communication and Computing*, 17(6) :453–469, 2006.

[CGD06c] S. Coupet-Grimal and W. Delobel. Une preuve effective de la bonne fondaison de l'ordre récursif multi-ensemble sur les chemins. In *Proceedings of the 17èmes Journées Francophones des Langages Applicatifs*, 2006.

[CH88] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76(2-3) :95–120, 1988.

[Che83] P. Le Chenadec. *Formes canoniques dans les algèbres finiment présentées*. PhD thesis, Université Paris-Sud, France, 1983.

[Che85] P. Le Chenadec. A catalogue of complete group representations. Technical Report 439, INRIA, France, 1985.

[Che86] P. Le Chenadec. *Canonical forms in finitely presented algebras*. Research notes in theoretical computer science. Pitman, 1986.

[Che98] G. Chen. *Subtyping, Type Conversion and Transitivity Elimination*. PhD thesis, Université Paris VII, France, 1998.

[Chr03a] J. Chrząszcz. Implementation of modules in the Coq system. In *Proceedings of the 16th International Conference on Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science 2758, 2003.

[Chr03b] J. Chrząszcz. Modules in Coq are and will be correct. In *Proceedings of the International Workshop on Types for Proofs and Programs*, Lecture Notes in Computer Science 3085, 2003.

[Chr04] J. Chrząszcz. *Modules in Type Theory with Generative Definitions*. PhD thesis, Université d'Orsay, France and Warsaw University, Poland, 2004.

[Chu32] A. Church. A set of postulates for the foundations of logic. *Annals of Mathematics*, 2(33) :346–366, 1932. and 34, 839-864.

[Chu40] A. Church. A simple theory of types. *Journal of Symbolic Logic*, 5 :56–68, 1940.

[Cir00] H. Cirstea. *Calcul de Réécriture : Fondements et Applications.* PhD thesis, Université Nancy I, France, 2000.

[CK99] H. Cirstea and C. Kirchner. An introduction to the rewriting calculus. Technical Report RR 3818, INRIA, France, 1999.

[CK01a] W. N. Chin and S. C. Khoo. Calculating sized types. *Journal of Higher-Order and Symbolic Computation*, 14(2-3) :261–300, 2001.

[CK01b] H. Cirstea and C. Kirchner. The rewriting calculus. *Logic Journal of the Interest Group in Pure and applied Logic*, 9(3) :339–410, 2001.

[CM96] E. Contejean and C. Marché. CiME : Completion modulo E. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 1103, 1996.

[CMMU04] E. Contejean, C. Marché, B. Monate, and X. Urbain. CiME version 2.02. `http://cime.lri.fr`, 2004.

[Con04] E. Contejean. A certified AC matching algorithm. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 3091, 2004.

[Con07] E. Contejean. Modelling permutations in Coq for Coccinelle. In *Rewriting, Computation and Proof – Essays Dedicated to Jean-Pierre Jouannaud on the Occasion of His 60th Birthday*, volume 4600 of *Lecture Notes in Computer Science*, 2007.

[Coq85] T. Coquand. *Une théorie des constructions.* PhD thesis, Université Paris VII, France, 1985.

[Coq86] T. Coquand. An analysis of Girard's paradox. In *Proceedings of the 1st IEEE Symposium on Logic in Computer Science*, 1986.

[Coq92] T. Coquand. Pattern matching with dependent types. In *Proceedings of the International Workshop on Types for Proofs and Programs*, 1992.

[Coq10] Coq Development Team. *The Coq Reference Manual, Version 8.3.* INRIA Rocquencourt, France, 2010. `http://coq.inria.fr/`.

[Cou98] J. Courant. *Un calcul de modules pour les systèmes de types purs.* PhD thesis, École Normale Supérieure, Lyon, France, 1998.

[Cou01a] P. Courtieu. Normalized types. In *Proceedings of the 15th International Conference on Computer Science Logic*, Lecture Notes in Computer Science 2142, 2001.

[Cou01b] P. Courtieu. *Représentation de structures non libres en théorie des types.* PhD thesis, University Paris-Sud, France, 2001.

[Cou02] J. Courant. Strong normalization with singleton types. In *Proceedings of the 2nd Workshop on Intersection Types and Related Systems*, Electronic Notes in Theoretical Computer Science 70(1), 2002.

[Cou07] J. Courant. MC_2 a module calculus for pute type systems. *Journal of Functional Programming*, 17(3) :287–352, 2007.

[Cou09] D. Cousineau. *Models and proof normalization.* PhD thesis, École Polytechnique, France, 2009.

[CPU+10] E. Contejean, A. Paskevich, X. Urbain, P. Courtieu, O. Pons, and J. Forest. A3PAT, an approach for certified automated termination proofs. In *Proceedings of the ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, 2010.

[CS06] T. Coquand and A. Spiwack. A proof of strong normalization using domain theory. In *Proceedings of the 21st IEEE Symposium on Logic in Computer Science*, 2006.

[CS07] T. Coquand and A. Spiwack. A proof of strong normalization using domain theory. *Logical Methods in Computer Science*, 3(4) :1–16, 2007.

[dB68] N. de Bruijn. The mathematical language AUTOMATH, its usage, and some of its extensions. In *Proc. of the Symp. on Automatic Demonstration*, Lecture Notes in Mathematics. Springer, 1968. Reprinted in [GNdV94].

[Del01] D. Delahaye. *Conception de langages pour décrire les preuves et les automatisations dans les outils d'aide à la preuve - Une étude dans le cadre du système Coq.* PhD thesis, Université Paris VI, France, 2001.

