Master Parisien de Recherche en Informatique
Université Paris VII

Rapport de stage

Coercion par prédicats en Coq

Matthieu Sozeau

Stage effectué au LRI dans l'équipe Démons, sous la direction de :
Christine Paulin-Mohring

Remerciements

Je remercie l'équipe Démons du LRI pour son accueil chaleureux. Je remercie en particulier Christine Paulin-Mohring pour son attention et sa présence constantes et Jean-Christophe Filliâtre pour son aide précieuse lors du développement, son dynamisme et sa passion pour le domaine de la programmation qui n'ont pas été sans influence sur mon orientation vers la recherche depuis mon arrivée à la faculté d'Orsay. Je suis reconnaissant envers mes colocataires thésards pour m'avoir supporté jusqu'ici. Je remercie enfin Claire et Joëlle sans qui ma vie serait bien différente.

Résumé

Coq est un assistant de preuve d'une grande expressivité pour le développement de théories mathématiques et informatiques, ce qui permet de traiter un large éventail de problèmes. Le langage de Coq, constitué d'un noyau fonctionnel de type ML enrichi par des types dépendants, permet de spécifier, vérifier puis extraire des programmes corrects par construction. En contrepartie, les programmes sont plus difficiles à écrire et maintenir que dans un pur langage de programmation de type ML, puisqu'ils mélangent les parties logiques et calculatoires. Pour remédier à ce problème, on propose le nouveau langage de description de programmes Russell, s'intégrant parfaitement à l'environnement de développement existant, qui permet de donner dans un premier temps le code et la spécification du programme et d'engendrer ensuite ses conditions de correction. Nous allons tout d'abord étudier le langage Russell, construire un algorithme de typage pour ses termes et montrer comment les traduire en termes à trous valides dans le système Coq. Notre contribution revient à intégrer un langage avec types dépendant à la PVS dans Coq. Une version préliminaire de notre méthode a d'ailleurs été intégrée à la version de développement de Coq.

Table des matières

1 Introduction **6**
- 1.1 Présentation de Coq . . . 6
  - 1.1.1 Preuve . . . 6
  - 1.1.2 Programmes . . . 7
- 1.2 Motivation . . . 7
  - 1.2.1 Un langage trop expressif ? . . . 7
  - 1.2.2 Mélange logique et calcul . . . 8
  - 1.2.3 Objectif . . . 8
- 1.3 Travaux Connexes . . . 8
  - 1.3.1 La tactique Program . . . 8
  - 1.3.2 Types sous-ensemble . . . 9
  - 1.3.3 Coercions . . . 9

2 Le calcul de coercion par prédicats **10**
- 2.1 Le langage Russell . . . 10
  - 2.1.1 Syntaxe . . . 10
  - 2.1.2 Sémantique . . . 10
- 2.2 Élaboration du système algorithmique et propriétés . . . 14
  - 2.2.1 Notations . . . 14
  - 2.2.2 Correction . . . 14
  - 2.2.3 Complétude et décidabilité . . . 17
- 2.3 Génération des obligations de preuve . . . 26
  - 2.3.1 Interprétation . . . 26
  - 2.3.2 Propriétés . . . 27

3 Subtac **58**
- 3.1 Existentielles . . . 58
  - 3.1.1 La tactique **eterm** . . . 58
- 3.2 Traitement de la récursion . . . 59
- 3.3 Traitement des inductifs . . . 59
- 3.4 `Program` et `Recursive program` . . . 59

4 Conclusion **61**

Bibliographie **62**

A Exemples **64**

Table des figures

2.1 Syntaxe . . . 11
2.2 Calcul de coercion par prédicats - version déclarative . . . 12
2.3 Coercion par prédicats - version déclarative . . . 13
2.4 Définition de $\mathcal{R}$ . . . 13
2.5 Calcul de coercion par prédicats - version algorithmique . . . 15
2.6 Définition de $\mu_\bullet()$ . . . 16
2.7 Coercion par prédicats - version algorithmique . . . 16
2.8 Interprétation dans CCI . . . 55
2.9 Définition de la réduction de tête . . . 55
2.10 Théorie équationnelle de $C_C?$ . . . 56
2.11 Réécriture de la coercion vers CCI . . . 56
2.12 Définition du jugement $\Gamma \vdash_\bullet t : T \equiv_{\beta\pi} u : U$ . . . 57

A.1 Script de preuve de la division euclidienne . . . 64
A.2 La division euclidienne avec SUBTAC . . . 65
A.3 La division euclidienne avec SUBTAC : script . . . 66

Chapitre 1

Introduction

Nous nous plaçons dans le cadre du système d'aide à la preuve Coq, auquel nous souhaitons intégrer un langage de programmation plus souple que le langage actuellement utilisé.

1.1 Présentation de Coq

Coq est un assistant de preuve dont la première version date de 1985, et qui est aujourd'hui développé dans le projet PCRI LogiCal (INRIA, LIX, LRI, CNRS). Originellement basé sur le Calcul des Constructions (CoC), il a été étendu au Calcul des Constructions Inductives (Cci) et contient aujourd'hui de nombreuses améliorations telles qu'un système sophistiqué d'extraction de programmes ou encore des procédures de décision pour automatiser la preuve.

Le développement de Coq est intimement lié à l'isomorphisme de Curry-Howard qui montre le lien entre logique intuitionniste et calcul. De cet isomorphisme, on peut déduire qu'élaborer une preuve du calcul propositionnel intuitionniste est équivalent à écrire un terme du λ-calcul simplement typé ($\lambda\!\rightarrow$). Par exemple, montrer que $A \Rightarrow A$ pour un certain A revient à écrire la fonction identité $\lambda x : A.x$ qui a bien pour type $A \rightarrow A$. Chaque logique constructive est donc associée à un λ-calcul particulier. Dans Coq, on utilise cet isomorphisme pour vérifier les preuves. Le noyau est simplement un typeur pour Cci. Si on peut typer un terme t de type T, alors on est assuré d'avoir trouvé une preuve constructive t de la formule T. Cette dualité se reflète aussi à l'utilisation de Coq où l'on a les deux visions : logique (développement mathématique, preuve) et calcul (développement informatique, programme).

1.1.1 Preuve

Coq est utilisé le plus souvent pour élaborer des théories mathématiques prouvées mécaniquement. Dans cette optique, l'utilisateur modélise un problème par des structures mathématiques et veut prouver certaines propriétés sur ce modèle (par exemple la preuve du théorème des quatres couleurs récemment terminée [7] utilisait des résultats de géométrie algébrique).

Pour prouver un but sous certaines hypothèses, on utilise des tactiques qui simulent un raisonnement déductif pour l'utilisateur. Celles-ci permettent par exemple d'introduire une hypothèse : pour le but $A \Rightarrow A$ on peut introduire l'hypothèse $H : A$ pour obtenir le but A ; ou bien d'en appliquer une (ou tout autre résultat déjà établi) : en appliquant l'hypothèse H on prouve le but directement. Ces tactiques peuvent être d'une complexité arbitraire (réécritures, procédures de décision pour l'arithmétique, etc . . .).

Les tactiques utilisées pour créer des preuves ne sont en fait qu'une interface au-dessus du noyau de Coq qui se réduit à un typeur pour Cci. A la fin d'une preuve, on a en effet construit un terme ($\lambda H : A.H$ dans notre exemple) que l'on va soumettre au typeur dont le but est de vérifier qu'il est bien de type $A \rightarrow A$. Les tactiques peuvent cependant être arbitrairement complexes (résolution d'équations de l'arithmétique, réecritures, etc...).

1.1.2 Programmes

D'un point de vue preuve de programmes, on a donc un environnement qui permet de vérifier qu'un programme (un terme du calcul) vérifie une certaine spécification (son type). Les types dépendants permettent de spécifier fortement les termes. Par exemple, la fonction de division euclidienne (qui renvoie le quotient et le reste de la division de son premier argument par son second) **div** : $\texttt{nat} \to \texttt{nat} \to \texttt{nat} * \texttt{nat}$ de ML est plus fortement spécifiée en Coq par **div** : $\texttt{nat} \to \{\ x : \texttt{nat} \mid x \neq 0\ \} \to \texttt{nat} * \texttt{nat}$ qui assure par typage qu'on ne divise pas par 0. Seulement, on ne peut pas écrire simplement un programme ML et donner sa spécification forte. Comme on a enrichi les types, on doit aussi enrichir les termes avec des termes de preuve, inutiles au calcul mais nécessaires pour garantir la correction logique du programme et le fait que la machine puisse vérifier mécaniquement la correction (annotations,...). Par exemple, si l'on veut appeler **div** sur 1 et n (pour n : nat), il faut construire un terme **div** 1 (**elt** $(\lambda x : \texttt{nat} \to x \neq 0)\ x\ p$) où p est une preuve de $n \neq 0$.

Nous allons nous intéresser particulièrement dans la suite au type sous-ensemble. Ce type est familier du mathématicien puisqu'on le retrouve partout en théorie des ensembles, et il est tristement célèbre puisque c'est en l'utilisant que Bertrand Russell à découvert le paradoxe éponyme en considérant le sous-ensemble particulier $\{x \mid x \notin x\}$ (il a ensuite étudié la première théorie des types en mathématiques !). Un type sous-ensemble est composé d'un type pour l'ensemble de départ et d'une propriété pouvant porter sur les objets de ce type.

Définition 1.1.1 (Type sous-ensemble). *En* Coq, $\{\ x : T \mid P\ \}$ *est le type des termes de type T vérifiant la propriété P. C'est un type inductif avec un unique constructeur* ***elt*** *prenant en arguments le type du témoin T, la propriété P puis le témoin t et la preuve p :* $\textbf{\textit{elt}} : \forall T : \texttt{Set}, \forall P : T \to \texttt{Prop}, \forall x : T, P\ x \to \{\ x : T \mid P\ \}$.

Informellement, un objet de type $\{\ x : T \mid P\ \}$ peut être vu comme une paire (x, p) où x est un objet de type T (le témoin) et p une preuve de $P[t/x]$. Le typeur a besoin de plus d'information, l'annotation $\lambda x : \texttt{nat} \to x \neq 0$ de notre exemple peut être nécessaire (en fait, dans ce cas précis, on peut reposer sur le système d'arguments implicites de Coq) pour avoir un système d'inférence décidable (on ne peut pas toujours inférer la propriété P à partir de sa preuve p puisqu'il est de type $P[t/x]$). On voit donc ici que l'on doit rajouter de nombreuses informations d'ordre logique à nos termes.

A l'inverse, on peut extraire un programme de toute preuve en éliminant les parties logiques et en ne conservant que la partie calculatoire d'un terme.

1.2 Motivation

Coq permet de développer des programmes complexes, de leur donner des spécifications fortes et de les vérifier automatiquement. On peut même extraire de ces développements des programmes corrects par construction. Il y a cependant certaines difficultés à développer en Coq que nous allons étudier maintenant.

1.2.1 Un langage trop expressif ?

Le langage de CCI permet de bien spécifier des fonctions non triviales, par exemple, on peut spécifier très fortement notre division euclidienne (dont on a vu qu'on pouvait la typer **div** : $\texttt{nat} \to \{\ x : \texttt{nat} \mid x \neq 0\ \} \to \texttt{nat} * \texttt{nat}$ précédement) :

$$\textbf{Definition div} : \forall a : \texttt{nat}, \forall b : \texttt{nat}, b \neq 0 \to \{\ q : \texttt{nat}\ \&\ \{\ r : \texttt{nat} \mid r < b \wedge a = b * q + r\ \}\}$$

Les types dépendants permettent de bien relier les entrées aux sorties et donc de spécifier les programmes aussi fortement que l'on désire, mais aussi de façon concise. En revanche, le terme de preuve correspondant à **div** est nettement plus long (de l'ordre de 60 lignes), et ne peut simplement pas être écrit d'une traite sans une expertise approfondie. Pour remédier à ce problème, on utilise des tactiques qui permettent d'écrire la

preuve/programme incrémentalement (voir figure A.1 page 64). L'inconvénient de cette méthode est que l'on n'obtient pas toujours le programme désiré au départ, puisque les tactiques cachent profondément leur effet sur le terme de preuve et donc le programme extrait. Cependant ce mode de fonctionnement est utile et utilisé par la majorité des utilisateurs de Coq avec succès (certification d'un compilateur (en partant de code faiblement spécifié) [8], théorème des quatres couleurs [7], . . .).

1.2.2 Mélange logique et calcul

Une difficulté essentielle lorsque l'on veut permettre à des utilisateurs non experts de développer dans Coq est le "mélange des genres" permanent entre logique et calcul. Pour appliquer une fonction division qui attend un dénominateur différent de 0 par exemple, il faut passer à la fois l'argument lui même, mais aussi une preuve de sa non-nullité. Lorsque l'on a l'habitude de programmer, ça n'est pas la chose la plus naturelle et l'on aimerait pouvoir découpler les parties codage et preuve pour simplement diviser le problème. Les parties logiques pourront souvent être résolues automatiquement par des tactiques.

1.2.3 Objectif

A long terme, on souhaite permettre à un utilisateur de programmer dans un langage proche de ML et de prouver ses programmes dans un deuxième temps à l'aide de Coq et ses tactiques. Une fois les preuves terminées, on peut extraire un programme correct par construction et essentiellement équivalent à celui de départ ou le réutiliser facilement dans l'environnement Coq.

1.3 Travaux Connexes

La preuve de programmes fonctionnels est un domaine de recherche actif. L'idée d'étendre les langages ML avec des types dépendants a été développée dans DML [18], Cayenne [1] et Ωmega [17]. Il s'agit dans ces travaux de faire un langage dont l'inférence est décidable, donc de restreindre les types dépendants à des domaines où l'on peut faire de la preuve automatique (DML) ou bien d'accroître la puissance du langage pour rendre l'utilisation des types dépendants plus aisée (Cayenne a la récursivité générale par exemple, comme n'importe quel langage de programmation usuel) mais en perdant l'idée de correction (et en perdant même la décidabilité du typage pour Cayenne).

Nous prenons le contre-pied de ces travaux en acceptant de générer des obligations de preuve et en essayant de trouver un langage le plus proche de ML possible tout en retenant la puissance de Coq et des types dépendants. Nous présentons maintenant des travaux directement liés à notre contribution.

1.3.1 La tactique Program

Il existe un travail réalisé dans Coq couvrant une partie de nos objectifs. Développée par Catherine Parent [13], la tactique Program permettait de synthétiser des preuves à partir de programmes. L'idée était de trouver un langage de programmation suffisamment restrictif pour réaliser une inversion de l'extraction, c'est-à-dire, à partir d'un terme essentiellement calculatoire (des annotations étaient nécessaires), retrouver un terme de preuve réalisant la spécification donnée. A partir de là, on était assuré que le programme extrait serait identique à celui que l'on écrivait pour sa partie informative. Cette méthode générale avait l'inconvénient d'être peu intuitive et de ne pas s'intégrer à l'environnement Coq. En particulier, appeler une fonction définie avec Program est aussi difficile qu'avec n'importe quelle définition Coq. Il n'existe pas de mécanisme permettant de faire la distinction de phase codage/preuve, qui permettrait de faire de simples appels et de vérifier ensuite que les arguments sont valides, ce qui est beaucoup plus naturel lorsque l'on programme. Liée à l'extraction interne qui a disparu dans les dernières versions de Coq (remplacée par la contribution de Pierre Letouzey [9]), la tactique Program n'est plus maintenue aujourd'hui.

1.3.2 Types sous-ensemble

Plutôt que de continuer dans la même direction, nous avons cherché à assouplir le système. L'assistant de preuve PVS [12] aux capacités similaires à Coq, intègre un mécanisme dénommé *Predicate subtyping* que nous allons présenter maintenant.

Les types sous-ensembles (définition 1.1.1) sont d'une grande utilité pour la spécification de programmes, comme nous l'avons vu pour **div** précécement : **Definition div** : `nat` $\rightarrow$ $\{ x : nat \mid x \neq 0 \} \rightarrow$ `nat * nat`.

L'idée du *Predicate subtyping* implémenté dans PVS [16, 14] est de considérer tout objet de type T comme un objet de type $\{ x : T \mid P \}$ pour P vraie et vice-versa. Comme tout objet t de type T ne vérifie pas forcément la propriété P, on génère des "*Type-checking conditions*" (Tcc), c'est à dire que l'on demande à l'utilisateur de prouver $P[t/x]$ pour assurer que le programme est correct.

1.3.3 Coercions

PVS n'a pas la même architecture que Coq, en particulier il n'y a pas de termes de preuve et de noyau pour vérifier ces termes. Il faut donc faire confiance à la quasi-totalité du code pour croire en la correction des programmes vérifiés. Le critère de De Bruijn, qui dit en substance qu'un petit noyau est plus sûr n'est pas respecté, alors que celui de Coq a même été formellement vérifié [2].

Dans notre cas, il faut générer des termes de preuve et donc le code correspondant à ce "sous-typage". Une littérature importante [6, 11] existe autour des systèmes à coercions explicites dont nous nous sommes inspirés pour réaliser la génération des termes. Dans un système à coercions explicites, on peut faire des abus de notations tels que utiliser un objet de type T à la place d'un de type U, mais on applique une coercion qui amène l'objet vers le type U avant de retyper dans un système sans coercions. Généralement les coercions sont très similaires à des identités, c'est-à-dire qu'elles sont calculatoirement insignifiantes mais leur utilisation facilite le développement. Dans Coq par exemple le système de coercions [15] a permis de développer des théories algébriques réutilisables sur plusieurs structures instantanément (un théorème sur les corps pouvant s'appliquer aux anneaux).

Les extensions du Calcul des Constructions avec des notions de sous-typage comme $\lambda C_{\leq}$ de Chen ne sont cependant pas dans la même catégorie que notre travail. En particulier, nous ne nous intéressons pas aux propriétés de normalisation ou de préservation du typage. On peut le voir plutôt comme un système syntaxique intelligent au-dessus du Calcul des Constructions.

Chapitre 2

Le calcul de coercion par prédicats

Nous avons développé un langage supportant le *Predicate subtyping* utilisable dans Coq. L'utilisateur peut définir des programmes dans un langage souple puis prouver certains buts pour obtenir finalement un terme de Cci complet vérifiable par le noyau. On peut finalement utiliser les types dépendants comme des types simples et s'occuper des dépendances dans un deuxième temps (pour la preuve). L'architecture de notre système est la suivante : on type le programme dans notre langage Russell où l'on peut faire des abus de notations avec les objets de type sous-ensemble, puis l'on réécrit le terme typé dans Cci en laissant des "trous" dans les termes qui désambigüent les abus et enfin Coq se charge de générer les obligations correspondant à ces trous.

On va donc tout d'abord présenter le langage Russell (section 2.1), puis un algorithme de typage correct et complet pour les programmes écrits en Russell. Ensuite on montrera comment plonger ce langage dans Cci en ajoutant les coercions adéquates et enfin on expliquera comment se déroule la génération des obligations de preuves à partir des termes engendrés par le plongement.

2.1 Le langage Russell

Le langage que nous voulons est très proche de ml, plus les annotations nécessaires pour avoir un typage précis et décidable. On étudie ici une restriction de ml, purement fonctionnelle et sans filtrage, qu'on étendra dans la suite de notre travail. On n'a donc pas de types inductifs mais on considère les types Σ, généralisation des tuples de ml formés par l'opérateur $*$.

2.1.1 Syntaxe

La syntaxe (figure 2.1) est directement inspirée des langages fonctionnels. On part du λ-calcul (variables, abstraction et application) puis l'on ajoute des constantes (pour les entiers, booléens, etc...) ainsi que les couples. La syntaxe $(x := \alpha, t : \tau)$ permet de créer des paires dépendantes, de type $\Sigma x : \tau.\tau$. On peut aussi appliquer un terme à un type pour instancier une fonction polymorphe par exemple.

Du côté des types, on a tout d'abord les types simples (constantes, flèche, produit cartésien) qui sont des cas particuliers du produit (Π) et de la somme (Σ) dépendants. Les variables introduites par ces types peuvent être utilisées lors des applications de types. On peut de plus abstraire sur les types avec le λ (polymorphisme) et les sortes. Enfin on peut appliquer un type à un terme $(\tau\ \alpha)$.

2.1.2 Sémantique

La sémantique du langage nous est donnée par un système de typage (figure 2.2 page 12). Le jugement de typage est défini inductivement par un ensemble de règles d'inférence. Dans notre cas ce sont les règles du Calcul des Constructions (Cc) étendu avec les Σ-types auxquelles on a ajouté une règle de coercion

$$
\begin{array}{rcl}
\alpha & ::= & x \\
& | & \lambda x : \tau \Rightarrow \alpha \\
& | & \alpha\ \alpha \\
& | & \alpha\ \tau \\
& | & \mathit{constant} \\
& | & (\alpha, \alpha)_{\Sigma x:\tau.\tau} \\
& | & \pi_1\ \alpha \mid \pi_2\ \alpha
\end{array}
$$

(a) Termes

$$
\begin{array}{rcl}
\tau & ::= & x \\
& | & \tau\ \tau \\
& | & \tau\ \alpha \\
& | & \lambda x : \tau \Rightarrow \tau \\
& | & \Pi x : \tau.\tau \\
& | & \Sigma x : \tau.\tau \\
& | & \{\, x : \tau \mid \tau \,\} \\
& | & \texttt{Set} \\
& | & \texttt{Prop} \\
& | & \texttt{Type} \\
& | & \mathit{constant}
\end{array}
$$

(b) Types

FIG. 2.1 – Syntaxe

(COERCE, figure 2.2) que l'on trouve classiquement dans les systèmes avec sous-typage avec le nom de subsumption. Le jugement $\Gamma \vdash t : T$ se lit : dans l'environnement Γ, t est de type T.

L'équivalence utilisée est $\equiv_{\beta\pi}$, soit la β-réduction classique ainsi que les projections π_i pour les paires.

La relation $\mathcal{R}$ définissant les produits formables dans le système est définie par les règles suivantes : On a un système proche du Calcul des Constructions avec types Σ, mais avec `Set` prédicatif (comme dans Coq). On n'a pas (`Prop`, `Set`, `Set`) dans notre relation $\mathcal{R}$ pour une bonne raison. Cela permet de créer des fonctions dépendant de propositions, par exemple $\Pi n : \texttt{nat}, n > 0 \to \Pi l : \text{list}\ A\ n \to A$. Or on veut à tout prix éviter d'introduire des termes de preuve dans notre langage, et l'on voit que cette fonction pourrait naturellement s'écrire $\Pi n : \{\, n : \texttt{nat} \mid n > 0 \,\} \to \Pi l : \text{list}\ A\ n \to A$. Encore une fois le type sous-ensemble nous permet d'éviter d'avoir à passer des termes de preuve directement.

Les sommes formables dans le système sont réduites au couples d'objets de types de même sorte $s \in \{\texttt{Prop}, \texttt{Set}\}$. Dans le premier cas les habitants sont les couples de propositions (codage du $\wedge$), dans le second ce sont les couples d'objets, soit les paires de ML. Intuitivement, c'est le type sous-ensemble $\{\, x : T \mid P \,\}$ qui permet de faire des couples `Set`, `Prop` habitant `Set`. Les types $\Sigma x : U.V$ où U : `Prop` et V : `Set` n'ont pas d'intérêt dans notre cas puisqu'ils représentent des objets de type $U \wedge V$ mais on ne peut pas utiliser U dans notre système. On préfère coder ces objets par des objets de type $\{\, x : V \mid U \,\}$ (on n'est pas intéressé par la preuve de U pour programmer).