[Der79] N. Dershowitz. Orderings for term rewriting systems. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, 1979.

[Der82] N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17 :279–301, 1982.

[DHDG04] C. Dubois, T. Hardin, and V. Viguié Donzeau-Gouge. Building certified components within Focal. In *Proceedings of the 5thSymposium on Trends in Functional Programming*, 2004.

[DHK98] G. Dowek, T. Hardin, and C. Kirchner. Theorem proving modulo. Technical Report 3400, INRIA Rocquencourt, France, 1998.

[DHK01] G. Dowek, T. Hardin, and C. Kirchner. HOL-lambda-sigma : an intentional first-order expression of higher-order logic. *Mathematical Structures in Computer Science*, 11 :1–25, 2001.

[DHK03] G. Dowek, T. Hardin, and C. Kirchner. Theorem proving modulo. *Journal of Automated Reasoning*, 31 :33–72, 2003.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6. North-Holland, 1990.

[dK03] N. de Kleijn. Well-foundedness of RPO in Coq. Master's thesis, Free University of Amsterdam, 2003.

[dKKvR04] N. de Kleijn, A. Koprowski, and F. van Raamsdonk. Well-foundedness of the recursive path ordering in Coq. In *Dutch Proof Tools Day*, 2004.

[DM07] G. Dowek and A. Miquel. Cut elimination for zermelo set theory, 2007. Draft.

[DOS88] N. Dershowitz, M. Okada, and G. Sivakumar. Canonical conditional rewrite systems. In *Proceedings of the 9th International Conference on Automated Deduction*, Lecture Notes in Computer Science 310, 1988.

[Dou91] D. Dougherty. Adding algebraic rewriting to the untyped lambda calculus. In *Proceedings of the 4th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 488, 1991.

[Dou92] D. Dougherty. Adding algebraic rewriting to the untyped lambda calculus. *Information and Computation*, 101(2) :251–267, 1992.

[Dow01a] G. Dowek. Higher-order unification and matching. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, chapter 16. Elsevier, 2001.

[Dow01b] G. Dowek. The Stratified Foundations as a theory modulo. In *Proceedings of the 5th International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science 2044, 2001.

[Dow06] G. Dowek. Truth values algebras and proof normalization. In *Proceedings of the International Workshop on Types for Proofs and Programs*, Lecture Notes in Computer Science 4502, 2006.

[Dow09] G. Dowek. On the convergence of reduction-based and model-based methods in proof theory. *Logic Journal of the Interest Group in Pure and applied Logic*, 17(5) :489–497, 2009.

[DST80] P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpression problem. *Journal of the ACM*, 27(4) :758–771, 1980.

[DW98] G. Dowek and B. Werner. Proof normalization modulo. Technical Report 3542, INRIA Rocquencourt, France, 1998.

[DW03] G. Dowek and B. Werner. Proof normalization modulo. *Journal of Symbolic Logic*, 68(4) :1289–1316, 2003.

[DW05] G. Dowek and B. Werner. Arithmetic as a theory modulo. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 3467, 2005.

[Dyb00] P. Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, 65(2) :525–549, 2000.

[ECU09] C. Marché E. Contejean and X. Urbain. CiME version 3. `http://a3pat.ensiie.fr/`, 2009.

[Eke96] S. Eker. Fast matching in combinations of regular equational theories. In *Proceedings of the 1st International Workshop on Rewriting Logic and Applications*, Electronic Notes in Theoretical Computer Science 4, 1996.

[EWZ06] J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, Lecture Notes in Computer Science 4130, 2006.

[FM01] F. Le Fessant and L. Maranget. Optimizing pattern-matching. In *Proceedings of the 6th ACM International Conference on Functional Programming*, SIGPLAN Notices 36(10), 2001.

[FM07] J.-C. Filliâtre and C. Marché. The Why/Krakatoa/Caduceus platform for deductive program verification. In *Proceedings of the 19th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science 4590, 2007.

[FM10] A. Fox and M. Myreen. A trustworthy monadic formalization of the ARMv7 instruction set architecture. In *Proceedings of the International Conference on Interactive Theorem Proving*, Lecture Notes in Computer Science 6172, 2010.

[For56] L. R. Ford. Network flow theory. Technical Report P-923, RAND, 1956.

[Fox01] A. Fox. A HOL specification of the ARM instruction set architecture. Technical Report 545, University of Cambridge, Computer Laboratory, 2001.

[Gal90] J. Gallier. On Girard's "candidats de réductibilité". In P.-G. Odifreddi, editor, *Logic and Computer Science*. North-Holland, 1990.

[Gen33] G. Gentzen. *Investigations into logical deduction*. PhD thesis, Göttingen, Germany, 1933.

[Geu94] H. Geuvers. A short and flexible proof of strong normalization for the calculus of constructions. In *Proceedings of the International Workshop on Types for Proofs and Programs*, Lecture Notes in Computer Science 996, 1994.

[Ghe05] F. Ghenassia. *Transaction Level Modeling with SystemC. TLM Concepts and Applications for Embedded Systems*. Springer, 2005.

[GHLS03] J.-M. Gaillourdet, T. Hillenbrand, B. Löchner, and H. Spies. The new Waldmeister loop at work. In *Proceedings of the 19th International Conference on Automated Deduction*, Lecture Notes in Computer Science 2741, 2003. `http://www.waldmeister.org/`.

[Gim96] E. Giménez. *Un Calcul de Constructions infinies et son application à la vérification de systèmes communiquants*. PhD thesis, ENS Lyon, France, 1996.

[Gir71] J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse et son application à l'élimination des coupures dans l'analyse et la théorie des types. In J. Fenstad, editor, *Proc. of the 2nd Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1971.

[Gir72] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmetique d'ordre supérieur*. PhD thesis, Université Paris VII, France, 1972.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50 :1–102, 1987.

[GKK05] J. Glauert, D. Kesner, and Z. Khasidashvili. Expression reduction systems and extensions : An overview. In *Processes, Terms and Cycles : Steps to the Road of Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion os His 60th Birthday*, volume 3838 of *Lecture Notes in Computer Science*, 2005.

[GL02] B. Grégoire and X. Leroy. A compiled implementation of strong reduction. In *Proceedings of the 7th ACM International Conference on Functional Programming*, SIGPLAN Notices 37(9), 2002.

[GLT88] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1988.

[GMR+07] G. Gonthier, A. Mahboubi, L. Rideau, E. Tassi, and L. Théry. A modular formalisation of finite group theory. In *Proceedings of the 20th International Conference on Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science 4732, 2007.

[GMW77] M. Gordon, R. Milner, and C. Wadsworth. A metalanguage for interactive proof in lcf. Technical Report CSR-16-77, University of Edinburgh, UK, 1977.

[GNdV94] H. Geuvers, R. Nederpelt, and R. de Vrijer, editors. *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1994.

[Gog94] H. Goguen. *A typed operational semantics for type theory*. PhD thesis, Edinburgh University, UK, 1994.

[Gon] G. Gonthier. A computer-checked proof of the four colour theorem. `research.microsoft.com/en-us/people/gonthier/4colproof.pdf`.

[Gon07] G. Gonthier. The four colour theorem : Engineering of a formal proof. In *Proceedings of the 8th Asian Symposium on Computer Mathematics*, Lecture Notes in Computer Science 5081, 2007.

[Gor88] M. J. C. Gordon. HOL : a proof generating system for higher order logic. In *VLSI Specification, Verification and Synthesis*. Kluwer Academic Publishers, 1988.

[GPWZ02] H. Geuvers, R. Pollack, F. Wiedijk, and J. Zwanenburg. A constructive algebraic hierarchy in Coq. *Journal of Symbolic Computation*, 34(4) :271–286, 2002.

[Gré03] B. Grégoire. *Compilation des termes de preuves : un (nouveau) mariage entre Coq et OCaml*. PhD thesis, Université Paris 7, France, 2003.

[GRSK+10] J. Giesl, M. Raffelsieper, P. Schneider-Kamp, S. Swiderski, and R. Thiemann. Automated termination proofs for Haskell by term rewriting. *ACM Transactions on Programming Languages and Systems*, 28(2) :256–289, 2010.

[GSKT06] J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2 : Automatic termination proofs in the dependency pair framework. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, Lecture Notes in Computer Science 4130, 2006.

[GSSKT06] J. Giesl, S. Swiderski, P. Schneider-Kamp, and R. Thiemann. Automated termination analysis for Haskell : From term rewriting to programming languages. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 4098, 2006.

[GTSKF03] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Improving dependency pairs. In *Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science 2850, 2003.

[GTSKF06] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3) :155–203, 2006.

[GTW06] B. Grégoire, L. Théry, and B. Werner. A computational approach to pocklington certificates in type theory. In *Proceedings of the 8th Fuji International Symposium on Functional and Logic Programming*, Lecture Notes in Computer Science 3945, 2006.

[Gut77] J. Guttag. Abstract data types and the development of data structures. *Communications of the ACM*, 20 :396–404, 1977.

[GWZ00] H. Geuvers, F. Wiedijk, and J. Zwanenburg. A constructive proof of the fundamental theorem of algebra without using the rationals. In *Proceedings of the International Workshop on Types for Proofs and Programs*, Lecture Notes in Computer Science 2277, 2000.

[Haf09] F. Haftmann. *Code generation from specifications in higher order logic*. PhD thesis, Technische Universität München, Germany, 2009.

[Hal05] T. C. Hales. A proof of the Kepler conjecture. *Annals of Mathematics*, 162(3) :1063–1183, 2005.

[Ham07] M. Hamana. Higher-order semantic labelling for inductive datatype systems. In *Proceedings of the 9th ACM SIGPLAN International*

Conference on Principles and Practice of Declarative Programming, 2007.

[Har10] J. Harrison. *The Hol Light system reference*. University of Cambridge, DSTO and SRI International, 2010. `http://www.cl.cam.ac.uk/~jrh13/hol-light/`.

[HHM+05] T. C. Hales, J. Harrison, S. McLaughlin, T. Nipkow, S. Obua, and R. Zumkeller. A revision of the proof of the Kepler conjecture. *Discrete and Computational Geometry*, 44(1) :1–34, 2005.

[Hin82] J. R. Hindley. The simple semantics for coppo-dezani-sallé types. In *International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, 1982.

[Hin04] S. Hinderer. Certification des preuves de terminaison par interprétations polynomiales. Master's thesis, Université Henri Poincaré, Nancy, France, 2004.

[HJX09] C. Helmstetter, V. Joloboff, and H. Xiao. SimSoC : A full system simulation software for embedded systems. In *Proceedings of the IEEE International Workshop on Open-source Software for Scientific Computation*, 2009.

[HM07] N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool : Techniques and features. *Information and Computation*, 205(4) :474–511, 2007.

[HN10] F. Haftmann and T. Nipkow. Code generation via higher-order rewrite systems. In *Proceedings of the 10th Fuji International Symposium on Functional and Logic Programming*, Lecture Notes in Computer Science 6009, 2010.