La règle COERCE formalise l'idée que l'on peut utiliser un terme de type T à la place d'un terme de type U si T et U sont dans une certaine relation. C'est là qu'interviendront les types sous-ensemble. CC contient une règle de typage similaire à COERCE, la règle de conversion (CONV), qui dit essentiellement que deux types β-convertibles (on rappelle que l'on peut calculer dans les types puisqu'on a l'abstraction, l'application, etc...) sont équivalents. On peut directement intégrer cette relation de β-convertibilité à notre système de coercion comme montré figure 2.3 ($\rhd$-CONV), à condition d'avoir l'inclusion $\equiv_{\beta\pi} \subseteq\ \rhd + \rhd$-CONV. En fait notre notion de réduction est un peu plus large que β puisqu'on peut réduire les **let** : **let** $(x, y) = (u, v)$ **in** t se réduit en $t[u/x][v/y]$. En pratique cette constructions est du sucre syntaxique pratique au niveau du typage (on peut inférer le type de t), mais elle est inessentielle au niveau du calcul. On peut aisément rajouter un **let** $x = t$ **in** v à notre langage de façon similaire : c'est équivalent à $(\lambda x : T.v)\ t$, mais T peut est inféré plutôt que donné par l'utilisateur.

On considère les constantes comme des variables prédéfinies dans nos contextes, par exemple on a la constante **list** : $\Pi x : \mathit{nat}.\texttt{Set}$. L'ajout d'une constante à un contexte ne doit pas altérer sa bonne formation comme pour le cas des variables, donc son type doit être bien formé (en général, toute définition de Coq donne lieu à une constante dans notre système si elle est bien typée).

$$\textsc{Wf-Empty}\ \frac{}{\vdash [] \ \mathbf{wf}} \qquad \textsc{Wf-Var}\ \frac{\Gamma \vdash A : s}{\vdash \Gamma, x : A \ \mathbf{wf}}\ s \in \mathcal{S} \wedge x \notin \Gamma$$

$$\textsc{Axiom}\ \frac{\vdash \Gamma \ \mathbf{wf}}{\Gamma \vdash s_1 : s_2}\ (s_1, s_2) \in \mathcal{A}$$

$$\textsc{Var}\ \frac{\vdash \Gamma \ \mathbf{wf} \qquad x : A \in \Gamma}{\Gamma \vdash x : A}$$

$$\textsc{Prod}\ \frac{\Gamma \vdash T : s_1 \qquad \Gamma, x : T \vdash U : s_2}{\Gamma \vdash \Pi x : T.U : s_3}\ (s_1, s_2, s_3) \in \mathcal{R}$$

$$\textsc{Abs}\ \frac{\Gamma \vdash \Pi x : T.U : s \qquad \Gamma, x : T \vdash M : U}{\Gamma \vdash \lambda x : T.M : \Pi x : T.U}$$

$$\textsc{App}\ \frac{\Gamma \vdash f : \Pi x : V.W \qquad \Gamma \vdash u : V}{\Gamma \vdash (f u) : W[u/x]}$$

$$\textsc{Sum}\ \frac{\Gamma \vdash T : s \qquad \Gamma, x : T \vdash U : s}{\Gamma \vdash \Sigma x : T.U : s}\ s \in \{\texttt{Prop}, \texttt{Set}\}$$

$$\textsc{Pair}\ \frac{\Gamma \vdash \Sigma x : T.U : s \qquad \Gamma \vdash t : T \qquad \Gamma \vdash u : U[t/x]}{\Gamma \vdash (t, u)_{\Sigma x:T.U} : \Sigma x : T.U}$$

$$\textsc{Pi-1}\ \frac{\Gamma \vdash t : \Sigma x : T.U}{\Gamma \vdash \pi_1\ t : T} \qquad \textsc{Pi-2}\ \frac{\Gamma \vdash t : \Sigma x : T.U}{\Gamma \vdash \pi_2\ t : U[\pi_1\ t/x]}$$

$$\textsc{Subset}\ \frac{\Gamma \vdash U : \texttt{Set} \qquad \Gamma, x : U \vdash P : \texttt{Prop}}{\Gamma \vdash \{\ x : U \mid P\ \} : \texttt{Set}}$$

$$\textsc{Coerce}\ \frac{\Gamma \vdash t : U \qquad \Gamma \vdash T : s \qquad \Gamma \vdash U \rhd T : s}{\Gamma \vdash t : T}$$

Fig. 2.2 – Calcul de coercion par prédicats - version déclarative

Jugement de coercion

Notre système de coercion par prédicats permet à l'utilisateur d'utiliser une valeur de type U là où l'on attend une valeur de type $\{\ x : V \mid P\ \}$ ($\rhd$-Proof) si U est lui-même coercible en V. A l'inverse, on permet aussi d'utiliser une valeur de type $\{\ x : U \mid P\ \}$ ($\rhd$-Subset) à la place d'une valeur de type V si U est coercible vers V. Notre jugement de coercion est donc symétrique et laisse beaucoup de liberté à l'utilisateur au moment du codage. Par exemple on peut dériver $u : \texttt{nat} \vdash u : \{\ x : \texttt{nat} \mid \bot\ \}$ Seulement, lors de la traduction de la dérivation de coercion $\texttt{nat} \rhd \{\ x : \texttt{nat} \mid \bot\ \}$ (nécessaire pour traduire l'abus de notation $x : \{\ x : \texttt{nat} \mid \bot\ \}$), l'utilisateur aura à résoudre une obligation de preuve de $\bot$. On repose donc toujours sur la cohérence du Calcul des Constructions. Les règles $\rhd$-Prod et $\rhd$-Sum permettent de faire des coercions dans les types composites. Classiquement, la règle pour le produit fonctionnel est contravariante (une fonction sous-type d'une autre accepte plus d'entrées mais donne une sortie plus fine, voir [4]) et la règle pour le produit cartésien covariante (une paire est coercible en une autre si leurs composantes sont coercibles deux-à-deux). Le sens des coercions n'a pas d'importance dans le système déclaratif puisqu'il est symétrique mais il est essentiel lors de la création des coercions que nous décrirons plus tard.

La règle $\rhd$-Trans assure que l'on a un système compositionnel. Il y a ici une analogie avec l'élimination des coupures dans les systèmes logiques, où l'on montre que toute dérivation utilisant la règle de *modus ponens* ($A \Rightarrow B$ et $B \Rightarrow C$ implique $A \Rightarrow C$) peut se réécrire en une dérivation ne l'utilisant jamais. Dans les systèmes à sous-typage, on montre de façon équivalente que l'on peut éliminer la règle de transitivité ;

$$\rhd\text{-Conv}\ \frac{\Gamma \vdash T \equiv_{\beta\pi} U : s}{\Gamma \vdash T \rhd U : s}$$

$$\rhd\text{-Sym}\ \frac{\Gamma \vdash U \rhd T : s}{\Gamma \vdash T \rhd U : s} \qquad \rhd\text{-Trans}\ \frac{\Gamma \vdash S \rhd T : s \qquad \Gamma \vdash T \rhd U : s}{\Gamma \vdash S \rhd U : s}$$

$$\rhd\text{-Prod}\ \frac{\Gamma \vdash U \rhd T : s_1 \qquad \Gamma, x : U \vdash V \rhd W : s_2}{\Gamma \vdash \Pi x : T.V \rhd \Pi x : U.W : s_2}\ (s_1, s_2) \in \mathcal{R}$$

$$\rhd\text{-Sum}\ \frac{\Gamma \vdash T \rhd U : s \qquad \Gamma, x : T \vdash V \rhd W : s}{\Gamma \vdash \Sigma x : T.V \rhd \Sigma y : U.W : s}\ s \in \{\texttt{Set}, \texttt{Prop}\}$$

$$\rhd\text{-Subset}\ \frac{\Gamma \vdash U \rhd V : \texttt{Set} \qquad \Gamma, x : U \vdash P : \texttt{Prop}}{\Gamma \vdash \{\, x : U \mid P \,\} \rhd V : \texttt{Set}}$$

$$\rhd\text{-Proof}\ \frac{\Gamma \vdash U \rhd V : \texttt{Set} \qquad \Gamma, x : V \vdash P : \texttt{Prop}}{\Gamma \vdash U \rhd \{\, x : V \mid P \,\} : \texttt{Set}}$$

Fig. 2.3 – Coercion par prédicats - version déclarative

s_1	s_2	$s_3 = s_2$	Trait
Set	Set	Set	Dependent arrow
Type	Set	Set	Polymorphism
Set	Type	Type	Dependent types
Set	Prop	Prop	Terms in propositions
Prop	Prop	Prop	Implication
Type	Prop	Prop	Impredicativity

Fig. 2.4 – Définition de $\mathcal{R}$

première étape vers un système décidable.

Notre jugement de coercion identifie les types U et $\{\, x : U \mid P \,\}$ mais notre système de typage ne permet pas d'éliminer (prendre la partie preuve) ou d'introduire (créer un couple témoin,preuve) des objets de type sous-ensemble. Cela nous assure une certaine cohérence, puisque même si l'on ne vérifie pas qu'un objet de type U a bien la propriété P, on ne peut pas raisonner sur le fait que U a la propriété dans le langage.

On ne fera pas la métathéorie du système déclaratif ici, puisque c'est une extension conservative du Calcul des Constructions et l'on étudiera en détail le système algorithmique. Notre preuve de conservativité est simple : si l'on oublie les utilisations des types sous-ensemble de notre système de typage (Subset) et de coercion ($\rhd$-Proof et $\rhd$-Subset), alors le jugement de coercion est juste la β-convertibilité et donc Coerce et Conv sont équivalentes. Comme les autres règles de notre système déclaratif proviennent directement de Cc, on arrive à un système strictement égal au système du calcul des constructions. On peut donc s'appuyer sur les résultats connus pour Cc pour une partie de notre système.

Pour une étude complète du Calcul des Constructions, se référer à [3, 10]. On va plutôt s'intéresser à la construction d'un algorithme de typage correspondant à notre système déclaratif.

2.2 Élaboration du système algorithmique et propriétés

Pour pouvoir implanter le typeur, il nous faut un système dirigé par la syntaxe. Ce n'est pas le cas du système déclaratif, aussi bien pour le typage que pour la coercion. On va donc raffiner ces systèmes dans l'optique d'en extraire un algorithme. On note $\vdash_\bullet$ le jugement de typage algorithmique défini figure 2.5 page 15 et $\rhd_\bullet$ le jugement de coercion algorithmique présenté figure 2.7. Ces deux systèmes sont quelque peu éloignés des originaux, néanmoins nous allons montrer qu'ils sont corrects et complets vis-à-vis de leurs géniteurs. La correction (si l'on a une dérivation d'un jugement dans le système algorithmique, alors c'est un jugement valide du système déclaratif) est le sens le plus facile à montrer, nous allons donc commencer par là. On décrira ensuite la méthode de construction des systèmes algorithmiques pour aboutir d'une part au théorème de complétude qui montre qu'on peut dériver les mêmes jugements (à coercion près) dans le système algorithmique que dans le sytème déclaratif et d'autre part à la décidabilité du jugement de typage algorithmique.

Il nous a fallu changer quelque peu les règles pour obtenir le système algorithmique. En particulier, on a utilisé la fonction μ_0 de PVS [12] renommée $\mu_\bullet()$ (figure 2.6) ici pour opérer des *décompréhensions*. Cette fonction efface les constructeurs de type sous-ensemble en tête d'un type, par exemple : $\mu_\bullet(\{\ f : \texttt{nat} \to \texttt{nat} \mid f\ 0 = 0\ \}) = \texttt{nat} \to \texttt{nat}$. On verra son utilité dans la suite. Notons aussi que les jugements de la forme $\Gamma \vdash_\bullet t : s$ où $s \in \{\texttt{Set}, \texttt{Prop}, \texttt{Type}\}$ sont une abréviation pour $\Gamma \vdash_\bullet t : T \wedge T \to \beta\rho s$, c'est une pratique courante lorsque l'on présente un système algorithmique basé sur système de types purs.

2.2.1 Notations

On note $x^\downarrow$ la mise en forme normale de tête de x selon la réduction définie précédemment. On note $\hat{=}$ l'égalité définitionelle. On omet le contexte pour le jugement de coercion lorsqu'il est dérivable du contexte.

Voici quelques propriétés élémentaires du système :

Lemme 2.2.1 (Bonne formation des contextes). *Si $\Gamma \vdash_\bullet t : T$ alors $\vdash \Gamma$.*

Démonstration. Par induction sur la dérivation de typage. □

Fait 2.2.2 (Inversion du jugement de bonne formation). *Si $\vdash \Gamma, x : U$ alors il existe s, $\Gamma \vdash_\bullet U : s$ et $s \in \{\texttt{Set}, \texttt{Prop}, \texttt{Type}\}$.*

Lemme 2.2.3 (Affaiblissement). *Si $\Gamma, \Delta \vdash_\bullet t : T$ alors pour tout $x : S \notin \Gamma, \Delta$ tel que $\vdash_\bullet \simeq \mathit{wf} G, x : S, \Delta$ on a $\Gamma, x : S, \Delta \vdash_\bullet t : T$*

Démonstration. Par induction sur la dérivation de typage.

- PropSet : Trivial.
- Var : On a $x : S \notin \Gamma, \Delta$, donc $\Gamma, x : S, \Delta \vdash_\bullet y : T$ est toujours dérivable.
- Prod : Par induction $\Gamma, x : S, \Delta \vdash_\bullet T : s1$ et $\Gamma, x : S, \Delta, y : T \vdash_\bullet U : s2$. On applique Prod pour obtenir $\Gamma, x : S, \Delta \vdash_\bullet \Pi x : T.U : s2$. De même pour le reste des règles.

□

2.2.2 Correction

On montre tout d'abord la correction de la coercion algorithmique qui sera nécessaire pour la correction du typage :

Théorème 2.2.4 (Correction de la coercion). *Si $\Gamma \vdash_\bullet U \rhd_\bullet V :$ alors $U \rhd V$.*

Démonstration. Les règles du système algorithmique sont un sous-ensemble des règles du système déclaratif, excepté pour la règle $\rhd$-$\downarrow$. On utilise $\rhd$-Conv et $\rhd$-Trans pour montrer son admissibilité dans dans le système déclaratif.

$$\textsc{Wf-Empty}\ \frac{}{\vdash_\bullet [] \ \mathbf{wf}} \qquad \textsc{Wf-Var}\ \frac{\Gamma \vdash_\bullet A : s}{\vdash_\bullet \Gamma, x : A \ \mathbf{wf}}\ s \in \mathcal{S} \wedge x \notin \Gamma$$

$$\textsc{Axiom}\ \frac{\vdash_\bullet \Gamma \ \mathbf{wf}}{\Gamma \vdash_\bullet s_1 : s_2}\ (s_1, s_2) \in \mathcal{A}$$

$$\textsc{Var}\ \frac{\vdash_\bullet \Gamma \ \mathbf{wf} \qquad x : A \in \Gamma}{\Gamma \vdash_\bullet x : A}$$

$$\textsc{Prod}\ \frac{\Gamma \vdash_\bullet T : s_1 \qquad \Gamma, x : T \vdash_\bullet U : s_2}{\Gamma \vdash_\bullet \Pi x : T.U : s_3}\ (s_1, s_2, s_3) \in \mathcal{R}$$

$$\textsc{Abs}\ \frac{\Gamma \vdash_\bullet \Pi x : T.U : s \qquad \Gamma, x : T \vdash_\bullet M : U}{\Gamma \vdash_\bullet \lambda x : T.M : \Pi x : T.U}$$

$$\textsc{App}\ \frac{\Gamma \vdash_\bullet f : T \quad \mu_\bullet(T) = \Pi x : V.W : s \qquad \Gamma \vdash_\bullet u : U \qquad \Gamma \vdash_\bullet U \rhd_\bullet V : s'}{\Gamma \vdash_\bullet (f u) : W[u/x]}$$

$$\textsc{Sum}\ \frac{\Gamma \vdash_\bullet T : s \qquad \Gamma, x : T \vdash_\bullet U : s}{\Gamma \vdash_\bullet \Sigma x : T.U : s}\ s \in \{\texttt{Prop}, \texttt{Set}\}$$

$$\textsc{Pair}\ \frac{\Gamma \vdash_\bullet \Sigma x : T.U : s \qquad \Gamma \vdash_\bullet u : U' \quad \Gamma \vdash_\bullet t : T' \quad \Gamma \vdash_\bullet T' \rhd_\bullet T : s \quad \Gamma \vdash_\bullet U' \rhd_\bullet U[t/x] : s}{\Gamma \vdash_\bullet (t, u)_{\Sigma x:T.U} : \Sigma x : T.U}$$

$$\textsc{Pi-1}\ \frac{\Gamma \vdash_\bullet t : S \qquad \mu_\bullet(S) = \Sigma x : T.U}{\Gamma \vdash_\bullet \pi_1\ t : T} \qquad \textsc{Pi-2}\ \frac{\Gamma \vdash_\bullet t : S \qquad \mu_\bullet(S) = \Sigma x : T.U}{\Gamma \vdash_\bullet \pi_2\ t : U[\pi_1\ t/x]}$$

$$\textsc{Subset}\ \frac{\Gamma \vdash_\bullet U : \texttt{Set} \qquad \Gamma, x : U \vdash_\bullet P : \texttt{Prop}}{\Gamma \vdash_\bullet \{\, x : U \mid P \,\} : \texttt{Set}}$$

Fig. 2.5 – Calcul de coercion par prédicats - version algorithmique

$$\frac{\dfrac{U \equiv_{\beta\pi} U^\downarrow}{U \rhd U^\downarrow} \qquad \dfrac{U^\downarrow \rhd T^\downarrow \qquad \dfrac{T^\downarrow \equiv_{\beta\pi} T}{T^\downarrow \rhd T}}{U^\downarrow \rhd T}}{U \rhd T}$$

□

On a besoin d'un lemme sur l'opération $\mu_\bullet()$ définie figure 2.6.

Lemme 2.2.5 ($\mu_\bullet()$ et coercion). *Si* $\Gamma \vdash_\bullet T : s$ *alors* $\Gamma \vdash_\bullet T \rhd_\bullet \mu_\bullet(T)$:.

Démonstration. Il suffit de suivre la définition de $\mu_\bullet()$. La mise en forme normale de tête est équivalente à l'utilisation de $\rhd$-$\downarrow$ dans notre jugement de coercion. $\mu_\bullet()$ est en fait l'application répétée de la règle $\rhd$-Subset. □

Lemme 2.2.6 (Conservation des sortes par $\mu_\bullet()$). *Si* $\Gamma \vdash_\bullet S : s$ *alors* $\Gamma \vdash_\bullet \mu_\bullet(S) : s$

Démonstration. Par le simple fait que si $S = \{\, x : U \mid P \,\}$ et $G \vdash_\bullet S : s$ alors $S : \texttt{Set}$ et $U : \texttt{Set}$ (par Subset), sinon $S = \mu_\bullet(S)$. □

Théorème 2.2.7 (Correction du typage). *Si* $\Gamma \vdash_\bullet t : T$ *alors* $\Gamma \vdash t : T$

Démonstration. Par induction sur la dérivation de typage dans le système algorithmique.

$$\begin{array}{lcl} \mu'_\bullet(\{\, x : U \mid P \,\}) & \Rightarrow & \mu'_\bullet(\downarrow U) \\ \mu'_\bullet(x) \mid x \neq \{\, x : U \mid P \,\} & \Rightarrow & x \\ \\ \mu_\bullet(x) & \Rightarrow & \mu'_\bullet(\downarrow x) \end{array}$$

Fig. 2.6 – Définition de $\mu_\bullet()$

$$\rhd\text{-Conv}\ \frac{T \equiv_{\beta\pi} U \qquad \Gamma \vdash_\bullet T, U : s}{\Gamma \vdash_\bullet T \rhd_\bullet U : s}\ T = T^\downarrow \wedge T \neq \Pi, \Sigma, \{|\} \wedge U = U^\downarrow$$

$$\rhd\text{-}\downarrow\ \frac{\Gamma \vdash_\bullet T^\downarrow \rhd_\bullet U^\downarrow : s \qquad \Gamma \vdash_\bullet T, U : s}{\Gamma \vdash_\bullet T \rhd_\bullet U : s}\ T \neq T^\downarrow \vee U \neq U^\downarrow$$

$$\rhd\text{-Prod}\ \frac{\Gamma \vdash_\bullet U \rhd_\bullet T : s_1 \qquad \Gamma, x : U \vdash_\bullet V \rhd_\bullet W : s_2}{\Gamma \vdash_\bullet \Pi x : T.V \rhd_\bullet \Pi x : U.W : s_2}\ (s_1, s_2) \in \mathcal{R}$$

$$\rhd\text{-Sum}\ \frac{\Gamma \vdash_\bullet T \rhd_\bullet U : s \qquad \Gamma, x : T \vdash_\bullet V \rhd_\bullet W : s}{\Gamma \vdash_\bullet \Sigma x : T.V \rhd_\bullet \Sigma x : U.W : s}\ s \in \{\texttt{Set}, \texttt{Prop}\}$$

$$\rhd\text{-Proof}\ \frac{\Gamma \vdash_\bullet T \rhd_\bullet U : \texttt{Set}}{\Gamma \vdash_\bullet T \rhd_\bullet \{\, x : U \mid P \,\} : \texttt{Set}}\ T = T^\downarrow$$

$$\rhd\text{-Subset}\ \frac{\Gamma \vdash_\bullet U \rhd_\bullet T : \texttt{Set} \qquad \Gamma, x : U \vdash_\bullet P : \texttt{Prop}}{\Gamma \vdash_\bullet \{\, x : U \mid P \,\} \rhd_\bullet T : \texttt{Set}}\ T = T^\downarrow$$

Fig. 2.7 – Coercion par prédicats - version algorithmique

– Wf-Empty,Wf-Var,PropSet,Var,Prod,Abs, Sum, Sum : règles inchangées.

– App : On a

$$\frac{\Gamma \vdash_\bullet f : T \quad \mu_\bullet(T) = \Pi x : V.W : s \qquad \Gamma \vdash_\bullet u : U \qquad \Gamma \vdash_\bullet U \rhd_\bullet V : s'}{\Gamma \vdash_\bullet (fu) : W[u/x]}$$

Par induction, $\Gamma \vdash f : T$, et $T \rhd \Pi x : V.W$ par le lemme 2.2.5 et la correction de la coercion. On peut donc dériver $\Gamma \vdash f : \Pi x : V.W$ à l'aide de la règle Coerce. Par le lemme 2.2.4 appliqué à $\Gamma \vdash_\bullet U \rhd_\bullet V :$ et l'hypothèse $\Gamma \vdash u : U$, on obtient $\Gamma \vdash u : V$ par Coerce. Donc, par App, on a bien $\Gamma \vdash fu : W[u/x]$.