[Hof95a] M. Hofmann. *Extensional concepts in intensional type theory*. PhD thesis, Edinburgh University, UK, 1995.

[Hof95b] M. Hofmann. A simple model for quotient types. In *Proceedings of the 2nd International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science 902, 1995.

[HOL10] *The HOL system reference*, 2010. `http://hol.sourceforge.net/`.

[How80] W. A. Howard. The formulae-as-types notion of construction (1969). In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, 1980.

[HPS96] J. Hughes, L. Pareto, and A. Sabry. Proving the correctness of reactive systems using sized types. In *Proceedings of the 23th ACM Symposium on Principles of Programming Languages*, 1996.

[HPWD09] T. Hardin, F. Pessaux, P. Weis, and D. Doligez. *FoCaLiZe 0.6.0 Reference Manual*, 2009. `http://focalize.inria.fr/`.

[Hub04] T. Hubert. Certification des preuves de terminaison en Coq. Master's thesis, Université Paris 7, France, 2004.

[Hue76] G. Huet. Résolution d'équations dans les langages d'ordre 1, 2, ..., ω, 1976. Thèse d'État, Université Paris VII, France.

[Hue80] G. Huet. Confluent reductions : Abstract properties and applications to term-rewriting systems. *Journal of the ACM*, 27(4) :797–821, 1980.

[Hul79] J.-M. Hullot. Associative-commutative pattern matching. In *Proceedings of the 6th International Joint Conference on Artificial Intelligence*, 1979.

[Hul80a] J.-M. Hullot. Canonical forms and unification. In *Proceedings of the 5th International Conference on Automated Deduction*, Lecture Notes in Computer Science 87, 1980.

[Hul80b] J.-M. Hullot. *Compilation de formes canoniques dans les théories équationnelles*. PhD thesis, Université Paris-Sud, 1980.

[IEE06] IEEE Computer Society. *IEEE Standard SystemC Language Reference Manual*, 2006.

[JB04] N. D. Jones and N. Bohr. Termination analysis of the untyped lambda-calculus. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 3091, 2004.

[JB08] N. D. Jones and N. Bohr. Call-by-value termination in the untyped λ-calculus. *Logical Methods in Computer Science*, 4(1) :1–39, 2008.

[JH08] V. Joloboff and C. Helmstetter. SimSoC : A SystemC TLM integrated ISS for full system simulation. In *Proceedings of the IEEE Asia Pacific Conference on Circuits and Systems*, 2008.

[JK86] J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal on Computing*, 15(4) :1155–1194, 1986.

[JL10] J.-P. Jouannaud and J. Li. A computability path ordering for polymorphic terms. In *11th International Workshop on Termination*, 2010.

[JO91] J.-P. Jouannaud and M. Okada. A computation model for executable higher-order algebraic specification languages. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, 1991.

[JO97a] J.-P. Jouannaud and M. Okada. Abstract data type systems. *Theoretical Computer Science*, 173(2) :349–391, 1997.

[JO97b] J.-P. Jouannaud and M. Okada. Inductive data type systems : Strong normalization and all that, 1997. Unpublished draft.

[Jou06] J.-P. Jouannaud. Modular church-rosser modulo. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 4098, 2006.

[JR96] J.-P. Jouannaud and A. Rubio. A recursive path ordering for higher-order terms in η-long β-normal form. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 1103, 1996.

[JR99] J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *Proceedings of the 14th IEEE Symposium on Logic in Computer Science*, 1999.

[JR06] J.-P. Jouannaud and A. Rubio. Higher-order orderings for normal rewriting. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 4098, 2006.

[JR07] J.-P. Jouannaud and A. Rubio. Polymorphic higher-order recursive path orderings. *Journal of the ACM*, 54(1) :1–48, 2007.

[JT08] J.-P. Jouannaud and Y. Toyama. Modular Church-Rosser modulo : The complete picture. *International Journal of Software and Informatics*, 2(1) :61–75, 2008.

[Kah95] S. Kahrs. Confluence of curried term-rewriting systems. *Journal of Symbolic Computation*, 19(6) :601–623, 1995.

[KB70] D. Knuth and P. Bendix. Simple word problems in universal algebra. In *Computational problems in abstract algebra, Proceedings of a Conference held at Oxford in 1967*, pages 263–297. Pergamon Press, 1970.

[KEH+09] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4 : Formal verification of an OS kernel. In *Proceedings of the 22nd ACM Symposium on Operating System Principles*, 2009.

[KF09] J. Kanig and J.-C. Filliâtre. Who : a verifier for effectful higher-order programs. In *Proceedings of the ACM SIGPLAN Workshop on ML*, 2009.

[Kha90] Z. Khasidashvili. Expression reduction systems. Technical Report 36, I. Vekua Institute of Applied Mathematics of Tbilisi State University, 1990.

[KISB09] K. Kusakari, Y. Isogai, M. Sakai, and F. Blanqui. Static dependency pair method based on strong computability for higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E92-D(10) :2007–2015, 2009.

[Kle36] S. C. Kleene. λ-definability and recursiveness. *Duke Mathematical Journal*, pages 340–353, 1936.

[Kle52] S. C. Kleene. Permutability of inferences in Gentzen's calculi LK and LJ. *Memoirs of the American Mathematical Society*, 10 :1–26, 1952.

[Klo80] J. W. Klop. *Combinatory Reduction Systems*. PhD thesis, Utrecht Universiteit, Netherlands, 1980. Published as Mathematical Center Tract 129.

[KM01] H. Kirchner and P.-E. Moreau. Promoting rewriting to a programming language : A compiler for non-deterministic rewrite programs in associative-commutative theories. *Journal of Functional Programming*, 11(2) :207–251, 2001.

[KMTdV94] J. W. Klop, A. Middeldorp, Y. Toyama, and R. de Vrijer. Modularity of confluence : A simplified proof. *Information Processing Letters*, 49 :101–109, 1994.

[KNT99] K. Kusakari, M. Nakamura, and Y. Toyama. Argument filtering transformation. In *Proceedings of the 1st International Conference on Principles and Practice of Declarative Programming*, Lecture Notes in Computer Science 1702, 1999.

[Kop04] A. Koprowski. Well-foundedness of the higher-order recursive path ordering in Coq. Master's thesis, Free University of Amsterdam, Netherlands, 2004. Technical report TI-IR-004.

[Kop06a] A. Koprowski. Certified higher-order recursive path ordering. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 4098, 2006.

[Kop06b] A. Koprwoski. A formalization of the simply typed lambda calculus in Coq, 2006.

[Kop08] A. Koprowski. *Termination of rewriting and its certification*. PhD thesis, Technische Universiteit Eindhoven, Netherlands, 2008.

[Kop09] A. Koprowski. Coq formalization of the higher-order recursive path ordering. *Applicable Algebra in Engineering Communication and Computing*, 20(5-6) :379–425, 2009.

[Kop10] C. Kop. Wanda, a higher-order termination tool. http ://www.few.vu.nl/ kop/, 2010.

[Kou85] E. Kounalis. Completeness in data type specifications. In *Proceedings of the European Conference on Computer Algebra*, Lecture Notes in Computer Science 204, 1985.

[KR35] S. C. Kleene and J. B. Rosser. The inconsistency of certain formal logics. *Annals of Mathematics*, 2(36) :630–636, 1935.

[Kri97] J.-L. Krivine. *Lambda-calcul, types et modèles*. Dunod, 1997.

[Kru60] J. B. Kruskal. Well-quasi-ordering, the tree theorem, and vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95 :210–225, 1960.

[KvO95] Z. Khasidashvili and V. van Oostrom. Context-sensitive conditional expression reduction systems. In *SEGRAGRA 1995, Joint COMPUGRAPH/SEMAGRAPH Workshop on Graph Rewriting*

and Computation, volume 2 of *Electronic Notes in Theoretical Computer Science*, 1995.

[KvOdV08] J. W. Klop, V. van Oostrom, and R. de Vrijer. Lambda calculus with patterns. *Theoretical Computer Science*, 398(1-3) :16–31, 2008.

[KvOvR93] J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems : introduction and survey. *Theoretical Computer Science*, 121 :279–308, 1993.

[KW08] A. Koprowski and J. Waldmann. Arctic termination... below zero. In *Proceedings of the 19th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 5117, 2008.

[KW10] C. Keller and B. Werner. Importing HOL Light into Coq. In *Proceedings of the International Conference on Interactive Theorem Proving*, Lecture Notes in Computer Science 6172, 2010.

[KZ07] A. Koprowski and H. Zantema. Certification of matrix interpretations in Coq. In *9th International Workshop on Termination*, 2007.

[KZ08] A. Koprowski and H. Zantema. Certification of proving termination of term rewriting by matrix interpretations. In *Proceedings of the 34th International Conference on Current Trends in Theory and Practice of Computer Science*, Lecture Notes in Computer Science 4910, 2008.

[Lan75] D. S. Lankford. Canonical algebraic simplification in computational logic. Technical Report ATP-25, Automatic Theorem Proving Project, University of Texas, Austin, USA, 1975.

[LB77] D. Lankford and A. Ballantyne. Decision procedures for simple equational theories with commutative-associative axioms : Complete sets of commutative-associative reductions. Technical Report ATP-39, Automatic Theorem Proving Project, University of Texas, Austin, USA, 1977.

[LDF+10] X. Leroy, D. Doligez, A. Frisch, J. Garrigue, D. Rémy, and J. Vouillon. *The Objective Caml system release 3.12, Documentation and user's manual*. INRIA, France, 2010. `http://caml.inria.fr/`.

[Ler09] X. Leroy. A formally verified compiler back-end. *Journal of Automated Reasoning*, 43(4) :363–446, 2009.

[Let02] P. Letouzey. A new extraction for Coq. In *Proceedings of the International Workshop on Types for Proofs and Programs*, Lecture Notes in Computer Science 2646, 2002.

[Let04] P. Letouzey. *Programmation fonctionnelle certifiée : l'extraction de programmes dans l'assistant Coq*. PhD thesis, Université Paris-Sud, France, 2004.

[LJBA01] C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages*, 2001.

[LMR90] G. Lueker, N. Megiddo, and V. Ramachandran. Linear programming with two variables per inequality in poly-log time. *SIAM Journal on Computing*, 19(6) :1000–1010, 1990.

[Loa03] R. Loader. Higher-order β-matching is undecidable. *Logic Journal of the Interest Group in Pure and applied Logic*, 11(1) :51–68, 2003.

[LP95] O. Lysne and J. Piris. A termination ordering for higher order rewrite systems. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 914, 1995.

[LSS92] C. Loria-Saenz and J. Steinbach. Termination of combined (rewrite and λ-calculus) systems. In *Proceedings of the 3rd International Workshop on Conditional and Typed Rewriting Systems*, Lecture Notes in Computer Science 656, 1992.

[Luo90] Z. Luo. *An Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, UK, 1990.

[LZ74] B. H. Liskov and S. N. Zilles. Programming with abstract data types. In *Proceedings of the ACM SIGPLAN Symposium on Very High Level Languages*, volume 9 of *SIGPLAN Notices*, 1974.