– Pair : On a

$$\frac{\Gamma \vdash_\bullet \Sigma x : T.U : s \qquad \begin{array}{c} \Gamma \vdash_\bullet t : T' \quad \Gamma \vdash_\bullet T' \rhd_\bullet T : s \\ \Gamma \vdash_\bullet u : U' \quad \Gamma \vdash_\bullet U' \rhd_\bullet U[t/x] : s \end{array}}{\Gamma \vdash_\bullet (t, u)_{\Sigma x:T.U} : \Sigma x : T.U}$$

Par induction et correction de la coercion, on peut dériver : $\Gamma \vdash t : T$ et $\Gamma \vdash u : U[t/x]$. On a $\Gamma \vdash \Sigma x : T.U : s$, on peut donc appliquer Pair

– Pi-2 : On a

$$\frac{\Gamma \vdash_\bullet t : S \qquad \mu_\bullet(S) = \Sigma x : T.U}{\Gamma \vdash_\bullet \pi_2\ t : U[\pi_1\ t/x]}$$

Par induction, $\Gamma \vdash t : S$, et par le lemme 2.2.5 et la correction de la coercion $S \rhd \Sigma x : T.U$. On peut donc dériver $\Gamma \vdash t : \Sigma x : T.U$ à l'aide de Coerce. On peut directement appliquer Pi-2 à cette prémisse pour obtenir $\Gamma \vdash \pi_2\ t : U[\pi_1\ t/x]$. De même pour Pi-1.

□

On a prouvé que notre système algorithmique était correct, c'est-à-dire que ses jugements valides sont bien inclus dans ceux du système déclaratif, il faut maintenant montrer qu'il les inclut tous (ou presque !).

Notations

On introduit la notation $\Gamma \vdash_\bullet T, U : s$ pour $\Gamma \vdash_\bullet T : s \wedge \Gamma \vdash_\bullet U : s$.

2.2.3 Complétude et décidabilité

On va maintenant repartir des systèmes déclaratifs pour montrer comment l'on a construit les systèmes algorithmiques.

On s'intéresse tout d'abord au jugement de coercion. Pour rendre le jugement de coercion décidable, il faut traiter les règles $\rhd$-Conv et $\rhd$-$\downarrow$ qu'on peut appliquer à n'importe quel moment et la règle $\rhd$-Trans qui n'est pas dirigée par la syntaxe (il faut "deviner" un type T). Le système de coercion algorithmique (figure 2.7) est le même que le système déclaratif (figure 2.3) mais où l'on n'applique $\rhd$-Conv seulement si aucune autre règle ne s'applique après avoir appliqué $\rhd$-$\downarrow$ et sans la règle $\rhd$-Trans.

Décidabilité et complétude de la coercion

On va montrer que les deux systèmes de coercion sont équivalents vis-à-vis de la conversion. On montrera plus tard pourquoi on peut éliminer la règle de transitivité.

Il nous faut tout d'abord des lemmes d'inversion sur la conversion :

Lemme 2.2.8. *Si $\Pi x : T.U \equiv_{\beta\pi} S$ alors $S^\downarrow = \Pi x : T'.U'$ avec $T \equiv_{\beta\pi} T'$ et $U \equiv_{\beta\pi} U'$.*

Lemme 2.2.9. *Si $\Sigma x : T.U \equiv_{\beta\pi} S$ alors $S^\downarrow = \Sigma x : T'.U'$ avec $T \equiv_{\beta\pi} T'$ et $U \equiv_{\beta\pi} U'$.*

Lemme 2.2.10 (Coercion de sortes). *Si $\Gamma \vdash_\bullet T \rhd_\bullet s :$ où $\Gamma \vdash_\bullet s \rhd_\bullet T :$ où $s \in \{\texttt{Set}, \texttt{Prop}, \texttt{Type}\}$ alors $T \equiv_{\beta\pi} s$.*

Démonstration. Les seuls règles permettant de dériver de tels jugements sont $\rhd$-Conv, $\rhd$-$\downarrow$ et $\rhd$-Proof ou $\rhd$-Subset.

- $\rhd$-Conv : Trivial.
- $\rhd$-$\downarrow$: Par induction.
- $\rhd$-Proof, $\rhd$-Subset : On ne peut pas avoir un jugement de la forme $\Gamma \vdash_\bullet s \rhd_\bullet \{ x : U \mid P \}$: puisque cela impliquerait que $\Gamma \vdash_\bullet s, U : s'$ avec $\Gamma \vdash_\bullet U : \texttt{Set}$ donc $\Gamma \vdash_\bullet s : \texttt{Set}$ ce qui est impossible. De façon symétrique pour $\rhd$-Subset.

□

Lemme 2.2.11 (Convertibilité avec une sorte et réduction). *Si $T \equiv_{\beta\pi} s$ alors $T \to_{\beta\rho} s$.*

Démonstration. Par la propriété de Church-Rosser pour la réduction $\to_{\beta\rho}$ il existe v tel que $T \twoheadrightarrow_{\beta\rho} v \twoheadleftarrow_{\beta\rho} s$. Or s ne peut se réduire pas vers un autre terme, on a donc $T \twoheadrightarrow_{\beta\rho} s$. □

Corollaire 2.2.12 (Coercion de sortes et réduction). *Si $\Gamma \vdash_\bullet T \rhd_\bullet s :$ ou $\Gamma \vdash_\bullet s \rhd_\bullet T :$ où $s \in \{\texttt{Set}, \texttt{Prop}, \texttt{Type}\}$ alors $T \twoheadrightarrow_{\beta\pi} s$.*

L'affaiblissement montre que notre notion de coercion joue un rôle similaire à la seule conversion vis-à-vis du typage. On peut dériver les mêmes jugements dans des contextes où les variables ont des types équivalents par la coercion. Ici la taille des dérivations ne change pas.

Lemme 2.2.13 (Affaiblissement).

$$\Gamma \vdash_\bullet S, S' : s, S' \rhd_\bullet S \Rightarrow \begin{cases} \vdash \Gamma, x : S, \Delta & \Rightarrow \quad \vdash \Gamma, x : S', \Delta \\ \Gamma, x : S, \Delta \vdash_\bullet t : T & \Rightarrow \quad \Gamma, x : S', \Delta \vdash_\bullet t : T' \rhd_\bullet T \\ \Gamma, x : S, \Delta \vdash_\bullet T \rhd_\bullet T' & \Rightarrow \quad \Gamma, x : S', \Delta \vdash_\bullet T \rhd_\bullet T' \end{cases}$$

Démonstration. Par induction sur mutuelle sur les dérivations de typage, coercion et bonne formation.

– Wf-Empty : Trivial.

– Wf-Var : La conclusion est $\vdash \Gamma, x : S, \Delta$

- $\Delta = []$: La racine de la dérivation est de la forme :

$$\frac{\Gamma \vdash_\bullet S : s}{\vdash_\bullet \simeq \mathbf{wf} G, x : S} \; s \in \{\texttt{Set}, \texttt{Prop}, \texttt{Type}\}$$

  On a donc $\Gamma \vdash_\bullet S' : s$, et par Wf-Var, $\vdash \Gamma, x : S'$.

- $\Delta \equiv \Delta', y : U$: La racine de la dérivation est de la forme :

$$\frac{\Gamma, x : S, \Delta' \vdash_\bullet U : s}{\vdash_\bullet \simeq \mathbf{wf} G, x : S, \Delta', y : U} \; s \in \{\texttt{Set}, \texttt{Prop}, \texttt{Type}\}$$

  Par induction sur la dérivation de typage $\Gamma, x : S', \Delta' \vdash_\bullet U : s$, on a donc bien $\vdash \Gamma, x : S', \Delta', y : U$ par Wf-Var.

– PropSet : Par induction, $\vdash \Gamma, x : S', \Delta$, on applique simplement la règle.

– Var : Par induction, $\vdash \Gamma, x : S', \Delta$. La seule différence avec le contexte précédent est le type associé à x, donc si $t \neq x$, on peut simplement réappliquer Var. Si $t \equiv x$ on a la dérivation :

$$\frac{\vdash_\bullet \simeq \mathbf{wf} G, x : S', \Delta \qquad x : S' \in \Gamma}{\Gamma, x : S', \Delta \vdash_\bullet x : S'}$$

On a bien $S' \rhd_\bullet S$, la propriété est donc bien vérifiée.

– Prod : Par induction, $\Gamma, x : S', \Delta \vdash_\bullet T : V \rhd_\bullet s_1$ et $\Gamma, x : S', \Delta y : T \vdash_\bullet U : W \rhd_\bullet s_2$. On a donc $V \twoheadrightarrow_{\beta\pi} s_1$ et $W \twoheadrightarrow_{\beta\pi} s_2$. On en déduit $\Gamma, x : S', \Delta \vdash_\bullet T : s_1$ et $\Gamma, x : S', \Delta y : T \vdash_\bullet U : s_2$. On applique Prod pour obtenir $\Gamma, x : S', \Delta \vdash_\bullet \Pi x : T.U : s_3$. De même, direct par induction pour le reste des règles.

– $\rhd$-Prod, $\rhd$-Sum : Par induction on a $\Gamma, x : S', \Delta \vdash_\bullet U \rhd_\bullet T :$ et $\Gamma, x : S', \Delta, x : U \vdash_\bullet V \rhd_\bullet W :$, il suffit d'appliquer $\rhd$-Prod. De même pour $\rhd$-Sum.

– $\rhd$-Conv,$\rhd$-$\downarrow$,$\rhd$-Proof,$\rhd$-Subset : De même, direct par induction. On utilise l'hypothèse d'induction mutuelle pour les dérivations de typage en prémisse.

□

On peut maintenant montrer :

Lemme 2.2.14 (Conservation de la conversion par coercion). *Si* $\Gamma \vdash_\bullet T, U : s$ *et* $T \equiv_{\beta\pi} U$ *alors* $\Gamma \vdash_\bullet T \rhd_\bullet U :$.

Démonstration. Par induction sur le nombre de constructeurs $\Pi, \Sigma, \{|\}$ dans la forme normale complète de T.

– $T^\downarrow = \Pi x : X.Y$: Alors $U^\downarrow = \Pi x : V.W$ et $X \equiv_{\beta\pi} V$, $Y \equiv_{\beta\pi} W$ d'après le lemme 2.2.8. On a $\Gamma, x : X \vdash_\bullet Y : s$ et $\Gamma, x : V \vdash_\bullet W : s$. Par induction $\Gamma \vdash_\bullet V \rhd_\bullet X :$. On peut donc appliquer le lemme 2.2.13 pour obtenir $\Gamma, x : V \vdash_\bullet Y, W : s$. On applique l'hypothèse d'induction pour obtenir $\Gamma, x : V \vdash_\bullet Y \rhd_\bullet W :$. On applique alors $\rhd$-Prod à ces deux prémisses.

– $T^\downarrow = \Sigma x : X.Y$: Alors $U^\downarrow = \Sigma x : V.W$, avec $X \equiv_{\beta\pi} V$ et $Y \equiv_{\beta\pi} W$. Par induction et application de $\rhd$-Sum en utilisant le lemme 2.2.13 pour la deuxième prémisse.

– $T^\downarrow \equiv \{ x : X \mid P \}$: On a alors $U^\downarrow = \{ x : X' \mid P' \}$ avec $X \equiv_{\beta\pi} X'$, $P \equiv_{\beta\pi} P'$, et la propriété est vraie par $\rhd$-Left et $\rhd$-Right :

$$\rhd\text{-Right}\ \frac{\rhd\text{-Left}\ \dfrac{X \rhd_\bullet X'}{\{ x : X \mid P \} \rhd_\bullet X'}}{\{ x : X \mid P \} \rhd_\bullet \{ x : X' \mid P' \}}$$

– Sinon : On applique obligatoirement $\rhd$-Conv et l'on a la prémisse $T \equiv_{\beta\pi} U$, c'est donc direct.

□

Il n'y a pas de problème d'identification de sortes dans ce système, contrairement au système $\lambda\, C_{\leq}$ de Gang Chen [5], puisqu'on n'a pas de cumulativité. Par exemple on n'a pas besoin d'identifier `nat : Set` et $(\lambda x : \texttt{Type}.x)\texttt{nat} : \texttt{Type}$ puisque le deuxième terme n'est pas typable dans notre système.

On va maintenant montrer que la règle $\rhd$-Trans est admissible dans notre système algorithmique. On montre ceci en l'éliminant de toute dérivation possible la faisant intervenir.

Tout d'abord quelques lemmes nécessaires pour la preuve :

Lemme 2.2.15 (Coercion et $\mu_\bullet()$).
– *Si* $\Pi x : X.Y \rhd_\bullet U$ *alors* $\mu_\bullet(U) = \Pi x : X'.Y'$ *et* $X' \rhd_\bullet X$, $Y \rhd_\bullet Y'$.
– *Si* $\Sigma x : X.Y \rhd_\bullet U$ *alors* $\mu_\bullet(U) = \Sigma x : X'.Y'$ *et* $X \rhd_\bullet X'$, $Y \rhd_\bullet Y'$.
– *Pour tout* S, U, $S \rhd_\bullet \mu_\bullet(U)$ *si et seulement si* $S \rhd_\bullet U$.

Démonstration. Par induction sur les dérivations de $\rhd_\bullet$ et la définition de $\mu_\bullet()$.

Dans le dernier cas, de gauche à droite on construit la dérivation en ajoutant des applications de $\rhd$-Proof et dans l'autre sens on est assuré de trouver la preuve dans la dérivation même de $S \rhd_\bullet U$: si U n'est pas de la forme sous-ensemble c'est direct. Sinon, on peut trouver dans la preuve (en partant de la racine) la première utilisation de la règle $\rhd$-Proof. A partir de là, on cherche la première utilisation d'une règle autre que $\rhd$-Proof ou $\rhd$-Subset. On a une dérivation de $S' \rhd_\bullet \mu_\bullet(U)$, on peut réappliquer les règles $\rhd$-Subset oubliées précédemment pour obtenir la preuve de $S \rhd_\bullet \mu_\bullet(U)$. □

Lemme 2.2.16 (Coercion et conversion). *Si* $\Gamma \vdash_\bullet S, T, U : s$, $S \equiv_{\beta\pi} T$ *et* $T \rhd_\bullet U$ *alors* $\Gamma \vdash_\bullet S \rhd_\bullet U$:

Démonstration. Par simple inspection des règles on voit que le jugement ne peut distinguer deux termes β-équivalents (ils ont forcément les mêmes constructeurs de tête appliqués à des termes équivalents). □

Lemme 2.2.17 (Coercion et formes normales de tête). *Si* $T \rhd_\bullet U$ *alors* $T^\downarrow \rhd_\bullet U^\downarrow$ *est dérivable par une dérivation plus petite ou égale.*

Démonstration. Par induction sur la dérivation de sous-typage dans le système algorithmique.

– $\rhd$-Conv : Trivial.
– $\rhd$-$\downarrow$: On prend la dérivation en prémisse.
– $\rhd$-Prod, $\rhd$-Sum : T et U sont égaux à leurs formes normales de tête, direct.
– $\rhd$-Proof : Par induction $T^\downarrow \rhd_\bullet V^\downarrow$, on applique $\rhd$-Proof
– $\rhd$-Subset : idem.

□

Lemme 2.2.18 (Transitivité de la coercion). *Pour tout* S, T, U *tel que* $\Gamma \vdash_\bullet S, T, U : s$ *si* $S \rhd_\bullet T$ *et* $T \rhd_\bullet U$ *alors* $S \rhd_\bullet U$.

Démonstration. On procède par élimination de la règle $\rhd$-Trans dans toute dérivation de $S \rhd_\bullet U$. Par induction sur l'ordre lexicographique $< \text{depth}(S \rhd_\bullet T), \text{depth}(T \rhd_\bullet U) >$. On peut supposer qu'il n'y a pas d'applications successive de la règle $\rhd$-$\downarrow$ dans nos dérivations, par idempotence de la mise en forme normale de tête et les conditions $T = T^\downarrow \wedge U = U^\downarrow$ de cette règle.

– $\rhd$-Conv, _ :

$$\dfrac{\dfrac{S \equiv_{\beta\pi} T}{S \rhd_\bullet T} \qquad T \rhd_\bullet U}{S \rhd_\bullet U}$$

Par le lemme 2.2.16, on élimine trivialement $\rhd$-Trans.

– $\rhd$-$\downarrow$, _ :

$$\dfrac{\dfrac{S^{\downarrow} \rhd_\bullet T^{\downarrow}}{S \rhd_\bullet T} \qquad T \rhd_\bullet U}{S \rhd_\bullet U}$$

Par le lemme 2.2.17, il existe une dérivation de $T^{\downarrow} \rhd_\bullet U^{\downarrow}$ de taille plus petite ou égale à la dérivation de $T \rhd_\bullet U$ on peut donc se ramener au cas :

$$\dfrac{\dfrac{S^{\downarrow} \rhd_\bullet T^{\downarrow}}{S \rhd_\bullet T} \qquad \dfrac{T^{\downarrow} \rhd_\bullet U^{\downarrow}}{T \rhd_\bullet U}}{S \rhd_\bullet U}$$

Par application de l'hypothèse d'induction on a une dérivation de $S^{\downarrow} \rhd_\bullet U^{\downarrow}$ donc de $S \rhd_\bullet U$ par $\rhd$-$\downarrow$. On peut faire le même raisonnement si la dérivation de $T \rhd_\bullet U$ se termine par une application de $\rhd$-$\downarrow$. On peut donc se restreindre aux cas où l'on n'utilise ni la règle $\rhd$-$\downarrow$ ni la règle $\rhd$-Conv dans les prémisses.

– $\rhd$-Prod :

$$\dfrac{\dfrac{C \rhd_\bullet A \qquad B \rhd_\bullet D}{\Pi x : A.B \rhd_\bullet \Pi x : C.D} \qquad \Pi x : C.D \rhd_\bullet U}{\Pi x : A.B \rhd_\bullet U}$$

On n'a seulement à traiter le cas ou $\Pi x : C.D \rhd_\bullet U$ est dérivé par $\rhd$-Prod ou $\rhd$-Proof.

- $\rhd$-Prod : Alors on a

  $$\dfrac{E \rhd_\bullet C \qquad D \rhd_\bullet F}{\Pi x : C.D \rhd_\bullet \Pi x : E.F}$$

  On a donc la dérivation :

  $$\dfrac{\dfrac{E \rhd_\bullet C \qquad C \rhd_\bullet A}{E \rhd_\bullet A} \qquad \dfrac{B \rhd_\bullet D \qquad D \rhd_\bullet F}{B \rhd_\bullet F}}{S = \Pi x : A.B \rhd_\bullet \Pi x : E.F = U}$$

  La taille des dérivations de $E \rhd_\bullet C$, $C \rhd_\bullet A$ et $B \rhd_\bullet D$, $D \rhd_\bullet F$ étant plus petites que $\Pi x : A.B \rhd_\bullet \Pi x : C.D$ et $\Pi x : C.D \rhd_\bullet \Pi x : E.F$, on élimine bien la transitivité dans ce cas.

- $\rhd$-Proof : On a :

  $$\dfrac{\Pi x : C.D \rhd_\bullet E}{\Pi x : C.D \rhd_\bullet \{\, y : E \mid P \,\}}$$

  Par induction, on a :

  $$\dfrac{\dfrac{\Pi x : A.B \rhd_\bullet \Pi x : C.D \qquad \Pi x : C.D \rhd_\bullet E}{\Pi x : A.B \rhd_\bullet E}}{S = \Pi x : A.B \rhd_\bullet \{\, y : E \mid P \,\} = U}$$

  Car $\Pi x : C.D \rhd_\bullet E$ est une dérivation plus petite que $T \rhd_\bullet U$.

– $\rhd$-Sum : De façon équivalente au produit.

– $\rhd$-Proof :

$$\frac{\dfrac{S \rhd_\bullet C}{S \rhd_\bullet T = \{\, y : C \mid P \,\}} \qquad T \rhd_\bullet U}{S \rhd_\bullet U}$$

Encore une fois, par cas sur la dérivation de $\{\, y : C \mid P \,\} = T \rhd_\bullet U$:

- $\rhd$-Conv : Trivial.
- $\rhd$-Subset : On a :

  $$\frac{C \rhd_\bullet U}{\{\, y : C \mid P \,\} = T \rhd_\bullet U}$$

  Par induction, on obtient directement $S \rhd_\bullet U$ avec les dérivations de $S \rhd_\bullet C$ et $C \rhd_\bullet U$.
- $\rhd$-Proof : On a :

  $$\frac{T \rhd_\bullet D}{\{\, y : C \mid P \,\} = T \rhd_\bullet \{\, y : D \mid Q \,\} = U}$$

  On a donc une dérivation de $S \rhd_\bullet D$ par application de l'hypothèse d'induction. On en déduit une dérivation de $S \rhd_\bullet \{\, y : D \mid Q \,\} = U$ par $\rhd$-Proof.

– $\rhd$-Subset : De même.

□

Corollaire 2.2.19 (Complétude de la coercion). *Si* $\Gamma \vdash_\bullet U, V : s$, $U \rhd V$ *alors* $\Gamma \vdash_\bullet U \rhd_\bullet V$:.

Démonstration. Les règles des deux systèmes sont les mêmes excepté $\rhd$-Trans qui est admissible dans le système algorithmique. De plus l'application restreinte de la conversion ne change pas les jugements dérivables (lemme 2.2.14). Le fait de forcer les types à être bien sortés assure juste que l'on a les mêmes relations puisque le sortage est inclus dans la coercion algorithmique mais pas déclarative. □

En conséquence $\rhd$ et $\rhd_\bullet$ sont équivalentes. Le système d'inférence de $\rhd_\bullet$ donne donc un algorithme pour décider de la relation de coercion. L'indéterminisme entre les règles $\rhd$-Proof et $\rhd$-Subset ne pose pas de problème : on peut laisser le choix à l'implantation puisque le système est confluent. $\rhd$-$\downarrow$ formalise le fait qu'on peut avoir à réduire en tête avant d'appliquer les autres règles (pour obtenir un produit, une somme ou un sous-ensemble).

Décidabilité et complétude du typage

Le système algorithmique correspond au système déclaratif où l'on a enlevé la règle de coercion Coerce et changé certaines règles pour obtenir un système décidable (voir figure 2.5). On va procéder de façon similaire à l'élimination de la transitivité pour montrer que la règle Coerce n'est plus nécessaire dans le système algorithmique. On va montrer en fait que toute dérivation de typage utilisant Coerce peut se réécrire en une dérivation n'utilisant cette règle qu'à sa racine.

Élimination de la coercion On veut maintenant montrer la complétude de notre système. Dans un système à sous-typage, le théorème correspondant est parfois nommé typage minimal "*minimal typing*" puisque son énoncé revient à dire que tout terme a un type minimal dans les deux systèmes. En effet notre théorème est le suivant : $\Gamma \vdash t : T \Rightarrow \Gamma \vdash_\bullet t : U \rhd_\bullet T$. Le typage algorithmique assigne bien un seul type à un terme t mais comme on a des coercions, le type inféré U peut être un peu différent du type T. Dans notre cas particulier U sera peut-être un type moins riche que T (par exemple `nat` par rapport à $\{\, x : \texttt{nat} \mid P \,\}$). Lorsque l'on développera des programmes, on donnera une spécification forte et l'on fera une coercion entre le type inféré et la spécification pour obtenir au final (après réécriture dans Coq) un terme du type T le plus riche. On a besoin de quelques lemmes pour montrer que notre système où la règle Coerce a été éliminée est complet :

Lemme 2.2.20 ($\mu_\bullet()$ et types produits et sommes). *Si $X \rhd_\bullet Y$ et $\mu_\bullet(Y) = \Sigma x : T.U$ alors $\mu_\bullet(X) = \Sigma x : T'.U'$ et $T' \rhd_\bullet T$, $U' \rhd_\bullet U$. Si $X \rhd_\bullet Y$ et $\mu_\bullet(Y) = \Pi x : T.U$ alors $\mu_\bullet(X) = \Pi x : T'.U'$ et $T \rhd_\bullet T'$, $U' \rhd_\bullet U$.*

Démonstration. Par induction sur la dérivation de coercion, on fait le cas pour Σ.

- $\rhd$-Conv : Trivial, puisqu'on aura $\mu_\bullet(X) = \mu_\bullet(Y)$.
- $\rhd$-$\downarrow$: Trivial puisque pour tout x, $\mu_\bullet(x) = \mu_\bullet(x^\downarrow)$.
- $\rhd$-Prod : Impossible, $\mu_\bullet()$ ne traversant pas les produits.
- $\rhd$-Sum : Direct, on a une dérivation de $\Sigma x : T'.U' \rhd_\bullet \Sigma x : T.U$.
- $\rhd$-Proof : Ici, $Y \equiv \{\ x : V \mid P\ \}$, on peut donc déduire que $\mu_\bullet(Y) = \mu_\bullet(V) = \Sigma x : T.U$. On applique l'hypothèse de récurrence avec $X \rhd_\bullet V$ et on obtient : $\mu_\bullet(X) = \Sigma x : T'.U' \wedge T' \rhd_\bullet T \wedge U' \rhd_\bullet U$.
- $\rhd$-Subset : Ici, $X \equiv \{\ x : V \mid P\ \}$. Par induction, $\mu_\bullet(V) = \mu_\bullet(X) = \Sigma x : T'.U' \wedge T' \rhd_\bullet T \wedge U' \rhd_\bullet U$.

□

Il nous faut montrer des lemmes faisant intervenir la substitution pour pouvoir prouver la complétude.

Lemme 2.2.21 (Substitutivité de $\mu_\bullet()$). *Si $\mu_\bullet(T) = \Pi y : U.V$ alors $\mu_\bullet(T[u/x]) = \Pi y : U[u/x].V[u/x]$. Si $\mu_\bullet(T) = \Sigma y : U.V$ alors $\mu_\bullet(T[u/x]) = \Sigma y : U[u/x].V[u/x]$.*

Démonstration. On montre la propriété pour les produits, la preuve est similaire pour les sommes. Par induction sur le nombre de constructeurs $\Pi, \Sigma, \{|\}$ dans la forme normale complète de T.

Il suffit de suivre la définition de $\mu_\bullet()$. Si $T^\downarrow$ est de la forme $\{\ y : V \mid P\ \}$ alors on a $\mu_\bullet(V) = \Pi y : U.V$ et par induction $\mu_\bullet(V[u/x]) = \Pi y : U[u/x].V[u/x]$. Il s'ensuit directement que $\mu_\bullet(T[u/x]) = \Pi y : U[u/x].V[u/x]$.

Si $T^\downarrow$ est différent d'un type sous-ensemble alors $\mu_\bullet(T) = T^\downarrow$. On a alors $T^\downarrow = \Pi y : U.V$ et donc $T[u/x]^\downarrow = \Pi y : U[u/x].V[u/x] = \mu_\bullet(T[u/x])$. □

Lemme 2.2.22 (Substitutivité du typage). *Si $\Gamma \vdash_\bullet u : U$ alors*

$$\begin{cases} \Gamma, x : U, \Delta \vdash_\bullet t : T & \Rightarrow \quad \Gamma, \Delta[u/x] \vdash_\bullet t[u/x] : T[u/x] \\ \vdash_\bullet \simeq \mathit{wf} G, x : U, \Delta & \Rightarrow \quad \vdash_\bullet \simeq \mathit{wf} G, \Delta[u/x] \\ \Gamma, x : U, \Delta \vdash_\bullet U \rhd_\bullet T : & \Rightarrow \quad \Gamma, \Delta[u/x] \vdash_\bullet U[u/x] \rhd_\bullet T[u/x] : \end{cases}$$

Démonstration. Par induction mutuelle sur les dérivation de typage, bonne formation et coercion.

- Wf-Empty : Trivial.
- Wf-Var : Par induction sur Δ.
  - $\Delta = []$: On a alors $\Gamma \vdash_\bullet U : s$ donc $\vdash_\bullet \simeq \mathbf{wf} G$ et trivialement, $\vdash_\bullet \simeq \mathbf{wf} G, \Delta[u/x]$.
  - $\Delta = \Delta', y : T$: On a alors $\Gamma, x : U, \Delta' \vdash_\bullet T : s$ et par induction $\Gamma, \Delta'[u/x] \vdash_\bullet T[u/x] : s[u/x] = s$. Donc on peut appliquer Wf-Var pour obtenir $\vdash_\bullet \simeq \mathbf{wf} G, \Delta'[u/x], y : T[u/x]$ soit $\vdash_\bullet \simeq \mathbf{wf} G, \Delta[u/x]$
- PropSet : La substitution n'a aucun effet et $\Gamma, \Delta[u/x]$ est bien formé par induction.
- Var : Par induction, $\vdash_\bullet \simeq \mathbf{wf} G, \Delta[u/x]$. Si $t \equiv x$ alors on a $T = U$ et $T[u/x] = U$ puisque x n'apparaît pas dans U. On a donc $\Gamma, \Delta[u/x] \vdash_\bullet t[u/x] = u : T[u/x] = U$, qui peut s'obtenir par affaiblissement de $\Gamma \vdash_\bullet u : U$. Si $y : T \in \Gamma$ alors on applique simplement Var. Si $y : T \in \Delta$ alors $y : T[u/x] \in \Delta[u/x]$ et on obtient $\Gamma, \Delta[u/x] \vdash_\bullet y[u/x] : T[u/x]$ par Var.
- Prod : Par induction $\Gamma, \Delta[u/x] \vdash_\bullet T[u/x] : s_1[u/x]$ et $\Gamma, \Delta[u/x], y : T[u/x] \vdash_\bullet M[u/x] : s_2[u/x]$. On peut appliquer Prod pour obtenir $\Gamma, \Delta[u/x] \vdash_\bullet \Pi y : T[u/x].M[u/x] : s_2$ soit $\Gamma, \Delta[u/x] \vdash_\bullet (\Pi y : T.M)[u/x] : s_2$. De façon similaire pour les autres constructeurs de types.

– App : On étudie le cas de l'application qui requiert un lemme supplémentaire. Par induction, on a $\Gamma, \Delta[u/x] \vdash_\bullet f[u/x] : T[u/x]$ et $\Gamma, \Delta[u/x] \vdash_\bullet a[u/x] : A[u/x]$. Si $\mu_\bullet(T) = \Pi y : V.W$ alors $\mu_\bullet(T[u/x]) = \Pi y : V[u/x].W[u/x]$ (lemme 2.2.21). Par induction, on a aussi $\Gamma, \Delta[u/x] \vdash_\bullet A[u/x], V[u/x] : s$. Enfin, par substitutivité de la coercion on a $A[u/x] \rhd_\bullet V[u/x]$. On peut donc appliquer App pour obtenir $\Gamma, \Delta[u/x] \vdash_\bullet (f[u/x]\ a[u/x]) : W[u/x][a[u/x]/y]$. Or $W[u/x][a[u/x]/y] = W[a/y][u/x]$ ($y \notin \mathcal{FV}(u)$). On a donc bien $\Gamma, \Delta[u/x] \vdash_\bullet (f\ a)[u/x] : (W[a/y])[u/x]$.

– Pi-1 : Par induction on a $\Gamma, \Delta[u/x] \vdash_\bullet t[u/x] : S[u/x]$, et par substitutivité de $\mu_\bullet()$ on a aussi $\mu_\bullet(S[u/x]) = \Sigma y : T[u/x].U[u/x]$. Il suffit alors d'appliquer Pi-1

– Pi-2 : De même on se retrouve avec $\Gamma, \Delta[u/x] \vdash_\bullet t[u/x] : S[u/x]$, et $\mu_\bullet(S[u/x]) = \Sigma y : T[u/x].U[u/x]$. Il suffit alors d'appliquer Pi-2 pour obtenir :

$$\Gamma, \Delta[u/x] \vdash_\bullet \pi_2\ t[u/x] : U[u/x][\pi_1\ t[u/x]/y] = U[\pi_1\ t/y][u/x] \text{ car } y \notin \mathcal{FV}(u)$$

– Pair : On a :

$$\frac{\Gamma, x : V, \Delta \vdash_\bullet \Sigma y : T.V : s \qquad \Gamma, x : V, \Delta \vdash_\bullet t : T' \quad \Gamma, x : V, \Delta \vdash_\bullet T' \rhd_\bullet T : s \qquad \Gamma, x : V, \Delta \vdash_\bullet v : V' \quad \Gamma, x : V, \Delta \vdash_\bullet V' \rhd_\bullet V[t/y] : s}{\Gamma, x : V, \Delta \vdash_\bullet (t, v)_{\Sigma y:T.V} : \Sigma y : T.V}$$

Par induction et application de Pair :

$$\frac{\Gamma, \Delta[u/x] \vdash_\bullet \Sigma y : T[u/x].V[u/x] : s \qquad \Gamma, \Delta[u/x] \vdash_\bullet t[u/x] : T'[u/x] \quad \Gamma, \Delta[u/x] \vdash_\bullet T'[u/x] \rhd_\bullet T[u/x] : s \qquad \Gamma, \Delta[u/x] \vdash_\bullet v[u/x] : V'[u/x] \quad \Gamma, \Delta[u/x] \vdash_\bullet V'[u/x] \rhd_\bullet V[u/x][t[u/x]/y]}{\Gamma, \Delta[u/x] \vdash_\bullet (t[u/x], v[u/x])_{\Sigma y:T[u/x].V[u/x]} : \Sigma y : T[u/x].V[u/x]}$$

On a bien $V[u/x][t[u/x]/y] = V[t/y][u/x]$ car $y \notin \mathcal{FV}(u)$.

– $\rhd$-Conv : Direct par préservation de l'équivalence $\equiv_{\beta\pi}$ par substitution et application de l'hypothèse d'induction pour le typage.

– $\rhd$-$\downarrow$: Par induction, $(U^\downarrow)[u/x] \rhd_\bullet (T^\downarrow)[u/x]$. Par le lemme 2.2.17, $((U^\downarrow)[u/x])^\downarrow \rhd_\bullet ((T^\downarrow)[u/x])^\downarrow$. Donc $U[u/x]^\downarrow \rhd_\bullet T[u/x]^\downarrow$ et par $\rhd$-$\downarrow$, $U[u/x] \rhd_\bullet T[u/x]$.

– $\rhd$-Prod : Par induction $U[u/x] \rhd_\bullet T[u/x]$ et $V[u/x] \rhd_\bullet W[u/x]$, donc $\Pi y : T[u/x].V[u/x] \rhd_\bullet \Pi y : U[u/x].W[u/x]$. La propriété est donc bien vérifiée.

– $\rhd$-Sum : Direct par induction.

– $\rhd$-Subset : Par induction, $U'[u/x] \rhd_\bullet V[u/x]$. On applique $\rhd$-Left pour obtenir $\{\ y : U'[u/x] \mid P\ \} \rhd_\bullet V[u/x]$.

– $\rhd$-Right : Direct par induction.

□

Corollaire 2.2.23 (Substitutivité du typage avec coercion). *Si* $\Gamma, x : V \vdash_\bullet t : T \rhd_\bullet U$ *et* $\Gamma \vdash_\bullet u : V$ *alors* $\Gamma \vdash t[u/x] : T[u/x] \rhd_\bullet U[u/x]$.

On a maintenant tout les ingrédients pour montrer la complétude de notre système de typage vis-à-vis du système déclaratif.

Théorème 2.2.24 (Complétude du typage). *Si* $\Gamma \vdash T : s$ *alors* $\Gamma \vdash_\bullet T : s$. *Si* $\Gamma \vdash t : T$ *alors* $\exists U, \Gamma \vdash_\bullet t : U$, $\Gamma \vdash_\bullet T, U : s$ *et* $U \rhd_\bullet T$. *Si* $\vdash \Gamma$ *dans le système déclaratif alors* $\vdash \Gamma$ *dans le système algorithmique.*

Démonstration. Par induction mutuelle sur les dérivations de typage et bonne formation.

– Wf-Empty : Trivial.

– Wf-Var :

$$\frac{\Gamma \vdash A : s}{\vdash \Gamma, x : A \ \mathbf{wf}}\, s \in \mathcal{S} \wedge x \notin \Gamma$$

Par induction $\exists s', \Gamma \vdash_\bullet A : s' \rhd s$. On a forcément $s' = s$ puisque les sortes ne sont en relation qu'avec elles-mêmes. On applique Wf-Var pour obtenir $\vdash \Gamma, x : A$.

– PropSet : Trivial.

– Var :

$$\frac{\vdash \Gamma \ \mathbf{wf} \qquad x : A \in \Gamma}{\Gamma \vdash x : A}$$

Par induction $\vdash \Gamma$ et $x : A \in \Gamma$, direct par Var. On vérifie que si A est une sorte on dérive bien la même sorte dans le système algorithmique.

– Prod :

$$\frac{\Gamma \vdash T : s_1 \qquad \Gamma, x : T \vdash U : s_2}{\Gamma \vdash \Pi x : T.U : s_3}\, (s_1, s_2, s_3) \in \mathcal{R}$$

Direct par induction et le fait que $\mathcal{R}$ est fonctionnelle.

– Abs :

$$\frac{\Gamma \vdash \Pi x : T.U : s \qquad \Gamma, x : T \vdash M : U}{\Gamma \vdash \lambda x : T.M : \Pi x : T.U}$$

Par induction $\exists U', \Gamma, x : T \vdash_\bullet M : U' \rhd_\bullet U$ et $\Gamma, x : T \vdash_\bullet U', U : s$. On peut donc dériver $\Gamma \vdash_\bullet \Pi x : T.U' : s$. On a bien $\Gamma \vdash_\bullet \lambda x : T.M : \Pi x : T.U' \rhd_\bullet \Pi x : T.U$ et les deux types ont la même sorte.

– App : On a

$$\frac{\Gamma \vdash f : \Pi x : V.W \qquad \Gamma \vdash u : V}{\Gamma \vdash (fu) : W[u/x]}$$

Par induction, $\exists T, \Gamma \vdash_\bullet f : T \rhd_\bullet \Pi x : V.W$ et $\exists U, \Gamma \vdash_\bullet u : U \rhd_\bullet V$.
Si $T \rhd_\bullet \Pi x : V.W$ alors $\mu_\bullet(T) = \Pi x : V'.W'$ avec $V \rhd_\bullet V'$ et $W' \rhd_\bullet W$ (lemme 2.2.20).
Par transitivité de la coercion : $U \rhd_\bullet V'$, on peut donc dériver

$$\frac{\Gamma \vdash_\bullet f : T \quad \mu_\bullet(T) = \Pi x : V'.W' \qquad \Gamma \vdash_\bullet u : U \quad \Gamma \vdash_\bullet U, V' : s \qquad U \rhd_\bullet V'}{\Gamma \vdash_\bullet (fu) : W'[u/x]}$$

Par substitutivité de la coercion (lemme 2.2.22), $W'[u/x] \rhd_\bullet W[u/x]$, la propriété est donc bien vérifiée. Les conditions de sortes sont vérifiées du fait que $\mu_\bullet()$ conserve les sortes, donc $\Gamma \vdash_\bullet T, \Pi x : V'.W', \Pi x : V.W : s$ puis par inversion, $\Gamma, x : V \vdash_\bullet W', W : s$ et enfin par subtitutivité, $\Gamma \vdash_\bullet W'[u/x], W[u/x] : s$.

– Sum :

$$\frac{\Gamma \vdash T : s \qquad \Gamma, x : T \vdash U : s}{\Gamma \vdash \Sigma x : T.U : s}\, s \in \{\texttt{Prop}, \texttt{Set}\}$$

Par induction $\Gamma \vdash_\bullet T : s$ et $\Gamma, x : T \vdash_\bullet U : s$ où $s \in \{\texttt{Prop}, \texttt{Set}\}$. C'est direct par Sum.

– Pair :

$$\frac{\Gamma \vdash \Sigma x : T.U : s \qquad \Gamma \vdash t : T \qquad \Gamma \vdash u : U[t/x]}{\Gamma \vdash (t, u)_{\Sigma x:T.U} : \Sigma x : T.U}$$

Ici, l'annotation nous force à utiliser le jugement de coercion. Par induction, $\Sigma x : T.U : s$, $\exists T', \Gamma \vdash_\bullet t : T' \rhd_\bullet T$ et $\exists U', \Gamma \vdash_\bullet u : U' \rhd_\bullet U[t/x]$. On peut montrer $\Gamma \vdash_\bullet \Sigma x : T'.U : s$. En effet, par inversion de $\Gamma \vdash_\bullet \Sigma x : T.U : s$ on a $\Gamma, x : T \vdash_\bullet U : s$ et par le lemme 2.2.13 ($T' \rhd_\bullet T$), $\Gamma, x : T' \vdash_\bullet U : s$. Comme $T' \rhd_\bullet T$ on obtient $\Sigma x : T'.U \rhd_\bullet \Sigma x : T.U$. On peut donc dériver :

$$\frac{\Gamma \vdash_\bullet t : T' \qquad \Gamma \vdash_\bullet u : U' \qquad \Gamma \vdash_\bullet U[t/x], U' : s \quad U' \rhd_\bullet U[t/x] \qquad \Gamma \vdash_\bullet \Sigma x : T'.U : s}{\Gamma \vdash_\bullet (x := t, u : U) : \Sigma x : T'.U}$$

– Pi-1, Pi-2 : On a

$$\frac{\Gamma \vdash t : \Sigma x : T.U}{\Gamma \vdash \pi_1\ t : T}$$

Par induction, $\exists T', \Gamma \vdash_\bullet t : T' \rhd_\bullet \Sigma x : T.U$:. On en déduit que $\mu_\bullet(T') = \Sigma x : T'.U'$ avec $\Gamma \vdash_\bullet T' \rhd_\bullet T$: et $\Gamma, x : T' \vdash_\bullet U' \rhd_\bullet U$:. Clairement, $\Gamma \vdash_\bullet \pi_1\ t : T' \rhd_\bullet T$: et $\Gamma \vdash_\bullet \pi_2\ t : U'[\pi_1\ t/x] \rhd_\bullet U[\pi_1\ t/x]$: par substitutivité de la coercion.

– Conv, Coerce : Dans les deux cas on a inductivement $\exists T', \Gamma \vdash_\bullet t : T' \rhd_\bullet T$. Avec Conv on a $T \equiv_{\beta\pi} S$, donc $T' \rhd_\bullet S$ par le lemme 2.2.16. Pour Coerce on a $T \rhd S$. Par complétude de la coercion, $T \rhd_\bullet S$ et par transitivité de la coercion, $T' \rhd_\bullet S$. La propriété est donc bien vérifiée dans les deux cas.

□

On combine les théorèmes de correction et de complétude pour obtenir la propriété suivante entre les deux systèmes :

Corollaire 2.2.25 (Équivalence des systèmes déclaratif et algorithmique). $\Gamma \vdash t : T$ *si et seulement si il existe* U *tel que* $\Gamma \vdash_\bullet t : U$ *et* $U \rhd_\bullet T$.

On a maintenant un système raffiné dérivant les mêmes jugements (à coercion près) que le système déclaratif. On veut en extraire un algorithme de typage. Pour cela on doit pouvoir résoudre deux problèmes :

– **Vérification de type.** On donne Γ,t et T et l'on doit décider si $\Gamma \vdash_\bullet t : T$;
– **Inférence de type.** On donne Γ,t et l'on doit trouver T tel que $\Gamma \vdash_\bullet t : T$ si c'est dérivable, sinon on échoue.

En pratique, la vérification a besoin de l'inférence puisque lorsqu'on vérifie une application $fu : T$ on doit inférer le type de f. On montre donc les théorèmes suivants :

Théorème 2.2.26 (Décidabilité de l'inférence dans le système algorithmique). *Le problème d'inférence* $\Gamma \vdash_\bullet t : ?$ *est décidable.*

Démonstration. Il suffit d'observer que les règles de typage sont dirigées par la syntaxe du deuxième argument et permettent donc d'inférer un type pour tout terme. En lisant les prémisses de chaque règle de gauche à droite, on voit que l'inférence est décidable. □

Théorème 2.2.27 (Décidabilité de $\vdash_\bullet$). *La relation de typage* $\Gamma \vdash_\bullet t : T$ *est décidable.*

Démonstration. Direct. On utilise le théorème précédent pour le cas de l'application. □

On a désormais un algorithme de typage pour notre système avec coercions. Ce système est très libéral puisqu'il permet de considérer des objets comme vérifiant des propriétés arbitraires sans les montrer. Il nous faut maintenant remettre de la logique dans nos termes pour s'assurer qu'ils sont corrects.

2.3 Génération des obligations de preuve

On veut désormais traduire les dérivations du système algorithmique dans Cci dont le jugement de typage est $\vdash_{CCI}$. Les termes de Russell ne sont pas directement typables dans Cci puisque nous avons permis d'utiliser des objets comme s'ils avaient des types différents de leurs types originaux avec la règle de coercion. Il va donc falloir maintenant expliciter ces coercions pour obtenir des termes typables dans Cci. Cependant, on ne peut pas créer un terme complet à partir de notre dérivation, puisqu'on ne peut pas inférer des preuves arbitraires. On utilise donc des existentielles (intuitivement des trous dont on ne connait que le type des habitants) pour traduire le fait qu'il est de la responsabilité de l'utilisateur de prouver que son utilisation de la coercion n'était pas incorrecte.

2.3.1 Interprétation

On définit l'interprétation $[\![t]\!]_\Gamma$ par récurrence sur la forme des termes (figure 2.8). Cette interprétation renvoie un terme t' réecrit que l'on montrera bien typé dans l'environnement Cci $[\![\Gamma]\!]$.

Définition 2.3.1 (Interprétation des contextes). *On fait l'extension aux contextes de la façon suivante :*

- $[\![[]]\!] = []$
- $[\![\Gamma, x : T]\!] = [\![\Gamma]\!], x : [\![T]\!]_\Gamma$

Chaque jugement de coercion du système algorithmique permet de dériver une coercion explicite qui sera directement appliquée à un objet.

On formalise donc les coercions par des contextes d'évaluation classiques.

Définition 2.3.2 (Contextes d'évaluation). *Un contexte d'évaluation est un terme formé à partir de la grammaire originale des termes à laquelle on ajoute des terminaux $\bullet$ dans chacune de règles.*

Définition 2.3.3 (Substitution et composition de coercions). *La substitution (l'application) dans un contexte d'évaluation est notée $c[d]$, elle remplace toutes les occurrences de $\bullet$ dans c par d.*

Le composition de deux coercions notée $c \circ d$ est égale à $c[d]$, son élément neutre est $\bullet$.