[Mar92] L. Maranget. Compiling lazy pattern matching. In *Proceedings of the ACM Symposium on Lisp and Fonctional Programming*, 1992.

[Mar03] J.-Y. Marion. Analysing the implicit complexity of programs. *Information and Computation*, 183(1) :2–18, 2003.

[Mat98] R. Matthes. *Extensions of System F by Iteration and Primitive Recursion on Monotone Inductive Types*. PhD thesis, Ludwig Maximilians Universität, München, Germany, 1998.

[McB99] C. McBride. *Dependently typed functional programs and their proofs*. PhD thesis, University of Edinburgh, UK, 1999.

[Men87] N. P. Mendler. *Inductive Definition in Type Theory*. PhD thesis, Cornell University, USA, 1987.

[Mil84] R. Milner. A proposal for standard ML. In *Proceedings of the ACM Symposium on Lisp and Fonctional Programming*, 1984.

[Mil89] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. In *Proceedings of the International Workshop on Extensions of Logic Programming*, Lecture Notes in Computer Science 475, 1989.

[mkb10] mkbTT 2.0. `http://cl-informatik.uibk.ac.at/mkbtt/`, 2010.

[MN98] R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(2) :3–29, 1998.

[Mor73] J. H. Morris. Types are not sets. In *Proceedings of the 1st ACM Symposium on Principles of Programming Languages*, 1973.

[Mor99] P.-E. Moreau. *Compilation de règles de réécriture et de stratégies non-déterministes*. PhD thesis, Université Nancy I, France, 1999.

[MP69] Z. Manna and A. Pnueli. Formalization of properties of recursively defined functions. In *Proceedings of the 1st ACM Symposium on Theory of Computing*, 1969.

[MRV03] P.-E. Moreau, C. Ringeissen, and M. Vittek. A pattern matching compiler for multiple target languages. In *Proceedings of the 12th International Conference on Compiler Construction*, Lecture Notes in Computer Science 2622, 2003.

[Mül92] F. Müller. Confluence of the lambda calculus with left-linear algebraic rewriting. *Information Processing Letters*, 41(6) :293–299, 1992.

[MW02] A. Miquel and B. Werner. The not so simple proof-irrelevant model of CC. In *Proceedings of the International Workshop on Types for Proofs and Programs*, Lecture Notes in Computer Science 2646, 2002.

[Nip91] T. Nipkow. Higher-order critical pairs. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, 1991.

[NO77] G. Nelson and D. C. Oppen. Fast decision procedures based on union and find. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, 1977.

[NO80] G. Nelson and D. C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2) :356–364, 1980.

[NPW02] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL : A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002. `http://www.cl.cam.ac.uk/research/hvg/Isabelle/`.

[Ohl02] E. Ohlebusch. *Advanced Topics in Term Rewriting*. Springer, 2002.

[Oka89] M. Okada. Strong normalizability for the combined system of the typed lambda calculus and an arbitrary convergent term rewrite system. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, 1989.

[Oka98] C. Okasaki. Views for standard ML. In *Proceedings of the ACM SIGPLAN Workshop on ML*, 1998.

[Our05] N. Oury. Extensionality in the calculus of constructions. In *Proceedings of the 18th International Conference on Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science 3603, 2005.

[Pau86] L. Paulson. Proving termination of normalization functions for conditional expressions. *Journal of Automated Reasoning*, 2(1) :63–74, 1986.

[PJ87] S. Peyton-Jones. *The Implementation of Functional Programming Languages*. Prentice Hall, 1987.

[PJ03] S. Peyton-Jones, editor. *Haskell 98 Language and Libraries, The revised report*. Cambridge University Press, 2003.

[PM89] C. Paulin-Mohring. Extracting Fω's programs from proofs in the calculus of constructions. In *Proceedings of the 16th ACM Symposium on Principles of Programming Languages*, 1989.

[PM93] C. Paulin-Mohring. Inductive definitions in the system Coq - rules and properties. In *Proceedings of the 1st International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science 664, 1993.

[Pra65] D. Prawitz. *Natural deduction : a proof-theoretical study*. Almqvist and Wiksell, Stockholm, 1965.

[PS81] G. Peterson and M. Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28(2) :233–264, 1981.

[PS94] E. Poll and P. Severi. Pure type systems with definitions. In *Proceedings of the 3rd International Symposium on Logical Foundations of Computer Science*, Lecture Notes in Computer Science 813, 1994.

[Que84] D. Mac Queen. Modules for standard ML. In *Proceedings of the ACM Symposium on Lisp and Fonctional Programming*, 1984.

[Rey74] J. Reynolds. Towards a theory of type structure. In *Programming Symposium*, volume 19 of *Lecture Notes in Computer Science*, 1974.

[Rib07a] C. Riba. *Définitions par réécriture dans le lambda-calcul : confluence, réductibilité et typage*. PhD thesis, Institut National Polytechnique de Lorraine, Nancy, France, 2007.

[Rib07b] C. Riba. On the stability by union of reducibility candidates. In *Proceedings of the 10th International Conference on Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science 4423, 2007.

[Rib07c] C. Riba. Strong normalization as safe interaction. In *Proceedings of the 22nd IEEE Symposium on Logic in Computer Science*, 2007.

[Rib08] C. Riba. Stability by union of reducibility candidates for orthogonal constructor rewriting. In *Proceedings of the 4th Conference on Computability in Europe* , Lecture Notes in Computer Science 5028, 2008.

[Rib09] C. Riba. On the values of reducibility candidates. In *Proceedings of the 9th International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science 5608, 2009.