La substitution d'un terme pour une variable dans un contexte d'évaluation est notée $c[t/x]$ comme pour les termes.

Coercions explicites

On définit le système $\rhd_\bullet$ (figure 2.11) qui dérive une coercion à partir de deux types S et T dans un environnement Γ. On a introduit du déterminisme par rapport au jugement de coercion algorithmique puisqu'on donne priorité à la règle $\rhd$-Subset par rapport à la règle $\rhd$-Proof (ces règles sont confluentes comme nous le monterons lemme 2.3.4). On explicite aussi la priorité donnée à la mise en forme normale de tête (figure 2.9) puis à la dérivation par rapport au test de conversion dans la prémisse de $\rhd$-Conv.

Notre opération de mise en forme normale de tête est définie de la façon suivante :

On note donc $T^\downarrow$ la forme normale de tête T et $T\!\downarrow$ la forme normale de T.

On utilise l'équivalence $\equiv_{\beta\pi\eta\rho\sigma}$ définie comme la clôture réflexive, symétrique et transitive de la relation définie figure **??**. Cette relation sera dénotée par $\equiv$ pour plus de clarté. Cette relation contient la β-réduction et les projections pour les sommes dépendantes, mais aussi des relations nécessaires pour supporter l'interprétation de termes de Russell dans le langage. On a donc la règle η pour l'abstraction et ρ pour le *surjective pairing* qui s'applique aux sommes dépendantes et aux objets de type sous-ensemble. Enfin on a une forme limitée d'indifférence aux preuves pour les objets de type sous-ensemble. On ajoute une règle de typage au système de Cci pour typer les existentielles :

$$\frac{\Gamma \vdash_{CCI} P : \texttt{Prop}}{\Gamma \vdash_{CCI} ?_P : P}$$

Le système figure 2.11 dérive les termes de coercion. Il a de bonnes propriétés pour la preuve et l'implémentation telles que l'unicité et l'admissibilité de la transitivité que nous montrerons plus tard.

2.3.2 Propriétés

On veut montrer que si l'on a un jugement valide dans notre système algorithmique, alors son image par l'interprétation est un jugement valide de Cci. On rappelle que Cci est équivalent au premier calcul présenté où la règle de coercion est remplacée par la règle de conversion.

Correction

Notre problème se ramène à montrer le théorème suivant :

$$\Gamma \vdash_\bullet t : T \Rightarrow [\![\Gamma]\!] \vdash_{CCI} [\![t]\!]_\Gamma : [\![T]\!]_\Gamma$$

Ce résultat ne se montre pas aisément. En effet le jugement de coercion rend la preuve très difficile à cause de son caractère non local. Pour mieux comprendre ce problème, considérons l'exemple suivant :

Exemple Dans le système algorithmique, on peut très bien dériver $\Pi n : \mathtt{nat}.\mathbf{list}\ n \rhd_\bullet \Pi n : \{ x : \mathtt{nat} \mid P \}.\mathbf{list}\ n$ puisque $\{ x : \mathtt{nat} \mid P \} \rhd_\bullet \mathtt{nat}$ et $\mathbf{list}\ n \equiv_{\beta\pi} \mathbf{list}\ n$. Si l'on interprète ces deux types, une coercion va être insérée dans le second type : $[\![\Pi n : \{ x : \mathtt{nat} \mid P \}.\mathbf{list}\ n]\!]_\Gamma = \Pi n : \{ x : \mathtt{nat} \mid P \}.\mathbf{list}\ (\pi_1\ n)$. La coercion générée doit donc avoir pour type : $\Pi n : \mathtt{nat}.\mathbf{list}\ n \to \Pi n : \{ x : \mathtt{nat} \mid P \}.\mathbf{list}\ (\pi_1\ n)$, mais elle est dérivée en se basant seulement sur les types algorithmiques. On peut vérifier ici que l'intuition de la coercion par prédicats est bonne, puisqu'on peut dériver ce jugement :

$$\dfrac{\dfrac{\dfrac{\mathtt{nat} \equiv_{\beta\pi} \mathtt{nat}}{\Gamma' \vdash_{CCI} \bullet : \mathtt{nat} \rhd_\bullet \mathtt{nat} :}}{\Gamma' \vdash_{CCI} \pi_1\ \bullet : \{ x : \mathtt{nat} \mid P \} \rhd_\bullet \mathtt{nat} :} \quad \dfrac{\mathbf{list}\ n \equiv_{\beta\pi} \mathbf{list}\ n}{\Gamma, n : \{ x : \mathtt{nat} \mid P \} \vdash_{CCI} \bullet : \mathbf{list}\ n \rhd_\bullet \mathbf{list}\ n :}}{\Gamma' \vdash_{CCI} \lambda x : [\![\{ x : \mathtt{nat} \mid P \}]\!]_\Gamma.\bullet\ [\bullet\ (\pi_1\ x)] = \bullet\ (\pi_1 x) : \Pi n : \mathtt{nat}.\mathbf{list}\ n \rhd_\bullet \Pi n : \{ x : \mathtt{nat} \mid P \}.\mathbf{list}\ n :}$$

Supposons $\Gamma \vdash_{CCI} t : \Pi n : \mathtt{nat}.\mathbf{list}\ n$ alors on a la dérivation de typage suivante :

$$\dfrac{\dfrac{\Gamma, x : \{ x : \mathtt{nat} \mid [\![P]\!]_\Gamma \} \vdash_\bullet t : \Pi n : \mathtt{nat}.\mathbf{list}\ n \qquad \Gamma, x : \{ x : \mathtt{nat} \mid [\![P]\!]_\Gamma \} \vdash_\bullet \pi_1\ x : \mathtt{nat}}{\Gamma, x : \{ x : \mathtt{nat} \mid [\![P]\!]_\Gamma \} \vdash_\bullet t\ (\pi_1\ x) : \mathbf{list}\ (\pi_1\ x)}}{\Gamma \vdash_\bullet \lambda x : [\![\{ x : \mathtt{nat} \mid P \}]\!]_\Gamma.t\ (\pi_1\ x) : \Pi n : [\![\{ x : \mathtt{nat} \mid P \}]\!]_\Gamma.\mathbf{list}\ (\pi_1\ x)}$$

On crée donc bien un terme de type $[\![\Pi n : \{ x : \mathtt{nat} \mid P \}.\mathbf{list}\ n]\!]_\Gamma$ en appliquant la coercion a un terme de type $[\![\Pi n : \mathtt{nat}.\mathbf{list}\ n]\!]_\Gamma$, c'est l'effet recherché.

Lemme 2.3.4 (Unicité de la coercion). *Si* $\Gamma \vdash_\bullet T, U : s$ *alors si* $\Gamma \vdash_{CCI} c : T \rhd_\bullet U :$ *et* $\Gamma \vdash_{CCI} c' : T \rhd_\bullet U :$ *alors* $c = c'$.

Démonstration. Par simple inspection des règles, on remarque qu'une seule règle s'applique suivant la forme de T, sauf si T et U sont des types sous-ensemble. On montre donc la confluence des deux règles $\rhd$-Subset et $\rhd$-Proof :

$$\dfrac{\dfrac{\Gamma \vdash_{CCI} c : T' \rhd_\bullet U' :}{\Gamma \vdash_{CCI} c[\sigma_1\ \bullet] : \{ x : T' \mid P \} \rhd_\bullet U' :}}{\Gamma \vdash_{CCI} c' : T = \{ x : T' \mid P \} \rhd_\bullet U = \{ x : U' \mid Q \} :}$$

avec

$$c' = \mathrm{elt}\ [\![U']\!]_\Gamma\ [\![\lambda x : U'.Q]\!]_\Gamma\ c[\sigma_1\ \bullet]\ ?_{[\![Q]\!]_{\Gamma, x:U'}[c[\sigma_1\ \bullet]/x]}$$

D'autre part :

$$\dfrac{\dfrac{\Gamma \vdash_{CCI} c : T' \rhd_\bullet U' :}{\Gamma \vdash_{CCI} c'' : T' \rhd_\bullet \{ x : U' \mid Q \} :}}{\Gamma \vdash_{CCI} c''[\sigma_1 \bullet] : T = \{ x : T' \mid P \} \rhd_\bullet U :}$$

avec

$$c'' = \text{elt } [\![U']\!]_\Gamma \ [\![\lambda x : U'.Q]\!]_\Gamma \ c \ ?_{[\![Q]\!]_{\Gamma,x:U'}[c/x]}$$

Clairement, $c''[\sigma_1 \bullet] = \text{elt } [\![U']\!]_\Gamma \ [\![\lambda x : U'.Q]\!]_\Gamma \ c[\sigma_1 \bullet] \ ?_{[\![Q]\!]_{\Gamma,x:U'}[c[\sigma_1 \bullet]/x]} = c'$.

La confluence locale suffit puisqu'il n'est pas possible d'appliquer d'autres règles que ces deux là si ces deux là sont applicables. □

On peut donc raisonner comme ceci : si l'on parvient à construire une dérivation de $\Gamma \vdash_{CCI} c : T \rhd_\bullet U :$ toute autre dérivation de $T \rhd_\bullet U$ donne le même terme de coercion.

Lemme 2.3.5 (Réflexivité de la coercion). *Si $\Gamma \vdash_\bullet A : s$ alors il existe c, $\Gamma \vdash_{CCI} c : A \rhd_\bullet A :$ et $c \equiv \bullet$.*

Démonstration. Par induction sur le nombre de constructeurs $\Pi, \Sigma, \{|\}$ dans la forme normale de A.

S'il $A{\downarrow}$ n'a pas de constructeur $\Pi, \Sigma, \{|\}$ en tête, alors $A^{\downarrow}$ n'en a pas en tête et l'on peut directement appliquer $\rhd$-Conv.

Sinon, on va appliquer la ou les règles correspondant au constructeur en tête. Le cas le plus intéressant est si $A^{\downarrow}$ est de la forme $\{ x : U \mid P \}$. Alors on a la dérivation :

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash_{CCI} c' \equiv \bullet : U \rhd_\bullet U :}{\Gamma \vdash_{CCI} d = c'[\sigma_1 \bullet] : \{ x : U \mid P \} \rhd_\bullet U :}}{\Gamma \vdash_{CCI} c = \text{elt } [\![U]\!]_\Gamma \ [\![\lambda x : U.P]\!]_\Gamma \ d \ ?_{[\![P]\!]_{\Gamma,x:U}[d/x]} : A^{\downarrow} = \{ x : U \mid P \} \rhd_\bullet \{ x : U \mid P \} :}}{\Gamma \vdash_{CCI} c : A \rhd_\bullet A :}$$

On obtient la dérivation de c' par induction, puisque $U{\downarrow}$ contient strictement moins de constructeurs que $A{\downarrow}$.

On a :

$$\begin{aligned} c &\hat{=} \text{elt } [\![U]\!]_\Gamma \ [\![\lambda x : U.P]\!]_\Gamma \ d \ ?_{[\![P]\!]_{\Gamma,x:U}[d/x]} \\ &\hat{=} \text{elt } [\![U]\!]_\Gamma \ [\![\lambda x : U.P]\!]_\Gamma \ (\sigma_1 \bullet) \ ?_{[\![P]\!]_{\Gamma,x:U}[\sigma_1 \bullet/x]} \\ &=_\sigma \text{elt } [\![U]\!]_\Gamma \ [\![\lambda x : U.P]\!]_\Gamma \ (\sigma_1 \bullet) \ (\sigma_2 \bullet) \\ &=_\rho \bullet \end{aligned}$$

On utilise ici l'indifférence aux preuves (σ). Comme $(\sigma_2 \bullet)$ est de type $[\![P]\!]_{\Gamma,x:U}[\sigma_1 \bullet/x]$ dans un contexte où $\bullet$ est de type $[\![\{ x : U \mid P \}]\!]_\Gamma$, on peut remplacer directement l'existentielle par ce terme. En pratique, ces preuves pourront être directement déchargées par l'assistant de preuve. □

Lemme 2.3.6 (Coercion et sortage). *Si $\Gamma \vdash_\bullet T : s_1$, $\Gamma \vdash_\bullet U : s_2$ et $\Gamma \vdash_\bullet T \rhd_\bullet U :$ alors $s_1 = s_2$ et la taille des dérivations de sortage $\Gamma \vdash_\bullet T, U : s$ est plus petite que la taille de la dérivation de coercion.*

Démonstration. Par induction sur la dérivation de coercion.

– $\rhd$-Conv, $\rhd$-$\downarrow$: Trivial.

– $\rhd$-Prod : Par induction et application de la règle Prod ; on repose sur le fait que la relation $\mathcal{R}$ est fonctionnelle.

– $\rhd$-Sum : Direct par induction et application de la règle Sum.

– $\rhd$-Subset :

$$\dfrac{\Gamma \vdash_\bullet T' \rhd_\bullet U :}{\Gamma \vdash_\bullet \{ x : T' \mid P \} \rhd_\bullet U :}$$

Si $\Gamma \vdash_\bullet \{ x : T' \mid P \} : s_1$, alors $s_1 = \texttt{Set}$ et $\Gamma \vdash_\bullet T' : \texttt{Set}$. Donc par induction, $\Gamma \vdash_\bullet U : \texttt{Set} = s_2$.

– $\rhd$-Proof :

$$\frac{\Gamma \vdash_\bullet T \rhd_\bullet U' :}{\Gamma \vdash_\bullet T \rhd_\bullet \{ x : U' \mid P \} :}$$

Par inversion du jugement de typage $\Gamma \vdash_\bullet \{ x : U' \mid P \} : s_2$, on a $s_2 = \texttt{Set}$ et $\Gamma \vdash_\bullet U' : \texttt{Set}$, donc par application de l'hypothèse d'induction, $\Gamma \vdash_\bullet T : \texttt{Set}$.

□

Ce lemme montre que nous avons suffisament de conditions de sortes dans nos règles pour assurer qu'en n'importe quel point d'une dérivation de coercion commençant par deux types bien sortés, nous travaillons sur des types de même sorte, et que nous pouvons appliquer nos hypothèses d'induction mutuelle aux jugement de sortage correspondants.

Lemme 2.3.7 (Coercion et formes normales de tête). *Si $\Gamma \vdash_{CCI} c : T \rhd_\bullet U$ alors $\Gamma \vdash_{CCI} c' : T\downarrow \rhd_\bullet U\downarrow$ avec $c = c'$ est dérivable par une dérivation plus petite ou égale.*

Démonstration. Par idempotence de la mise en forme normale de tête, on a la même dérivation dans le cas où la dernière règle appliquée était $\rhd$-$\downarrow$, sinon c'est trivial. □

On va maintenant généraliser la réflexivité aux termes convertibles.

Lemme 2.3.8 (Coercion de termes convertibles). *Si $\Gamma \vdash_\bullet T, U : s$, $T \equiv_{\beta\pi} U$ alors il existe c, $\Gamma \vdash_{CCI} c : T \rhd_\bullet U :$ avec $c \equiv \bullet$.*

Démonstration. Par induction sur le nombre de constructeurs $\Pi, \Sigma, \{|\}$ dans les formes normales de T et U.

– Si $T\downarrow$ n'a pas pour symbole de tête, Π, Σ ou $\{|\}$, alors $\rhd$-Conv est la seule règle applicable et on a $c = \bullet$.

– Si $T^\downarrow = \Pi y : A.B$, alors $U^\downarrow = \Pi y : A'.B'$ avec $A \equiv_{\beta\pi} A'$, $B \equiv_{\beta\pi} B'$. Par induction, $\Gamma \vdash_{CCI} c_1 \equiv \bullet : A' \rhd_\bullet A :$ et $\Gamma, x : A' \vdash_{CCI} c_2 \equiv \bullet : B \rhd_\bullet B' : (A\downarrow, A'\downarrow, B\downarrow, B'\downarrow$ sont des sous-termes stricts de $T\downarrow$ et $U\downarrow)$. On peut donc dériver : $\Gamma \vdash_{CCI} \lambda x : [\![A']\!]_\Gamma . c_2[\bullet\ c_1[x]] : \Pi y : A.B \rhd_\bullet \Pi y : A'.B' :$. Puis par application de $\rhd$-$\downarrow$, la coercion de T à U.
On a donc :

$$\begin{aligned} c &\hat{=} \lambda x : [\![A']\!]_\Gamma . c_2[\bullet\ (c_1[x])] \\ &\equiv \lambda x : [\![A']\!]_\Gamma . \bullet\ x \\ &=_\eta \bullet \end{aligned}$$

– Si $T^\downarrow = \Sigma y : A.B$ alors $U^\downarrow = \Sigma y : A'.B'$ avec $A \equiv_{\beta\pi} A'$, $B \equiv_{\beta\pi} B'$. Par induction, $\Gamma \vdash_{CCI} c_1 \equiv \bullet : A \rhd_\bullet A' :$ et $\Gamma, y : A \vdash_{CCI} c_2 \equiv \bullet : B \rhd_\bullet B' : (A\downarrow, A'\downarrow, B\downarrow, B'\downarrow$ sont des sous-termes stricts de $T\downarrow$ et $U\downarrow)$. On peut donc dériver : $\Gamma \vdash_{CCI} (c_1[\pi_1\ \bullet], c_2[\pi_2\ \bullet][\pi_1\ \bullet/y])_{[\![\Sigma y:A'.B']\!]_\Gamma} : \Sigma y : A.B \rhd_\bullet \Sigma y : A'.B' :$. Puis par application de $\rhd$-$\downarrow$, la coercion de T à U.
On a donc :

$$\begin{aligned} c &\hat{=} (c_1[\pi_1\ \bullet], c_2[\pi_2\ \bullet][\pi_1\ \bullet/y])_{[\![\Sigma y:A'.B']\!]_\Gamma} \\ &\equiv (\pi_1\ \bullet, (\pi_2\ \bullet)[\pi_1\ \bullet/y])_{[\![\Sigma y:A'.B']\!]_\Gamma} \\ &= (\pi_1\ \bullet, \pi_2\ \bullet)_{[\![\Sigma y:A'.B']\!]_\Gamma} \\ &=_\rho \bullet \end{aligned}$$

La substitution est inutile puisque dans un contexte où $\bullet : \Sigma y : A.B$, $y \notin \pi_2\ \bullet$.

– Le cas des sous-ensembles est un peu différent. Si $T^{\downarrow} = \{ x : T' \mid P \}$ alors $U^{\downarrow} = \{ x : U' \mid P' \}$ avec $T' \equiv_{\beta\pi} U'$ et $P \equiv_{\beta\pi} P'$. La dérivation va avoir la forme suivante :

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash_{CCI} c' \equiv \bullet : T' \rhd_\bullet U' :}{\Gamma \vdash_{CCI} d = c'[\sigma_1 \bullet] : \{ x : T' \mid P \} \rhd_\bullet U' :}}{\Gamma \vdash_{CCI} c = \mathrm{elt}\ [\![U']\!]_\Gamma\ [\![\lambda x : U'.P']\!]_\Gamma\ d\ ?_{[\![P']\!]_{\Gamma,x:U'}[d/x]} : \{ x : T' \mid P \} \rhd_\bullet \{ x : U' \mid P' \} :}}{\Gamma \vdash_{CCI} c : T \rhd_\bullet U :}$$

On a donc $d = c'[\sigma_1 \bullet] \equiv \sigma_1 \bullet$.
On peut vérifier :

$$\begin{array}{rcl} c & \hat{=} & \mathrm{elt}\ [\![U']\!]_\Gamma\ [\![\lambda x : U'.P']\!]_\Gamma\ d\ ?_{[\![P']\!]_{\Gamma,x:U'}[d/x]} \\ & \equiv & \mathrm{elt}\ [\![U']\!]_\Gamma\ [\![\lambda x : U'.P']\!]_\Gamma\ (\sigma_1 \bullet)\ ?_{[\![P']\!]_{\Gamma,x:U'}[\sigma_1 \bullet/x]} \\ & =_\sigma & \mathrm{elt}\ [\![U']\!]_\Gamma\ [\![\lambda x : U'.P']\!]_\Gamma\ (\sigma_1 \bullet)\ (\sigma_2 \bullet) \\ & =_\rho & \bullet \end{array}$$

On fait ici un usage très libéral de l'indifférence aux preuves. En effet nous ne pouvons pas encore montrer que $\sigma_2 \bullet : [\![P']\!]_{\Gamma,x:U'}[\sigma_1 \bullet/x]$ puisqu'on sait seulement que dans le contexte où $\bullet : [\![\{ x : T' \mid P \}]\!]_\Gamma$, $\sigma_2 \bullet : [\![P]\!]_{\Gamma,x:T'}[\sigma_1 \bullet/x]$. Montrer que les interprétations de P et P' sont équivalentes requiert la stabilité de la convertibilité par interprétation, ce que nous montrerons plus tard.

□

Lemme 2.3.9 (Coercion de sortes). *Si $\Gamma \vdash_{CCI} e : s \rhd_\bullet T :$ ou $\Gamma \vdash_{CCI} e : T \rhd_\bullet s :$ alors $T \equiv_{\beta\pi} s$ et $e = \bullet$.*

Démonstration. Clairement on ne peut dériver $s \rhd_\bullet T$ que par $\rhd$-Conv (éventuellement précédé de $\rhd$-$\downarrow$). En effet seule la règle $\rhd$-Proof pourrait s'appliquer, mais cela impliquerait que $T \equiv_{\beta\pi} \{ x : U \mid P \}$ avec $s \rhd_\bullet U$ et ainsi de suite. La seule possibilité est de dériver $s \equiv_{\beta\pi} T$ ou $s \equiv_{\beta\pi} U$, auquel cas U est une sorte ce qui contredit le fait que $\{ x : U \mid P \} : s$ dans le cas précédent. On dérive donc $s \rhd_\bullet T$ si et seulement si $s \equiv_{\beta\pi} T$. □

Lemme 2.3.10 (Affaiblissement et interprétation). *Si $\Gamma \vdash_\bullet t : T$ alors pour tout $\Delta \supseteq \Gamma$, $[\![t]\!]_\Gamma = [\![t]\!]_\Delta$.*

Démonstration. Les seuls endroits où les environnement sont utilisés dans l'interprétation est lors des appels à la fonction de typage algorithmique, or on a bien la propriété que si $\Gamma \vdash_\bullet t : T$ alors $\Delta \vdash_\bullet t : T$ quand $\Gamma \subseteq \Delta$ par affaiblissement. Cette propriété est donc bien vérifiée. □

Lemme 2.3.11 (Stabilité de la coercion par substitution). *Si $\Gamma \vdash_\bullet u : U$, alors*

$$\begin{array}{lcl} \vdash_\square \Gamma, x : U, \Delta\ \boldsymbol{wf} & \Rightarrow & [\![\Gamma, \Delta[u/x]]\!] \equiv [\![\Gamma, x : U, \Delta]\!][[\![u]\!]_\Gamma/x] \\ \Gamma, x : U, \Delta \vdash_\bullet t : T & \Rightarrow & [\![t[u/x]]\!]_{\Gamma,\Delta[u/x]} \equiv [\![t]\!]_{\Gamma,x:U,\Delta}[[\![u]\!]_\Gamma/x] \\ \Gamma, x : U, \Delta \vdash_{CCI} c : T \rhd_\bullet T' : & \Rightarrow & \Gamma, \Delta[u/x] \vdash_{CCI} c' : T[u/x] \rhd_\bullet T'[u/x] : \wedge c' \equiv c[[\![u]\!]_\Gamma/x] \end{array}$$

Démonstration. Par induction mutuelle sur les dérivations de bonne formation, typage et coercion.

– Wf-Empty : Trivial.

– Wf-Var : Par induction sur la longueur de Δ.

- $\Delta = []$: Alors on a :

$$\dfrac{\Gamma \vdash_\square U : s}{\vdash_\square \simeq \mathbf{wf} G, x : U}\ s \in \{\mathtt{Set}, \mathtt{Prop}, \mathtt{Type}\} \wedge x \notin \Gamma$$

  Clairement $[\![\Gamma, \Delta[u/x]]\!] = [\![\Gamma]\!] = [\![\Gamma, x : U]\!][[\![u]\!]_\Gamma/x]$ puisque $x \notin \Gamma$.

- $\Delta = \Delta', x : A$: Alors on a :

$$\frac{\Gamma, x : U, \Delta' \vdash_\bullet A : s}{\vdash_\square \Gamma, x : U, \Delta', y : A \ \mathbf{wf}}$$

Par induction on a

$$[\![A[u/x]]\!]_{\Gamma,\Delta'[u/x]} \equiv [\![A]\!]_{\Gamma,x:U,\Delta'}[[\![u]\!]_\Gamma/x] \wedge [\![\Gamma, x : U, \Delta']\!][[\![u]\!]_\Gamma/x] \equiv [\![\Gamma, \Delta'[u/x]]\!]$$

donc

$$\begin{array}{rcl} [\![\Gamma, \Delta[u/x]]\!] & = & [\![\Gamma, \Delta'[u/x], A[u/x]]\!] \\ & \hat{=} & [\![\Gamma, \Delta'[u/x]]\!], [\![A[u/x]]\!]_{\Gamma,\Delta'[u/x]} \\ & = & [\![\Gamma, x : U, \Delta']\!][[\![u]\!]_\Gamma/x], [\![A]\!]_{\Gamma,x:U,\Delta'}[[\![u]\!]_\Gamma/x] \\ & = & [\![\Gamma, x : U, \Delta', A]\!][[\![u]\!]_\Gamma/x] \\ & = & [\![\Gamma, x : U, \Delta]\!][[\![u]\!]_\Gamma/x] \end{array}$$

– $\rhd$-$\downarrow$: On a :

$$\frac{\Gamma, x : U, \Delta \vdash_{CCI} c : T^\downarrow \rhd_\bullet T'^\downarrow :}{\Gamma, x : U, \Delta \vdash_{CCI} c : T \rhd_\bullet T' :}$$

Par induction on a $\Gamma, \Delta[u/x] \vdash_{CCI} c' : T^\downarrow[u/x] \rhd_\bullet T'^\downarrow[u/x] :$ avec $c' \equiv c[[\![u]\!]_\Gamma/x]$. Si $T[u/x]^\downarrow = T^\downarrow[u/x]$ et $T'[u/x]^\downarrow = T'^\downarrow[u/x]$ c'est direct par induction. Sinon, on a $T^\downarrow = x\ \overrightarrow{a}$ ou $T'^\downarrow = x\ \overrightarrow{a}$. Les deux cas sont similaires, on traite le cas ou $T^\downarrow = x\ \overrightarrow{a}$. Le jugement $x\ \overrightarrow{a} \rhd_\bullet T'^\downarrow$ ne peut être dérivé que par $\rhd$-Proof ou $\rhd$-Conv.

- $\rhd$-Proof : On a :

$$\frac{\Gamma, x : U, \Delta \vdash_{CCI} d : x\ \overrightarrow{a} \rhd_\bullet U' :}{\Gamma, x : U, \Delta \vdash_{CCI} c : x\ \overrightarrow{a} \rhd_\bullet T'^\downarrow = \{\ y : U' \mid P\ \} :}$$

avec

$$c = \mathrm{elt}\ [\![U']\!]_{\Gamma,x:U,\Delta}\ [\![\lambda x : U'.P]\!]_{\Gamma,x:U,\Delta}\ d\ ?_{[\![P]\!]_{\Gamma,x:U,\Delta,y:U'}[d/y]}$$

Par induction on a donc une dérivation de $\Gamma, \Delta[u/x] \vdash_{CCI} d' : u\ \overrightarrow{a[u/x]} \rhd_\bullet U'[u/x] :$ avec $d' \equiv d[[\![u]\!]_\Gamma/x]$. Par le lemme 2.3.7 on a aussi une dérivation de $\Gamma, \Delta[u/x] \vdash_{CCI} d'' : (u\ \overrightarrow{a[u/x]})^\downarrow \rhd_\bullet (U'[u/x])^\downarrow :$ avec $d'' \equiv d'[[\![u]\!]_\Gamma/x]$.
Par le lemme 2.3.6 on a $\Gamma, x : U, \Delta \vdash_\bullet \{\ y : U' \mid P\ \} : \mathtt{Set}$, et par inversion $\Gamma, x : U, \Delta \vdash_\bullet U' : \mathtt{Set}$ et $\Gamma, x : U, \Delta, y : U' \vdash_\bullet P : \mathtt{Prop}$. On peut donc appliquer l'hypothèse d'induction pour obtenir : $[\![U'[u/x]]\!]_{\Gamma,\Delta[u/x]} \equiv [\![U']\!]_{\Gamma,x:U,\Delta}[[\![u]\!]_\Gamma/x]$ et $[\![P[u/x]]\!]_{\Gamma,\Delta[u/x],y:U'[u/x]} \equiv [\![P]\!]_{\Gamma,x:U,\Delta,y:U'}[[\![u]\!]_\Gamma/x]$. Par substitutivité on a aussi $\Gamma, \Delta[u/x] \vdash_\bullet \{\ y : U'[u/x] \mid P[u/x]\ \} : \mathtt{Set}$. On peut donc dériver :

$$\frac{\dfrac{\dfrac{\Gamma, \Delta[u/x] \vdash_{CCI} d' : (u\ \overrightarrow{a[u/x]})^\downarrow \rhd_\bullet (U'[u/x])^\downarrow :}{\Gamma, \Delta[u/x] \vdash_{CCI} d' : (u\ \overrightarrow{a[u/x]})^\downarrow \rhd_\bullet U'[u/x] :}}{\Gamma, \Delta[u/x] \vdash_{CCI} c' : T[u/x]^\downarrow \rhd_\bullet (T'[u/x])^\downarrow = \{\ y : U'[u/x] \mid P[u/x]\ \} :}}{\Gamma, \Delta[u/x] \vdash_{CCI} c' : T[u/x] \rhd_\bullet T'[u/x] :}$$

avec

$$c' = \mathrm{elt}\ [\![U'[u/x]]\!]_{\Gamma,\Delta[u/x]}\ [\![\lambda y : U'[u/x].P[u/x]]\!]_{\Gamma,\Delta[u/x]}\ d'\ ?_{[\![P[u/x]]\!]_{\Gamma,\Delta[u/x],y:U'[u/x]}[d'/y]}$$

On a aussi :

$$\begin{array}{rcl} [\![P[u/x]]\!]_{\Gamma,\Delta[u/x],y:U'[u/x]}[d'/y] & \equiv & [\![P]\!]_{\Gamma,x:U,\Delta,y:U'}[[\![u]\!]_\Gamma/x][d'/y] \\ & \equiv & [\![P]\!]_{\Gamma,x:U,\Delta,y:U'}[[\![u]\!]_\Gamma/x][d[[\![u]\!]_\Gamma/x]/y] \\ & = & [\![P]\!]_{\Gamma,x:U,\Delta,y:U'}[d/y][[\![u]\!]_\Gamma/x] \end{array}$$

Soit

$$A \triangleq [\![P[u/x]]\!]_{\Gamma,\Delta[u/x],y:U'[u/x]}[d'/y] \wedge B \triangleq [\![P]\!]_{\Gamma,x:U,\Delta,y:U'}[d/y]$$

on a donc :

$$\begin{array}{rcl} c' & \triangleq & \mathrm{elt}\ [\![U'[u/x]]\!]_{\Gamma,\Delta[u/x]}\ [\![(\lambda y : U'[u/x].P[u/x])]\!]_{\Gamma,\Delta[u/x]}\ d'\ ?_A \\ & \equiv & \mathrm{elt}\ [\![U'[u/x]]\!]_{\Gamma,\Delta[u/x]}\ [\![\lambda y : U'[u/x].P[u/x]]\!]_{\Gamma,\Delta[u/x]}\ d[[\![u]\!]_\Gamma/x]\ ?_A \\ & \equiv & \mathrm{elt}\ [\![U']\!]_{\Gamma,x:U,\Delta}[[\![u]\!]_\Gamma/x]\ [\![\lambda y : U'.P]\!]_{\Gamma,x:U,\Delta}[[\![u]\!]_\Gamma/x]\ d[[\![u]\!]_\Gamma/x]\ (?_B)[[\![u]\!]_\Gamma/x] \\ & = & (\mathrm{elt}\ [\![U']\!]_{\Gamma,x:U,\Delta}\ [\![\lambda y : U'.P]\!]_{\Gamma,x:U,\Delta}\ d\ ?_B)[[\![u]\!]_\Gamma/x] \\ & = & c[[\![u]\!]_\Gamma/x] \end{array}$$

- $\rhd$-Conv : Alors on a $T^\downarrow = x\ \overrightarrow{a}$ et $T'^\downarrow = x\ \overrightarrow{b}$ où $\overrightarrow{a} \equiv \overrightarrow{b}$. Par substitutivité de l'équivalence, $T^\downarrow[u/x] \equiv T'^\downarrow[u/x]$, donc par le lemme 2.3.8, on a $\Gamma, \Delta[u/x] \vdash_{CCI} c' : u\ \overrightarrow{a[u/x]} \rhd_\bullet u\ \overrightarrow{b[u/x]}$: est dérivable et $c' \equiv \bullet = \bullet[[\![u]\!]_\Gamma/x]$.

Dans le cas où $T'^\downarrow = x\ \overrightarrow{a}$ et $T^\downarrow \neq x\ \overrightarrow{a}$, le jugement $\Gamma, x : U, \Delta \vdash_{CCI} c : T^\downarrow \rhd_\bullet x\ \overrightarrow{a}$: ne peut être dérivé que par $\rhd$-Subset ou $\rhd$-Conv.

- $\rhd$-Subset : On a :

$$\frac{\Gamma, x : U, \Delta \vdash_{CCI} c' : U' \rhd_\bullet x\ \overrightarrow{a} :}{\Gamma, x : U, \Delta \vdash_{CCI} c = c'[\sigma_1\ \bullet] : T^\downarrow = \{\ y : U' \mid P\ \} \rhd_\bullet x\ \overrightarrow{a}\ :}$$

Par induction, on a une dérivation de $\Gamma, \Delta[u/x] \vdash_{CCI} c'' : U'[u/x] \rhd_\bullet u\ \overrightarrow{a[u/x]}$: avec $c'' \equiv c'[[\![u]\!]_\Gamma/x]$. On peut donc dériver :

$$\frac{\dfrac{\Gamma, \Delta[u/x] \vdash_{CCI} c'' : U'[u/x] \rhd_\bullet u\ \overrightarrow{a[u/x]} :}{\Gamma, \Delta[u/x] \vdash_{CCI} c''[\sigma_1\ \bullet] : T[u/x]^\downarrow = \{\ y : U'[u/x] \mid P[u/x]\ \} \rhd_\bullet T'[u/x]^\downarrow :}}{\Gamma, \Delta[u/x] \vdash_{CCI} c''[\sigma_1\ \bullet] : T[u/x] \rhd_\bullet T'[u/x] :}$$

Clairement, $c''[\sigma_1\ \bullet] \equiv c'[[\![u]\!]_\Gamma/x][\sigma_1\ \bullet] = c'[\sigma_1\ \bullet][[\![u]\!]_\Gamma/x] = c[[\![u]\!]_\Gamma/x]$.

– $\rhd$-Conv : On a $T = T^\downarrow$, $T' = T'^\downarrow$, $T \equiv_{\beta\pi} T'$ et $c \equiv \bullet$. Par substitutivité de la $\beta\rho$-équivalence, on a aussi $T[u/x] \equiv_{\beta\pi} T'[u/x]$. Par le lemme 2.3.8, on sait qu'il existe une coercion c' telle que le jugement $\Gamma, \Delta[u/x] \vdash_{CCI} c' : T[u/x] \rhd_\bullet T'[u/x]$: est dérivable et $c' \equiv \bullet$. On a bien $c' \equiv c[[\![u]\!]_\Gamma/x]$.

– $\rhd$-Prod : On a :

$$\frac{\Gamma, x : U, \Delta \vdash_{CCI} c_1 : A' \rhd_\bullet A : \qquad \Gamma, x : U, \Delta, y : A' \vdash_{CCI} c_2 : B \rhd_\bullet B' :}{\Gamma, x : U, \Delta \vdash_{CCI} \lambda y : [\![A']\!]_{\Gamma,x:U,\Delta}.\ c_2[\bullet\ c_1[y]] : \Pi y : A.B \rhd_\bullet \Pi y : A'.B' :}$$

Par induction et application de $\rhd$-Prod :

$$\frac{\Gamma, \Delta[u/x] \vdash_{CCI} c'_1 : A'[u/x] \rhd_\bullet A[u/x] : \qquad \Gamma, \Delta[u/x], y : A'[u/x] \vdash_{CCI} c'_2 : B[u/x] \rhd_\bullet B'[u/x] :}{\Gamma, \Delta[u/x] \vdash_{CCI} c' : \Pi y : A[u/x].B[u/x] \rhd_\bullet \Pi y : A'[u/x].B'[u/x] :}$$

où

$$c' = \lambda y : [\![A'[u/x]]\!]_{\Gamma,\Delta[u/x]}.\ c'_2[\bullet\ c'_1[y]] \quad c'_1 \equiv c_1[[\![u]\!]_\Gamma/x] \quad c'_2 \equiv c_2[[\![u]\!]_\Gamma/x]$$

Par le lemme de coercion et sortage (2.3.6), avec $\Gamma, x : U, \Delta \vdash_\bullet \Pi y : A'.B', \Pi y : A.B : s$ on a $\Gamma, x : U, \Delta \vdash_\bullet A', A : s_1$. On applique l'hypothèse d'induction pour obtenir $[\![A'[u/x]]\!]_{\Gamma,\Delta[u/x]} \equiv [\![A']\!]_{\Gamma,x:U,\Delta}[[\![u]\!]_\Gamma/x]$. On a donc bien $c' \equiv c[[\![u]\!]_\Gamma/x]$.

– $\rhd$-Sum, $\rhd$-Proof, $\rhd$-Subset : Idem, direct par induction.

– PropSet : Trivial.

– Var : On a :

$$\frac{\vdash_\bullet \Gamma, x : U, \Delta \ \mathbf{wf} \qquad y : T \in (\Gamma, x : U, \Delta)}{\Gamma, x : U, \Delta \vdash_\bullet y : T}$$

- Si $x \neq y$, alors par définition de l'interprétation, on doit montrer $[\![y[u/x]]\!]_{\Gamma,\Delta[u/x]} = y = [\![y]\!]_{\Gamma,x:U,\Delta}[[\![u]\!]_\Gamma/x]$.
- Sinon, $t[u/x] = u$ et $[\![u]\!]_{\Gamma,\Delta[u/x]} = [\![x]\!]_{\Gamma,x:U,\Delta}[[\![u]\!]_\Gamma/x] = [\![u]\!]_{\Gamma,x:U,\Delta}$. On utilise ici le fait que $[\![t]\!]_\Gamma = [\![t]\!]_\Delta$ si $\Gamma \vdash_\bullet t : T$ pour tout Δ incluant Γ (lemme 2.3.10).

– App :

$$\frac{\Gamma, x : U, \Delta \vdash_\bullet f : F \quad \mu_\bullet(F) = \Pi y : A.B : s \qquad \Gamma, x : U, \Delta \vdash_\bullet e : E \qquad \Gamma, x : U, \Delta \vdash_\bullet E \rhd_\bullet A :}{\Gamma, x : U, \Delta \vdash_\bullet (f\ e) : B[e/y]}$$

Par induction :

$$[\![f[u/x]]\!]_{\Gamma,\Delta[u/x]} \equiv [\![f]\!]_{\Gamma,x:U,\Delta}[[\![u]\!]_\Gamma/x]$$
$$[\![e[u/x]]\!]_{\Gamma,\Delta[u/x]} \equiv [\![e]\!]_{\Gamma,x:U,\Delta}[[\![u]\!]_\Gamma/x]$$

Par définition de la substitution et de l'interprétation,

$$\begin{aligned} [\![(f\ e)]\!]_{\Gamma,x:U,\Delta} &= (((\pi_F\ [\![f]\!]_{\Gamma,x:U,\Delta})\ (c_e\ [\![e]\!]_{\Gamma,x:U,\Delta})) \\ \text{où} & \\ \pi_F &= \mathbf{coerce}_{\Gamma,x:U,\Delta}\ F\ (\Pi y : A.B) \\ c_e &= \mathbf{coerce}_{\Gamma,x:U,\Delta}\ E\ A. \end{aligned}$$

On a la substitutivité du typage 2.2.22 donc le jugement substitué est :

$$\frac{\begin{array}{ccc} \Gamma, \Delta[u/x] \vdash_\bullet f[u/x] : F[u/x] & & \Gamma \vdash_\bullet E[u/x], A[u/x] : s \\ \mu_\bullet(F[u/x]) = \Pi y : A[u/x].B[u/x] : s & \Gamma, \Delta[u/x] \vdash_\bullet e[u/x] : E[u/x] & E[u/x] \rhd_\bullet A[u/x] \end{array}}{\Gamma, \Delta[u/x] \vdash_\bullet (f\ e)[u/x] : B'[e[u/x]/y]}$$

Soit $e' = e[u/x]$ et $f' = f[u/x]$, on a donc d'autre part :

$$\begin{aligned} [\![(f\ e)[u/x]]\!]_{\Gamma,\Delta[u/x]} &= [\![f'\ e']\!]_{\Gamma,\Delta[u/x]} \\ &= \pi_{F[u/x]}[[\![f']\!]_{\Gamma,\Delta[u/x]}]\ c_{e'}[[\![e']\!]_{\Gamma,\Delta[u/x]}] \\ \text{où} & \\ \pi_{F[u/x]} &= \mathbf{coerce}_{\Gamma,\Delta[u/x]}\ F[u/x]\ (\Pi y : A[u/x].B[u/x]) \\ c_{e'} &= \mathbf{coerce}_{\Gamma,\Delta[u/x]}\ E[u/x]\ A[u/x] \end{aligned}$$

Par induction, il existe des coercions d, e telles que : $\Gamma, \Delta[u/x] \vdash_{CCI} d : F[u/x] \rhd_\bullet (\Pi y : A.B)[u/x]$: et $\Gamma, \Delta[u/x] \vdash_{CCI} e : E[u/x] \rhd_\bullet A[u/x]$: avec $d \equiv \pi_F[[\![u]\!]_\Gamma/x]$ et $e \equiv c_e[[\![u]\!]_\Gamma/x]$. Par unicité des coercions on en déduit que $d = \pi_{F[u/x]}$ et $e = c_{e'}$.

On peut vérifier :

$$
\begin{array}{rl}
 & [\![f\ e]\!]_{\Gamma,x:U,\Delta}[[\![u]\!]_\Gamma/x] \\
 & \quad \{ \textbf{Définition de l'interprétation} \} \\
\hat{=} & (\pi_F[[\![f]\!]_{\Gamma,x:U,\Delta}])\ (c_e[[\![e]\!]_{\Gamma,x:U,\Delta}])[[\![u]\!]_\Gamma/x] \\
 & \quad \{ \textbf{Définition de la substitution} \} \\
= & (\pi_F[[\![u]\!]_\Gamma/x][[\![f]\!]_{\Gamma,x:U,\Delta}[[\![u]\!]_\Gamma/x]])\ (c_e[[\![u]\!]_\Gamma/x][[\![e]\!]_{\Gamma,x:U,\Delta}[[\![u]\!]_\Gamma/x]]) \\
 & \quad \{ \textbf{Application de l'hypothèse d'induction pour les termes} \} \\
\equiv & (\pi_F[[\![u]\!]_\Gamma/x][[\![f']\!]_{\Gamma,\Delta[u/x]}])\ (c_e[[\![u]\!]_\Gamma/x][[\![e']\!]_{\Gamma,\Delta[u/x]}]) \\
 & \quad \{ \textbf{Application de l'hypothèse d'induction pour les coercions} \} \\
\equiv & (d[[\![f']\!]_{\Gamma,\Delta[u/x]})\ e[[\![e']\!]_{\Gamma,\Delta[u/x]}] \\
 & \quad \{ \textbf{Unicité des coercions} \} \\
= & \pi_{F[u/x]}[[\![f']\!]_{\Gamma,\Delta[u/x]}]\ c_{e'}[[\![e']\!]_{\Gamma,\Delta[u/x]}] \\
 & \quad \{ \textbf{Définition de l'interprétation} \} \\
\hat{=} & [\![(f\ e)[u/x]]\!]_{\Gamma,\Delta[u/x]}
\end{array}
$$

– Prod, Sum, Subset : Par induction.