[Ros73] B. K. Rosen. Tree manipulating systems and Church-Rosser theorems. *Journal of the ACM*, 20(1) :160–187, 1973.

[Rou07] S. Le Roux. Acyclicity and finite linear extendability : a formal and constructive equivalence. In *Proceedings of the 20th International Conference on Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science 4732, 2007.

[Rou11] C. Roux. *Termination of higher-order rewrite systems.* PhD thesis, Université Henri Poincaré, Nancy, France, 2011.

[Rub10] A. Rubio. A GNU-Prolog implementation of HORPO. `http://www.lsi.upc.es/~albert/`, 2010.

[Sal78] P. Sallé. Une extension de la theorie des types en λ-calcul. In *Proceedings of the 1st International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 62, 1978.

[Sch77] H. Schwichtenberg. Proof theory : applications of cut elimination. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 867–895. North-Holland, 1977.

[SJ05] D. Sereni and N. D. Jones. Termination analysis of higher-order functional programs. In *Proceedings of the 3rd Asian Symposium on Programming Languages and Systems*, Lecture Notes in Computer Science 3780, 2005.

[SK01] M. Sakai and K. Kusakari. On new dependency pair method for proving termination of higher-order rewrite systems. In *Proceedings of the 1st International Workshop on Rewriting in Proof and Computation*, 2001.

[SK05] M. Sakai and K. Kusakari. On dependency pair method for proving termination of higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E88-D(3) :583–593, 2005.

[SKB10] S. Suzuki, K. Kusakari, and F. Blanqui. Argument filterings and usable rules in higher-order rewrite systems. Technical Report SS2010-24, Vol 110, No 169, IEICE, 2010.

[SKB11] S. Suzuki, K. Kusakari, and F. Blanqui. Argument filterings and usable rules in higher-order rewrite systems. *IPSJ Transactions on Programming*, 4(2) :1–12, 2011.

[SM99] M.-O. Stehr and J. Meseguer. Pure type systems in rewriting logic. In *Proceedings of the Workshop on Logical Frameworks and Meta-languages*, 1999.

[SM08] C. Sternagel and A. Middeldorp. Root labeling. In *Proceedings of the 19th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 5117, 2008.

[Soz08a] M. Sozeau. First-class type classes. In *Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science 5170, 2008.

[Soz08b] M. Sozeau. *Un environnement pour la programmation avec types dépendants.* PhD thesis, Université Paris 11, France, 2008.

[Soz09] M. Sozeau. A new look at generalized rewriting in type theory. *Journal of Formalized Reasoning*, 2(1) :41–62, 2009.

[SP03] C. Schürmann and F. Pfenning. A coverage checking algorithm for LF. In *Proceedings of the 16th International Conference on Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science 2758, 2003.

[Spi05] A. Spiwack. Reducibility domain. Master's thesis, ENS Cachan, 2005.

[SS89] M. Schmidt-Schauss. Unification in a combination of arbitrary disjoint equational theories. *Journal of Symbolic Computation*, 8 :51–99, 1989.

[ST10] C. Sternagel and R. Thiemann. Certified subterm criterion and certified usable rules. In *Proceedings of the 21st International Conference on Rewriting Techniques and Applications*, Leibniz International Proceedings in Informatics 6, 2010.

[Sta79] R. Statman. The typed λ-calculus is not elementary recursive. *Theoretical Computer Science*, 9 :73–81, 1979.

[Ste02] M.-O. Stehr. *Programming, Specification, and Interactive Theorem Proving - Towards a Unified Language based on Equational Logic, Rewriting Logic, and Type Theory.* PhD thesis, University of Hamburg, Germany, 2002.

[Ste05a] M.-O. Stehr. The open calculus of constructions (part I) : An equational type theory with dependent types for programming, specification, and interactive theorem proving. *Fundamenta Informaticae*, 68(1-2) :131–174, 2005.

[Ste05b] M.-O. Stehr. The open calculus of constructions (part II) : An equational type theory with dependent types for programming, specification, and interactive theorem proving. *Fundamenta Informaticae*, 68(3) :249–288, 2005.

[Sti06] C. Stirling. A game-theoretic approach to deciding higher-order matching. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming, part II*, Lecture Notes in Computer Science 4052, 2006.

[Sti09] C. Stirling. Decidability of higher-order matching. *Logical Methods in Computer Science*, 5(3) :1–52, 2009.

[Str03] P.-Y. Strub. Intégration de procédures de décision en théorie des types. Master's thesis, Université Paris VII, France, 2003.

[Str08] P.-Y. Strub. *Type Theory and Decision Procedures.* PhD thesis, École Polytechnique, France, 2008.

[Str10a] P.-Y. Strub. Coq modulo theories. `http://pierre-yves.strub.nu/research/coqmt/`, 2010.

[Str10b] P.-Y. Strub. Coq modulo theory. In *Proceedings of the 24th International Conference on Computer Science Logic*, Lecture Notes in Computer Science 6247, 2010.

[STWZ09] C. Sternagel, R. Thiemann, S. Winkler, and H. Zankl. CeTA – a tool for certified termination analysis. In *10th International Workshop on Termination*, 2009.

[SWS01] M. Sakai, Y. Watanabe, and T. Sakabe. An extension of dependency pair method for proving termination of higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E84-D(8) :1025–1032, 2001.

[Tai67] W. W. Tait. Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, 32(2) :198–212, 1967.

[Tai75] W. W. Tait. A realizability interpretation of the theory of species. In R. Parikh, editor, *Proceedings of the 1972 Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, 1975.