– Abs : On a :

$$\frac{\Gamma, x : U, \Delta \vdash_\bullet \Pi y : T.U : s \qquad \Gamma, x : U, \Delta, y : T \vdash_\bullet M : U}{\Gamma, x : U, \Delta \vdash_\bullet \lambda y : T.M : \Pi y : T.U}$$

On a bien :

$$
\begin{array}{rl}
 & [\![\lambda y : T.M]\!]_{\Gamma,x:U,\Delta}[[\![u]\!]_\Gamma/x] \\
 & \quad \{ \textbf{Définition de l'interprétation} \} \\
\hat{=} & \lambda y : [\![T]\!]_{\Gamma,x:U,\Delta}[[\![u]\!]_\Gamma/x].[\![M]\!]_{\Gamma,x:U,\Delta,y:T}[[\![u]\!]_\Gamma/x] \\
 & \quad \{ \textbf{Application de l'hypothèse de récurrence} \} \\
\equiv & \lambda y : [\![T[u/x]]\!]_{\Gamma,\Delta[u/x]}.[\![M[u/x]]\!]_{\Gamma,\Delta[u/x],y:T[u/x]} \\
 & \quad \{ \textbf{Définition de l'interprétation} \} \\
\hat{=} & [\![\lambda y : T[u/x].M[u/x]]\!]_{\Gamma,\Delta[u/x]}
\end{array}
$$

– Pair : On a

$$\frac{\begin{array}{c}\Gamma, x : U, \Delta \vdash_\bullet t : T' \quad \Gamma, x : U, \Delta \vdash_\bullet T' \rhd_\bullet T : s \\ \Gamma, x : U, \Delta \vdash_\bullet \Sigma y : T.V : s \qquad \Gamma, x : U, \Delta \vdash_\bullet v : V' \quad \Gamma, x : U, \Delta \vdash_\bullet V' \rhd_\bullet V[t/y] : s\end{array}}{\Gamma, x : U, \Delta \vdash_\bullet (t, v)_{\Sigma y:T.V} : \Sigma y : T.V}$$

et le jugement substitué :

$$\frac{\begin{array}{c}\Gamma, \Delta[u/x] \vdash_\bullet t[u/x] : T'[u/x] \quad \Gamma, \Delta[u/x] \vdash_\bullet T'[u/x] \rhd_\bullet T[u/x] : s \\ \Gamma, \Delta[u/x] \vdash_\bullet \Sigma y : T[u/x].V[u/x] : s \qquad \Gamma, \Delta[u/x] \vdash_\bullet v[u/x] : V'[u/x] \quad \Gamma, \Delta[u/x] \vdash_\bullet V'[u/x] \rhd_\bullet V[u/x][t[u/x]/y]\end{array}}{\Gamma, \Delta[u/x] \vdash_\bullet (t[u/x], v[u/x])_{\Sigma y:T[u/x].V[u/x]} : \Sigma y : T[u/x].V[u/x]}$$

On a $V[u/x][t[u/x]/y] = V[t/y][u/x]$ puisque $y \notin \mathcal{FV}(u)$. Ici on a les coercions $\Gamma, x : U, \Delta \vdash_{CCI} c : T' \rhd_\bullet T$: et $\Gamma, x : U, \Delta \vdash_{CCI} d : V' \rhd_\bullet V[t/y]$:. Par induction on obtient $\Gamma, \Delta[u/x] \vdash_{CCI} c' : T'[u/x] \rhd_\bullet T[u/x]$: et $\Gamma, \Delta[u/x] \vdash_{CCI} c' : V'[u/x] \rhd_\bullet V[t/y][u/x]$: avec $c' \equiv c[[\![u]\!]_\Gamma/x]$ et $d' \equiv d[[\![u]\!]_\Gamma/x]$.

$$
\begin{aligned}
& [\![(t,v)_{\Sigma x:T.V}]\!]_{\Gamma,x:U,\Delta}[[\![u]\!]_\Gamma/x] \\
& \quad \textbf{\{ Définition de l'interprétation \}} \\
\hat{=} \;& (c[[\![t]\!]_{\Gamma,x:U,\Delta}], d[[\![v]\!]_{\Gamma,x:U,\Delta}])_{[\![\Sigma x:T.V]\!]_{\Gamma,x:U,\Delta}}[[\![u]\!]_\Gamma/x] \\
& \quad \textbf{\{ Application de l'hypothèse de récurrence \}} \\
\equiv \;& (c'[[\![t[u/x]]\!]_{\Gamma,\Delta[u/x]}], d'[[\![v[u/x]]\!]_{\Gamma,\Delta[u/x]}])_{[\![(\Sigma x:T.V)[u/x]]\!]_{\Gamma,\Delta[u/x]}} \\
& \quad \textbf{\{ Définition de l'interprétation \}} \\
\hat{=} \;& [\![((t,u)_{\Sigma x:T.V})[u/x]]\!]_{\Gamma,\Delta[u/x]}
\end{aligned}
$$

– Pi-1, Pi-2 : On a

$$\frac{\Gamma, x : U, \Delta \vdash_\bullet t : S \qquad \mu_\bullet(S) = \Sigma y : T.U}{\Gamma, x : U, \Delta \vdash_\bullet \pi_i\ t : \_}$$

$$\frac{\Gamma, \Delta[u/x] \vdash_\bullet t[u/x] : S[u/x] \qquad \mu_\bullet(S[u/x]) = (\Sigma y : T.U)[u/x]}{\Gamma, \Delta[u/x] \vdash_\bullet \pi_i\ t[u/x] : \_}$$

Encore une fois on obtient la coercion $\Gamma, \Delta[u/x] \vdash_{CCI} c' : S[u/x] \rhd_\bullet (\Sigma y : T.U)[u/x]$: par induction sur la dérivation de $\Gamma, x : U, \Delta \vdash_{CCI} c : S \rhd_\bullet \Sigma y : T.U$:. On a $c' \equiv c[[\![u]\!]_\Gamma/x]$.

$$
\begin{aligned}
& [\![\pi_i\ t]\!]_{\Gamma,x:U,\Delta}[[\![u]\!]_\Gamma/x] \\
& \quad \textbf{\{ Définition de l'interprétation \}} \\
\hat{=} \;& \pi_i\ c[[\![t]\!]_{\Gamma,x:U,\Delta}][[\![u]\!]_\Gamma/x] \\
& \quad \textbf{\{ Application de l'hypothèse de récurrence \}} \\
\equiv \;& \pi_i\ c'[[\![t[u/x]]\!]_{\Gamma,\Delta[u/x]}] \\
& \quad \textbf{\{ Définition de l'interprétation \}} \\
\hat{=} \;& [\![\pi_i\ t[u/x]]\!]_{\Gamma,\Delta[u/x]}
\end{aligned}
$$

□

On va maintenant étendre la relation de coercion aux contextes de manière canonique.

Définition 2.3.12 (Coercion de contextes). *On définit inductivement la coercion de deux contextes de coercions algorithmiques par les règles suivantes :*

– $[] \rhd_\bullet []$
– $(\Gamma, x : T) \rhd_\bullet (\Gamma', x : T')$ *si* $\Gamma \rhd_\bullet \Gamma'$ *et* $T \rhd_\bullet T'$.

De même pour les coercions explicites dérivées par le jugement $\Gamma \vdash_{CCI} c : T \rhd_\bullet S$:.

Définition 2.3.13 (Coercion explicites de contextes). *On définit inductivement la coercion de deux contextes de coercions explicites par les règles suivantes :*

– $[] \rhd_\bullet []$
– $\rho, c : (\Gamma, x : T) \rhd_\bullet (\Gamma', x : T')$ *si* $\rho : \Gamma \rhd_\bullet \Gamma'$ *et* $\Gamma \vdash_{CCI} c : T \rhd_\bullet T'$:.

Clairement toute coercion de contexte algorithmique correspond à une coercion de contexte explicite et vice-versa.

Définition 2.3.14 (Extension de la substitution aux coercions de contextes). *On définit la substitution d'une coercion de contexte inductivement :*

– $t[[]] = t$
– $t[(\rho, c) : (\Gamma, x : T) \rhd_\bullet (\Gamma', x : T')] = t[\rho : \Gamma \rhd_\bullet \Gamma'][c[x]/x]$

Lemme 2.3.15 (Stabilité par affaiblissement des coercions). *Si* $\Gamma \vdash_{CCI} c : T \rhd_\bullet T' :$*, alors pour tout* $\Delta, \rho : \Delta \rhd_\bullet \Gamma$ *tels que* $[\![T']\!]_\Delta \equiv [\![T']\!]_\Gamma[\rho]$*, on a* $\Delta \vdash_{CCI} c' : T \rhd_\bullet T' :$ *et* $c' \equiv c[\rho]$ *avec une dérivation de même taille.*

Démonstration. Par induction sur la dérivation de coercion $\Gamma \vdash_{CCI} c : T \rhd_\bullet T :$.

– $\rhd$-$\downarrow$: On a $\Gamma \vdash_{CCI} c : T^\downarrow \rhd_\bullet T'^\downarrow :$. Par induction, $\Delta \vdash_{CCI} c[\rho] : T^\downarrow \rhd_\bullet T'^\downarrow :$. On peut donc dériver $\Delta \vdash_{CCI} c[\rho] : T \rhd_\bullet T' :$ par $\rhd$-$\downarrow$.

– $\rhd$-Conv : On a $T \equiv_{\beta\pi} T'$ et $c = \bullet$. On peut donc dériver $\Delta \vdash_{CCI} c' = \bullet : T \rhd_\bullet T' :$, on a bien $c' \equiv c[\rho]$.

– $\rhd$-Prod : On a :

$$\frac{\Gamma \vdash_{CCI} c_1 : C \rhd_\bullet A : \qquad \Gamma, x : C \vdash_{CCI} c_2 : B \rhd_\bullet D :}{\Gamma \vdash_{CCI} \lambda x : [\![C]\!]_\Gamma.\ c_2[\bullet\ c_1[x]] : \Pi x : A.B \rhd_\bullet \Pi x : C.D :}$$

Par induction on a $\Delta \vdash_{CCI} c_1[\rho] : C \rhd_\bullet A :$. On peut définir la coercion $\sigma = \rho, \bullet : (\Delta, x : C) \rhd_\bullet (\Gamma, x : C)$ et obtenir par induction : $\Delta, x : C \vdash_{CCI} c_2' : B \rhd_\bullet D :$ avec $c_2' \equiv c_2[\sigma]$.
On peut alors appliquer $\rhd$-Prod pour obtenir :

$$\Delta \vdash_{CCI} c' = \lambda x : [\![C]\!]_\Delta.\ c_2[\sigma][\bullet\ c_1[\rho][x]] : \Pi x : A.B \rhd_\bullet \Pi x : C.D :$$

On a :

$$\begin{aligned}
& (\lambda x : [\![C]\!]_\Gamma.\ c_2[\bullet\ c_1[x]])[\rho] \\
& \quad \{\ \textbf{Définition de la substitution}\ \} \\
=\ & \lambda x : [\![C]\!]_\Gamma[\rho].(c_2[\rho])[\bullet\ (c_1[\rho])[x]] \\
& \quad \{\ \textbf{Condition}\ [\![T']\!]_\Delta \equiv [\![T']\!]_\Gamma[\rho]\ \} \\
\equiv\ & \lambda x : [\![C]\!]_\Delta.(c_2[\rho])[\bullet\ (c_1[\rho])[x]] \\
& \quad \{\ \textbf{Coercion identité dans}\ \sigma\ \} \\
=\ & \lambda x : [\![C]\!]_\Delta.(c_2[\sigma])[\bullet\ (c_1[\rho])[x]]
\end{aligned}$$

– $\rhd$-Sum :

$$\frac{\Gamma \vdash_{CCI} c_1 : A \rhd_\bullet C : \qquad \Gamma, x : A \vdash_{CCI} c_2 : B \rhd_\bullet D :}{\Gamma \vdash_{CCI} (c_1[\pi_1\ \bullet], c_2[\pi_1\ \bullet/x][\pi_2\ \bullet])_{[\![\Sigma x:C.D]\!]_\Gamma} : \Sigma x : A.B \rhd_\bullet \Sigma x : C.D :}$$

Par induction on a $\Delta \vdash_{CCI} c_1[\rho] : A \rhd_\bullet C :$. On peut définir la coercion $\sigma = \rho, \bullet : (\Delta, x : A) \rhd_\bullet (\Gamma, x : A)$ et obtenir par induction : $\Delta, x : A \vdash_{CCI} c_2[\sigma] : B \rhd_\bullet D :$
On peut alors appliquer $\rhd$-Sum pour obtenir :

$$\Delta \vdash_{CCI} c' = ((c_1[\rho])[\pi_1\ \bullet], (c_2[\sigma])[\pi_1\ \bullet/x][\pi_2\ \bullet])_{[\![\Sigma x:C.D]\!]_\Delta} : \Sigma x : A.B \rhd_\bullet \Sigma x : C.D :$$

On a :

$$\begin{aligned}
& c[\rho] \\
& \quad \{\ \textbf{Définition}\ \} \\
\hat{=}\ & ((c_1[\pi_1\ \bullet], c_2[\pi_2\ \bullet][\pi_1\ \bullet/x])_{[\![\Sigma x:C.D]\!]_\Gamma})[\rho] \\
& \quad \{\ \textbf{Définition de la substitution,}\ x \notin \rho\ \} \\
=\ & (c_1[\rho][\pi_1\ \bullet], c_2[\rho][\pi_2\ \bullet][\pi_1\ \bullet/x])_{[\![\Sigma x:C.D]\!]_\Gamma[\rho]} \\
& \quad \{\ \textbf{Condition}\ \} \\
=\ & (c_1[\rho][\pi_1\ \bullet], c_2[\rho][\pi_2\ \bullet][\pi_1\ \bullet/x])_{[\![\Sigma x:C.D]\!]_\Delta} \\
& \quad \{\ \textbf{Coercion identité dans}\ \sigma\ \} \\
=\ & (c_1[\rho][\pi_1\ \bullet], c_2[\sigma][\pi_2\ \bullet][\pi_1\ \bullet/x])_{[\![\Sigma x:C.D]\!]_\Delta} \\
& \quad \{\ \textbf{Définition}\ \} \\
\hat{=}\ & c'
\end{aligned}$$

– $\rhd$-Proof :

$$\frac{\Gamma \vdash_{CCI} d : T \rhd_\bullet U :}{\Gamma \vdash_{CCI} c = \text{elt } [\![U]\!]_\Gamma\ [\![(\lambda x : U.P)]\!]_\Gamma\ d\ ?_{[\![P]\!]_{\Gamma,x:U}[d/x]} : T \rhd_\bullet \{\ x : U \mid P\ \} :}$$

Par induction, on a $\Delta \vdash_{CCI} d[\rho] : T \rhd_\bullet U :$. On peut donc dériver :

$$\Delta \vdash_{CCI} c' = \text{elt } [\![U]\!]_\Delta\ [\![(\lambda x : U.P)]\!]_\Delta\ d[\rho]\ ?_{[\![P]\!]_{\Delta,x:U}[d[\rho]/x]} : T \rhd_\bullet \{\ x : U \mid P\ \} :$$

On a bien $c[\rho] \equiv c'$, car :

$$\begin{aligned}
& c[\rho] \\
& \quad \textbf{\{ Définition \}} \\
\triangleq\ & (\text{elt } [\![U]\!]_\Gamma\ [\![(\lambda x : U.P)]\!]_\Gamma\ d\ ?_{[\![P]\!]_{\Gamma,x:U}[d/x]})[\rho] \\
& \quad \textbf{\{ Condition } [\![\{\ x : U \mid P\ \}]\!]_\Delta \equiv [\![\{\ x : U \mid P\ \}]\!]_\Gamma[\rho] \textbf{ \}} \\
\equiv\ & \text{elt } [\![U]\!]_\Delta\ [\![(\lambda x : U.P)]\!]_\Delta\ d[\rho]\ ?_{[\![P]\!]_{\Gamma,x:U}[d/x][\rho]} \\
& \quad \textbf{\{ Définition de la substitution \}} \\
=\ & \text{elt } [\![U]\!]_\Delta\ [\![(\lambda x : U.P)]\!]_\Delta\ d[\rho]\ ?_{[\![P]\!]_{\Gamma,x:U}[\rho][d[\rho]/x]} \\
& \quad \textbf{\{ Condition } [\![P]\!]_{\Delta,x:U} \equiv [\![P]\!]_{\Gamma,x:U}[\rho] \textbf{ \}} \\
\equiv\ & \text{elt } [\![U]\!]_\Delta\ [\![(\lambda x : U.P)]\!]_\Delta\ d[\rho]\ ?_{[\![P]\!]_{\Delta,x:U}[d[\rho]/x]} \\
& \quad \textbf{\{ Définition \}} \\
\triangleq\ & c'
\end{aligned}$$

– $\rhd$-Subset : Direct par induction.

□

Pour montrer le prochain lemme, on a besoin d'une propriété essentielle de la coercion, la transitivité.

Lemme 2.3.16 (Transitivité de la coercion avec affaiblissement). *S'il existe c_1, c_2 tels que $\Gamma \vdash_{CCI} c_1 : S \rhd_\bullet T :$ et $\Delta \vdash_{CCI} c_2 : T \rhd_\bullet U :$ avec $\rho : \Delta \rhd_\bullet \Gamma$ et $[\![T]\!]_\Gamma[\rho] \equiv [\![T]\!]_\Delta$, alors $\exists! c, \Delta \vdash_{CCI} c : S \rhd_\bullet U :$ et $c \equiv c_2 \circ c_1[\rho]$.*

Démonstration. Par induction lexicographique sur la paire de dérivations de c_1 et c_2.

– $\rhd$-Conv : On va traiter les cas où cette règle est utilisée en racine d'une des deux dérivations, d'abord à gauche puis à droite.

$$\frac{S \equiv_{\beta\pi} T}{\Gamma \vdash_{CCI} c_1 = \bullet : S \rhd_\bullet T :} \qquad \Delta \vdash_{CCI} c_2 : T \rhd_\bullet U :$$

Les conditions de bord de $\rhd$-Conv nous donnent comme hypothèses que S et T sont en forme normale de tête et que $S \neq \Pi, \Sigma, \{|\}$. Par inversion de la $\beta\rho$-équivalence $S \equiv_{\beta\pi} T$, on a aussi $T \neq \Pi, \Sigma, \{|\}$. Les seules règles pouvant s'appliquer a la fin de la dérivation de c_2 sont donc $\rhd$-Conv, $\rhd$-$\downarrow$ et $\rhd$-Proof.

- $\rhd$-Conv : Alors on a $U = U^\downarrow \neq \Pi, \Sigma, \{|\}$. La coercion composée est alors $\bullet \circ \bullet$. C'est bien la coercion dérivée pour le jugement $\Delta \vdash_{CCI} \bullet : S \rhd_\bullet U :$ par $\rhd$-Conv.
- $\rhd$-$\downarrow$: On a alors $\Delta \vdash_{CCI} c_2 : T^\downarrow \rhd_\bullet U^\downarrow :$. Comme $T = T^\downarrow$, on peut appliquer l'hypothèse d'induction pour obtenir une coercion c telle que $\Delta \vdash_{CCI} c : S \rhd_\bullet U^\downarrow :$ et $c \equiv c_2 \circ c_1[\rho]$. Une application de $\rhd$-$\downarrow$ suffit pour obtenir une dérivation de $\Delta \vdash_{CCI} c : S \rhd_\bullet U :$ avec $c \equiv c_2 \circ c_1[\rho]$.
- $\rhd$-Proof : Ici on a :

$$\frac{\Delta \vdash_{CCI} d : T \rhd_\bullet U' :}{\Delta \vdash_{CCI} c_2 = \text{elt } [\![U']\!]_\Delta\ [\![\lambda x : U'.P]\!]_\Delta\ d\ ?_{[\![P]\!]_\Delta[d/x]} : T \rhd_\bullet U = \{\ x : U' \mid P\ \} :}$$

Par induction, il existe une coercion $d' \equiv d \circ c_1[\rho] = d$ telle que $\Delta \vdash_{CCI} d' : S \rhd_\bullet U' :$. On applique $\rhd$-Proof pour obtenir la coercion c de S à U. Clairement $c \equiv c_2 \circ c_1[\rho] = c_2 \circ \bullet = c_2$.

Supposons maintenant que la dérivation de c_2 termine par une application de $\rhd$-Conv. Alors T et U sont en forme normale de tête et $T, U \neq \Pi, \Sigma, \{|\}$. Les seules règles pouvant apparaître en racine de la dérivation de c_1 sont donc $\rhd$-Conv, $\rhd$-$\downarrow$ et $\rhd$-Subset.

- $\rhd$-Conv : Alors on a $S = S^\downarrow \neq \Pi, \Sigma, \{|\}$. La coercion composée est alors $\bullet \circ \bullet$. C'est bien la coercion dérivée pour le jugement $\Delta \vdash_{CCI} \bullet : S \rhd_\bullet U :$ par $\rhd$-Conv.
- $\rhd$-$\downarrow$: On a alors $\Gamma \vdash_{CCI} c_1 : S^\downarrow \rhd_\bullet T^\downarrow :$. Comme $T = T^\downarrow$, on peut appliquer l'hypothèse d'induction pour obtenir une coercion c telle que $\Delta \vdash_{CCI} c : S^\downarrow \rhd_\bullet U :$ et $c \equiv c_2 \circ c_1[\rho]$. Une application de $\rhd$-$\downarrow$ suffit pour obtenir une dérivation de $\Delta \vdash_{CCI} c : S \rhd_\bullet U :$ avec $c \equiv c_2 \circ c_1[\rho]$.
- $\rhd$-Subset : Ici on a :

$$\frac{\Gamma \vdash_{CCI} d : S' \rhd_\bullet T :}{\Gamma \vdash_{CCI} c_1 = d[\sigma_1\ \bullet] : S = \{\ x : S' \mid P\ \} \rhd_\bullet T :}$$

Par induction, il existe une coercion $d' \equiv c_2 \circ d[\rho] = d[\rho]$ telle que $\Delta \vdash_{CCI} d' : S' \rhd_\bullet U :$. On applique $\rhd$-Subset pour obtenir la coercion $c = d[\rho][\sigma_1\ \bullet]$ de S à U. On a

$$c = d[\rho][\sigma_1\ \bullet] = d[\sigma_1\ \bullet][\rho] \equiv c_2 \circ c_1[\rho]$$

– $\rhd$-$\downarrow$:

$$\frac{\Gamma \vdash_{CCI} c : S^\downarrow \rhd_\bullet T^\downarrow :}{\Gamma \vdash_{CCI} c : S \rhd_\bullet T :} \qquad \Delta \vdash_{CCI} d : T \rhd_\bullet U :$$

Si $T = T^\downarrow$ alors c'est trivial par induction et application de $\rhd$-$\downarrow$. Sinon, la seule règle permettant de dériver $\Delta \vdash_{CCI} d : T \rhd_\bullet U :$ est $\rhd$-$\downarrow$. Il suffit alors d'appliquer l'hypothèse d'induction pour obtenir une dérivation de $\Delta \vdash_{CCI} c : S^\downarrow \rhd_\bullet U^\downarrow :$ avec $c \equiv c_2 \circ c_1[\rho]$ puis $\rhd$-$\downarrow$ nous permet de construire le jugement $\Gamma \vdash_{CCI} c : S \rhd_\bullet U :$.
De même si l'on a une application de $\rhd$-$\downarrow$ à la racine de la dérivation de droite.
On peut donc se ramener au cas où $\rhd$-Conv et $\rhd$-$\downarrow$ ne sont appliquées à la racine d'aucune des deux dérivations.