[Tak93] M. Takahashi. λ-calculi with conditional rules. In *Proceedings of the 1st International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science 664, 1993.

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its application. *Pacific Journal of Mathematics*, 5 :285–309, 1955.

[TC] Termination competition. `http://termination-portal.org/wiki/Termination_Competition`.

[Ter89] J. Terlouw. Een nadera bewijstheoretische analyse van GSTT's. Technical report, Katholiecke Universiteit Nijmegen, Netherlands, 1989.

[TeR03] TeReSe. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[TG03] R. Thiemann and J. Giesl. Size-change termination for term rewriting. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 2706, 2003.

[TG05] R. Thiemann and J. Giesl. The size-change principle and dependency pairs for termination of term rewriting. *Applicable Algebra in Engineering Communication and Computing*, 16(4) :229–270, 2005.

[Thi07] R. Thiemann. *The DP Framework for Proving Termination of Term Rewriting*. PhD thesis, RWTH Aachen University, Germany, 2007.

[Tho86] S. Thompson. Laws in Miranda. In *Proceedings of the ACM Symposium on Lisp and Fonctional Programming*, 1986.

[Tho90] S. Thompson. Lawful functions and program verification in Miranda. *Science of Computer Programming*, 13(2-3) :181–218, 1990.

[Toy87] Y. Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1) :128–143, 1987.

[TPD] Termination problem data base. `http://termination-portal.org/wiki/TPDB`.

[TS09] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science 5674, 2009.

[Tur37] A. M. Turing. Computability and λ-definability. *Journal of Symbolic Logic*, 2(153-163), 1937.

[vB92] S. van Bakel. Complete restrictions of the intersection type discipline. *Theoretical Computer Science*, 102 :135–163, 1992.

[vB93] S. van Bakel. *Intersection Type Disciplines in Lambda Calculus and Applicative Term Rewriting Systems.* PhD thesis, University of Nijmegen, Netherlands, 1993.

[vB95] S. van Bakel. Intersection type assignment systems. *Theoretical Computer Science*, 151(2) :385–435, 1995.

[vBF97] S. van Bakel and M. Fernández. Normalization results for typeable rewrite systems. *Information and Computation*, 133(2) :73–116, 1997.

[vO90] V. van Oostrom. Lambda calculus with patterns. Technical Report IR 228, Vrije Universiteit, Amsterdam, Netherlands, 1990.

[vO94] V. van Oostrom. *Confluence for Abstract and Higher-Order Rewriting.* PhD thesis, Vrije Universiteit Amsterdam, Netherlands, 1994.

[vO97] V. van Oostrom. Developing developments. *Theoretical Computer Science*, 175(1) :159–181, 1997.

[vR96] F. van Raamsdonk. *Confluence and Normalization for Higher-Order Rewriting.* PhD thesis, Vrije University Amsterdam, Netherlands, 1996.

[Wac05] B. Wack. *Typage et déduction dans le calcul de réécriture.* PhD thesis, Université Henri Poincaré, Nancy, France, 2005.

[Wad87] P. Wadler. Views : a way for pattern matching to cohabit with data abstraction. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, 1987.

[Wah07] D. Wahlstedt. *Dependent Type Theory with First-Order Parameterized Data Types and Well-Founded Recursion.* PhD thesis, Chalmers University of Technology, Sweden, 2007.

[WC03a] D. Walukiewicz-Chrząszcz. Termination of rewriting in the calculus of constructions. *Journal of Functional Programming*, 13(2) :339–414, 2003.

[WC03b] D. Walukiewicz-Chrząszcz. *Termination of Rewriting in the Calculus of Constructions*. PhD thesis, Warsaw University, Poland and Université d'Orsay, France, 2003.

[WCC08] D. Walukiewicz-Chrząszcz and J. Chrząszcz. Consistency and completeness of rewriting in the calculus of constructions. *Logical Methods in Computer Science*, 4(3 :8) :1–20, 2008.

[WCC10] D. Walukiewicz-Chrząszcz and J. Chrząszcz. Inductive consequences in the calculus of constructions. In *Proceedings of the International Conference on Interactive Theorem Proving*, Lecture Notes in Computer Science 6172, 2010.

[Wei03] P. Weis. Private constructors in OCaml. Caml Weekly News `http://alan.petitepomme.net/cwn/2003.07.01.html#5` and `http://alan.petitepomme.net/cwn/2003.07.08.html#7`, 2003.

[Wer94] B. Werner. *Une Théorie des Constructions Inductives*. PhD thesis, Université Paris VII, France, 1994.

[Wes11] E. Westbrook. Uniform logical relations. `http://www.cs.rice.edu/~emw4/uniform-lr.pdf`, 2011.

[Wie99] T. Wierzbicki. Complexity of the higher order matching. In *Proceedings of the 16th International Conference on Automated Deduction*, Lecture Notes in Computer Science 1632, 1999.

[Xi01] H. Xi. Dependent types for program termination verification. In *Proceedings of the 16th IEEE Symposium on Logic in Computer Science*, 2001.

[Xi02] H. Xi. Dependent types for program termination verification. *Journal of Higher-Order and Symbolic Computation*, 15(1) :91–131, 2002.

[Zan95] H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24 :89–105, 1995.

[Zen97] C. Zenger. Indexed types. *Theoretical Computer Science*, 187(1-2) :147–165, 1997.

[ZSHM10] H. Zankl, C. Sternagel, D. Hofbauer, and A. Middeldorp. Finding and certifying loops. In *Proceedings of the 36th International Conference on Current Trends in Theory and Practice of Computer Science*, Lecture Notes in Computer Science 5901, 2010.