– $\rhd$-Prod :

$$\frac{\Gamma \vdash_{CCI} c_1 : X' \rhd_\bullet X : \qquad \Gamma, x : X' \vdash_{CCI} c_2 : Y \rhd_\bullet Y' :}{\Gamma \vdash_{CCI} c = \lambda x : [\![X']\!]_\Gamma.c_2[\bullet\ c_1[x]] : \Pi x : X.Y \rhd_\bullet \Pi x : X'.Y' :} \qquad \Delta \vdash_{CCI} d : \Pi x : X'.Y' \rhd_\bullet U :$$

Par induction sur la dérivation $\Delta \vdash_{CCI} d : \Pi x : X'.Y' \rhd_\bullet U :$. Seules deux règles peuvent s'appliquer à la racine :

- $\rhd$-Prod : On a $U = \Pi x : S.T$ et la dérivation a la forme :

$$\frac{\Delta \vdash_{CCI} d_1 : S \rhd_\bullet X' : \qquad \Delta, x : S \vdash_{CCI} d_2 : Y' \rhd_\bullet T :}{\Delta \vdash_{CCI} d = (\lambda x : [\![S]\!]_\Delta.d_2[\bullet\ d_1[x]]) : \Pi x : X'.Y' \rhd_\bullet \Pi x : S.T :}$$

On peut construire une coercion de contextes de $\theta = \rho, d_1 : (\Delta, x : S) \rhd_\bullet (\Gamma, x : X')$. Par le lemme 2.3.15 on obtient la dérivation $\Delta, x : S \vdash_{CCI} c'_2 : Y \rhd_\bullet Y' :$ de même taille que la dérivation de c_2 telle que $c'_2 \equiv c_2[\theta]$. On obtient de même $\Delta \vdash_{CCI} c'_1 : X' \rhd_\bullet X :$ avec $c'_1 \equiv c_1[\rho]$. En utilisant une coercion de contextes partout l'identité, on obtient par induction avec les dérivations de c'_1 et d_1 d'une part et c'_2 et d_2 d'autre part, deux coercions e_1, e_2 telles que :

$$\Delta \vdash_{CCI} e_1 \equiv c_1[\rho] \circ d_1 : S \rhd_\bullet X : \wedge \Delta, x : S \vdash_{CCI} e_2 \equiv d_2 \circ c_2[\rho] : Y \rhd_\bullet T :$$

On en déduit :

$$\frac{\Delta \vdash_{CCI} e_1 : S \rhd_\bullet X : \qquad \Delta, x : S \vdash_{CCI} e_2 : Y \rhd_\bullet T :}{\Delta \vdash_{CCI} e = \lambda x : [\![S]\!]_\Delta . e_2[\bullet\ e_1[x]] : \Pi x : X.Y \rhd_\bullet \Pi x : S.T :}$$

On a bien $e \equiv d \circ c[\rho]$:

$$\begin{array}{lll}
 & d \circ c[\rho] & \\
 & \quad \{ \textbf{Définition de } c \textbf{ et } d \} & \\
\hat{=} & (\lambda x : [\![S]\!]_\Delta . d_2[\bullet\ d_1[x]]) \circ (\lambda y : [\![X']\!]_\Gamma . c_2[\bullet\ c_1[y]])[\rho] & \\
 & \quad \{ \textbf{Composition des contextes} \} & \\
= & \lambda x : [\![S]\!]_\Delta . d_2[(\lambda y : [\![X']\!]_\Gamma . c_2[\bullet\ c_1[y]])[\rho]\ d_1[x]] & \\
 & \quad \{ \textbf{Substitution dans l'abstraction} \} & \\
= & \lambda x : [\![S]\!]_\Delta . d_2[(\lambda y : [\![X']\!]_\Gamma [\rho] . c_2[\bullet\ c_1[y]][\rho])\ d_1[x]] & \\
 & \quad \{ \textbf{Réduction} \} & \\
\rightarrow_\beta & \lambda x : [\![S]\!]_\Delta . d_2[(c_2[\bullet\ c_1[y]][\rho])[d_1[x]/y]] & \\
 & \quad \{ y \notin \rho \} & \\
= & \lambda x : [\![S]\!]_\Delta . d_2[(c_2[\rho][\bullet\ c_1[\rho][y]])[d_1[x]/y]] & \\
 & \quad \{ \textbf{Définition de } \theta, y \notin d1[x] \} & \\
\hat{=} & \lambda x : [\![S]\!]_\Delta . d_2[c_2[\theta][\bullet\ c_1[\theta][d_1[x]]]] & \\
 & \quad \{ d_2 \circ c_2[\theta] \equiv e_2 \} & \\
\equiv & \lambda x : [\![S]\!]_\Delta . e_2[\bullet\ c_1[\theta][d_1[x]]] & \\
 & \quad \{ x \notin c_1 \Rightarrow c_1[\theta] = c_1[\rho] \} & \\
\equiv & \lambda x : [\![S]\!]_\Delta . e_2[\bullet\ e_1[x]] & \\
 & \quad \{ \textbf{Définition} \} & \\
\hat{=} & e &
\end{array}$$

- $\rhd$-Proof : Ici $U = \{ y : U' \mid P \}$ et la dérivation commence par :

$$\frac{\Delta \vdash_{CCI} e : \Pi x : X'.Y' \rhd_\bullet U' :}{\Delta \vdash_{CCI} d = \text{elt}\ [\![U']\!]_\Delta\ [\![\lambda x : U'.P]\!]_\Delta\ e\ ?_{[\![P]\!]_\Delta[e/x]} : \Pi x : X'.Y' \rhd_\bullet \{ y : U' \mid P \} :}$$

Par induction on a $\Delta \vdash_{CCI} f \equiv e \circ c[\rho] : \Pi x : X.Y \rhd_\bullet U' :$. On peut donc dériver :

$$\frac{\Delta \vdash_{CCI} f : \Pi x : X.Y \rhd_\bullet U' :}{\Delta \vdash_{CCI} d' = \text{elt}\ [\![U']\!]_\Delta\ [\![\lambda x : U'.P]\!]_\Delta\ f\ ?_{[\![P]\!]_\Delta[f/x]} : \Pi x : X.Y \rhd_\bullet U :}$$

On peut vérifier que $d' \equiv d \circ c[\rho]$:

$$\begin{array}{lll}
d \circ c[\rho] & \hat{=} & (\text{elt}\ [\![U']\!]_\Delta\ [\![\lambda x : U'.P]\!]_\Delta\ e\ ?_{[\![P]\!]_\Delta[e/x]})[c[\rho]] \\
 & = & \text{elt}\ [\![U']\!]_\Delta\ [\![\lambda x : U'.P]\!]_\Delta\ e[c[\rho]]\ ?_{[\![P]\!]_\Delta[e[c[\rho]]/x]} \\
 & \equiv & \text{elt}\ [\![U']\!]_\Delta\ [\![\lambda x : U'.P]\!]_\Delta\ f\ ?_{[\![P]\!]_\Delta[f/x]} \\
 & \hat{=} & d'
\end{array}$$

– $\rhd$-Sum : De façon équivalente à $\rhd$-Prod, on fait le cas où $\rhd$-Sum est utilisée à la fin de la dérivation de d.

$$\Gamma \vdash_{CCI} c : T \rhd_\bullet \Sigma x : X'.Y' : \qquad \frac{\Delta \vdash_{CCI} d_1 : X' \rhd_\bullet X : \qquad \Delta, x : X' \vdash_{CCI} d_2 : Y' \rhd_\bullet Y :}{\Delta \vdash_{CCI} d = (d_1[\pi_1\ \bullet], d_2[\pi_2\ \bullet][\pi_1\ \bullet/x])_{[\![\Sigma x:X.Y]\!]_\Delta} : \Sigma x : X'.Y' \rhd_\bullet \Sigma x : X.Y :}$$

Par induction sur la dérivation de $\Gamma \vdash_{CCI} c : T \rhd_\bullet \Sigma x : X'.Y' :$:

- $\rhd$-Subset : On a :

$$\frac{\Gamma \vdash_{CCI} c_1 : T \rhd_\bullet \Sigma x : X'.Y' :}{\Gamma \vdash_{CCI} c = c_1[\sigma_1 \bullet] : \{ y : T \mid P \} \rhd_\bullet \Sigma x : X'.Y' :}$$

Par induction, $\Delta \vdash_{CCI} f \equiv d \circ c_1[\rho] : T \rhd_\bullet \Sigma x : X.Y :$, on a donc :

$$\frac{\Delta \vdash_{CCI} f \equiv d \circ c_1[\rho] : T \rhd_\bullet \Sigma x : X.Y :}{\Delta \vdash_{CCI} c' = f[\sigma_1 \bullet] : \{ y : T \mid P \} \rhd_\bullet \Sigma x : X.Y :}$$

On a bien :

$$\begin{aligned} d \circ c[\rho] &\equiv d \circ c_1[\sigma_1 \bullet][\rho] \\ &= d \circ c_1[\rho][\sigma_1 \bullet] \\ &\equiv f[\sigma_1 \bullet] \\ &\hat{=} c' \end{aligned}$$

- $\rhd$-Sum : On a :

$$\frac{\Gamma \vdash_{CCI} c_1 : S \rhd_\bullet X' : \qquad \Gamma, x : S \vdash_{CCI} c_2 : T \rhd_\bullet Y' :}{\Gamma \vdash_{CCI} c = (c_1[\pi_1 \bullet], c_2[\pi_1 \bullet / x][\pi_2 \bullet])_{[\![\Sigma x:X'.Y']\!]_\Gamma} : \Sigma x : S.T \rhd_\bullet \Sigma x : X'.Y' :}$$

Par affaiblissement pour les coercions (lemme 2.3.15) on a $\Delta \vdash_{CCI} c'_1 : S \rhd_\bullet X'$: avec $c'_1 \equiv c_1[\rho]$. On peut aussi construire la coercion de contexte $\theta = \rho, \bullet : (\Delta, x : S) \rhd_\bullet (\Gamma, x : S)$. Par le même lemme on obtient $\Delta, x : S \vdash_{CCI} c'_2 : T \rhd_\bullet Y'$: avec $c'_2 \equiv c_2[\theta] = c_2[\rho]$. On peut enfin construire le jugement $\Delta, x : S \vdash_{CCI} d'_2 : Y' \rhd_\bullet Y$: avec $d'_2 \equiv d_2[c'_1[x]/x]$.

Par induction en utilisant une coercion de contexte partout l'identité on a donc :

$$\frac{\Delta \vdash_{CCI} e_1 \equiv d_1 \circ c_1[\rho] : S \rhd_\bullet X : \qquad \Delta, x : S \vdash_{CCI} e_2 \equiv d'_2 \circ c'_2 : T \rhd_\bullet Y :}{\Delta \vdash_{CCI} e = (e_1[\pi_1 \bullet], e_2[\pi_1 \bullet / x][\pi_2 \bullet])_{[\![\Sigma x:X.Y]\!]_\Delta} : \Sigma x : S.T \rhd_\bullet \Sigma x : X.Y :}$$

On peut vérifier :

$$
\begin{array}{lll}
 & d \circ c[\rho] & \\
 & \quad \{ \textbf{Définition de } d \} & \\
\hat{=} & (d_1[\pi_1 \bullet], d_2[\pi_1 \bullet/x][\pi_2 \bullet])_{[\![\Sigma x{:}X.Y]\!]_\Delta}[c[\rho]] & \\
 & \quad \{ \textbf{Substitution} \} & \\
= & (d_1[\pi_1\ c[\rho]], d_2[\pi_1\ c[\rho]/x][\pi_2\ c[\rho]])_{[\![\Sigma x{:}X.Y]\!]_\Delta} & \\
 & \quad \{ \textbf{Réduction} \} & \\
\to_\pi & (d_1[c_1[\rho][\pi_1 \bullet]], d_2[(c_1[\pi_1 \bullet])[\rho]/x][c_2[\rho][\pi_1 \bullet/x][\pi_2 \bullet]])_{[\![\Sigma x{:}X.Y]\!]_\Delta} & \\
 & \quad \{ \textbf{Définition de } e_1 \} & \\
\equiv & (e_1[\pi_1 \bullet], d_2[c_1[\rho][\pi_1 \bullet]/x][c_2[\rho][\pi_1 \bullet/x][\pi_2 \bullet]])_{[\![\Sigma x{:}X.Y]\!]_\Delta} & \\
 & \quad \{ \textbf{Définition de } c'_1 \} & \\
\equiv & (e_1[\pi_1 \bullet], d_2[c'_1[\pi_1 \bullet]/x][c_2[\rho][\pi_1 \bullet/x][\pi_2 \bullet]])_{[\![\Sigma x{:}X.Y]\!]_\Delta} & \\
 & \quad \{ \textbf{Définition de } d'_2 \} & \\
\equiv & (e_1[\pi_1 \bullet], d'_2[\pi_1 \bullet/x][c_2[\rho][\pi_1 \bullet/x][\pi_2 \bullet]])_{[\![\Sigma x{:}X.Y]\!]_\Delta} & \\
 & \quad \{ x \notin c_2[\pi_1 \bullet/x] \} & \\
\equiv & (e_1[\pi_1 \bullet], d'_2[c_2[\rho][\pi_2 \bullet]][\pi_1 \bullet/x])_{[\![\Sigma x{:}X.Y]\!]_\Delta} & \\
 & \quad \{ \textbf{Définition de } c'_2 \} & \\
\equiv & (e_1[\pi_1 \bullet], d'_2[c'_2[\pi_2 \bullet]][\pi_1 \bullet/x])_{[\![\Sigma x{:}X.Y]\!]_\Delta} & \\
 & \quad \{ \textbf{Définition de } e_2 \} & \\
\equiv & (e_1[\pi_1 \bullet], e'_2[\pi_2 \bullet][\pi_1 \bullet/x])_{[\![\Sigma x{:}X.Y]\!]_\Delta} & \\
 & \quad \{ x \notin \pi_2 \bullet \} & \\
= & (e_1[\pi_1 \bullet], e_2[\pi_1 \bullet/x][\pi_2 \bullet])_{[\![\Sigma x{:}X.Y]\!]_\Delta} & \\
 & \quad \{ \textbf{Définition de } e \} & \\
\hat{=} & e &
\end{array}
$$

– $\rhd$-Proof :

$$\frac{\Gamma \vdash_{CCI} e : S \rhd_\bullet T :}{\Gamma \vdash_{CCI} c_1 = \text{elt } [\![T]\!]_\Gamma\ [\![\lambda x : T.P]\!]_\Gamma\ e\ ?_{[\![P]\!]_{\Gamma,x:T}[e/x]} : S \rhd_\bullet \{ x : T \mid P \} :} \qquad \Delta \vdash_{CCI} c_2 : \{ x : T \mid P \} \rhd_\bullet U :$$

Par induction sur la dérivation de c_2.

- $\rhd$-Subset : On a une dérivation $\Delta \vdash_{CCI} c'_2 : T \rhd_\bullet U :$, donc par induction on a $\Delta \vdash_{CCI} c' : S \rhd_\bullet U :$ avec $c' \equiv c'_2 \circ e[\rho]$.
  On a bien $c' = c_2 \circ c_1[\rho] :$

$$
\begin{array}{lll}
c_2 \circ c_1[\rho] & \equiv & c'_2[\sigma_1 \bullet] \circ c_1[\rho] \\
 & = & c'_2[\sigma_1 \bullet] \circ (\text{elt } [\![T]\!]_\Gamma\ [\![\lambda x : T.P]\!]_\Gamma\ e\ ?_{[\![P]\!]_{\Gamma,x:T}[e/x]})[\rho] \\
 & \to_{\sigma_1} & c'_2[e[\rho]] \\
 & \equiv & c'
\end{array}
$$

- $\rhd$-Proof : Alors on a :

$$\frac{\Delta \vdash_{CCI} c'_2 : \{ x : T \mid P \} \rhd_\bullet U' :}{\Delta \vdash_{CCI} c_2 = \text{elt } [\![U']\!]_\Delta\ [\![\lambda x : U'.P']\!]_\Delta\ c'_2\ ?_{[\![P']\!]_{\Delta,x:U'}[c'_2/x]} : \{ x : T \mid P \} \rhd_\bullet \{ x : U' \mid P' \} :}$$

On peut appliquer l'hypothèse d'induction pour obtenir une dérivation de $\Delta \vdash_{CCI} d \equiv c_2' \circ c_1[\rho] : S \rhd_\bullet U'$: et par application de $\rhd$-Proof on obtient une coercion c' :

$$\begin{aligned} c_2 \circ c_1[\rho] &\triangleq (\text{elt } [\![U']\!]_\Delta \, [\![\lambda x : U'.P']\!]_\Delta \, c_2' \, ?_{[\![P']\!]_{\Delta,x:U'}[c_2'/x]}) \circ c_1[\rho] \\ &= \text{elt } [\![U']\!]_\Delta \, [\![\lambda x : U'.P']\!]_\Delta \, c_2'[c_1[\rho]] \, ?_{[\![P']\!]_{\Delta,x:U'}[c_2'[c_1[\rho]]/x]} \\ &\equiv \text{elt } [\![U']\!]_\Delta \, [\![\lambda x : U'.P']\!]_\Delta \, d \, ?_{[\![P']\!]_{\Delta,x:U'}[d/x]} \\ &\triangleq c' \end{aligned}$$

– $\rhd$-Subset :

$$\frac{\Gamma \vdash_{CCI} c : S' \rhd_\bullet T :}{\Gamma \vdash_{CCI} d = c[\sigma_1 \bullet] : S = \{ x : S' \mid P \} \rhd_\bullet T :} \qquad \Delta \vdash_{CCI} e : T \rhd_\bullet U :$$

Par induction il existe une dérivation de $\Delta \vdash_{CCI} f \equiv e \circ c[\rho] : S' \rhd_\bullet U :$. On peut donc dériver :

$$\frac{\Delta \vdash_{CCI} f : S' \rhd_\bullet U :}{\Delta \vdash_{CCI} f[\sigma_1 \bullet] : S = \{ x : S' \mid P \} \rhd_\bullet U :}$$

On a bien $f[\sigma_1 \bullet] \equiv (e \circ c[\rho])[\sigma_1 \bullet] = e \circ c[\rho][\sigma_1 \bullet] = e \circ c[\sigma_1 \bullet][\rho] \triangleq e \circ d[\rho]$.
Dans le cas où $\rhd$-Subset est utilisée à droite, la seule règle applicable à gauche est $\rhd$-Proof et l'on a déjà traité ce cas.

□

Corollaire 2.3.17 (Transitivité de la coercion). *Si* $\Gamma \vdash_{CCI} c_1 : S \rhd_\bullet T :$ *et* $\Gamma \vdash_{CCI} c_2 : T \rhd_\bullet U :$ *alors il existe* $c \equiv c_2 \circ c_1$ *tel que* $\Gamma \vdash_{CCI} c : S \rhd_\bullet U :$.

On peut maintenant énoncer le lemme de symétrie :

Lemme 2.3.18 (Symétrie de la coercion). *S'il existe c tel que* $\Gamma \vdash_{CCI} c : A \rhd_\bullet B :$ *alors* $\exists! c^{-1}, \Gamma \vdash_{CCI} c^{-1} : B \rhd_\bullet A :$ *et* $c^{-1} \circ c \equiv \bullet \equiv c \circ c^{-1}$.

Démonstration. On sait que le jugement $\rhd_\bullet$ est symétrique, c'est à dire qu'on a l'existence des inverses. On utilise la transitivité pour montrer le reste du lemme. Si $\Gamma \vdash_{CCI} c : A \rhd_\bullet B :$ et $\Gamma \vdash_{CCI} c^{-1} : B \rhd_\bullet A :$ alors il existe des coercions f et f^{-1} telles que $\Gamma \vdash_{CCI} f \equiv c^{-1} \circ c : A \rhd_\bullet A :$ et $\Gamma \vdash_{CCI} f^{-1} \equiv c \circ c^{-1} : B \rhd_\bullet B :$. Par réflexivité (lemme 2.3.5) et unicité des coercions, $f \equiv \bullet$ et $f^{-1} \equiv \bullet$. □

Lemme 2.3.19 (Stabilité par affaiblissement). *Si* $\Gamma \vdash_\bullet t : T$ *alors pour tout* $\Delta, \rho : \Delta \rhd_\bullet \Gamma, \Delta \vdash_\bullet t : T'$ *et il existe* $\alpha, \Delta \vdash_{CCI} \alpha : T' \rhd_\bullet T :$ *avec* $[\![t]\!]_\Gamma[\rho] \equiv \alpha[[\![t]\!]_\Delta]$.

Démonstration. La première partie du lemme se déduit par applications répétées du lemme de restriction 2.2.13.

On va utiliser à plusieurs reprises le résultat suivant : Si une prémisse du jugement sur lequel on fait la récurrence est de la forme $\Gamma \vdash_\bullet T : s$, alors on peut appliquer l'hypothèse d'induction pour obtenir $\Delta \vdash_\bullet T : s'$ avec α tel que $\Delta \vdash_{CCI} \alpha : s \rhd_\bullet s' :$ et $[\![T]\!]_\Gamma[\rho] \equiv \alpha[[\![T]\!]_\Delta]$. Or comme s est une sorte, on a $s' = s$ et $\alpha = \bullet$. On a donc $[\![T]\!]_\Gamma[\rho] \equiv [\![T]\!]_\Delta$.

Par induction sur la dérivation de t.

– Var : On a :

$$\frac{\vdash_\bullet \simeq \mathbf{wf} G \qquad y : T \in \Gamma}{\Gamma \vdash_\bullet y : T}$$

Par définition de la coercion de contextes, il existe $c : (T' \rhd_\bullet T) \in \rho$. Soit $\alpha = c$, on a bien : $[\![y]\!]_\Gamma[\rho] = c[y] = \alpha[[\![y]\!]_\Delta]$.

– App : On a :

$$\frac{\Gamma \vdash_\bullet f : F \quad \mu_\bullet(F) = \Pi y : A.B \qquad \Gamma \vdash_\bullet e : E \quad \Gamma \vdash_\bullet E, A : s \qquad E \rhd_\bullet A}{\Gamma \vdash_\bullet (f\ e) : B[e/y]}$$

et donc par induction, $\Delta \vdash_\bullet f : F'$ avec $\Delta \vdash_{CCI} \alpha_f : F' \rhd_\bullet F$: et $\Delta \vdash_\bullet e : E'$ avec $\Delta \vdash_{CCI} \alpha_e : E' \rhd_\bullet E$:.
Par définition de l'interprétation,

$$\begin{array}{rcl} [\![(f\ e)]\!]_\Gamma & = & (((\pi_F[[\![f]\!]_\Gamma])\ (c_e[[\![e]\!]_\Gamma])) \\ \text{où} & & \\ \pi_F & = & \mathbf{coerce}_\Gamma\ F\ \Pi y : A.B \\ c_e & = & \mathbf{coerce}_\Gamma\ E\ A. \end{array}$$

De même :

$$\begin{array}{rcl} [\![f\ e]\!]_\Delta & = & (\pi_{F'}[[\![f']\!]_\Delta])\ (c_{e'}[[\![e']\!]_\Delta]) \\ \text{où} & & \\ \pi_{F'} & = & \mathbf{coerce}_\Delta\ F'\ \Pi y : A'.B' \\ c_{e'} & = & \mathbf{coerce}_\Delta\ E'\ A' \end{array}$$

On a $\Gamma \vdash_{CCI} \pi_F : F \rhd_\bullet \Pi y : A.B$:, donc par le lemme 2.3.15 on obtient $\Delta \vdash_{CCI} \pi_F[\rho] : F \rhd_\bullet \Pi y : A.B$:. On peut appliquer ce lemme puisque $[\![\Pi y : A.B]\!]_\Delta \equiv [\![\Pi y : A.B]\!]_\Gamma[\rho]$ par induction (α est nécessairement l'identité).
On a donc les coercions suivantes dans l'environnement Δ :

$$\begin{array}{ccc} F & \xrightarrow[\pi_F[\rho]]{} & \Pi y : A.B \\ {\scriptstyle \alpha_f}\uparrow & & \uparrow{\scriptstyle c_f} \\ F' & \xrightarrow[\pi_{F'}]{} & \Pi y : A'.B' \end{array}$$

Par symétrie et transitivité de la coercion, on en déduit qu'il existe c_f, $\Delta \vdash_{CCI} c_f : \Pi y : A'.B' \rhd_\bullet \Pi y : A.B$:. La coercion c_f est nécessairement de la forme : $\lambda y : [\![A]\!]_\Delta.c_2[\bullet\ c_1[y]]$ où $\Delta \vdash_{CCI} c_1 : A \rhd_\bullet A'$: et $\Delta, x : A \vdash_{CCI} c_2 : B' \rhd_\bullet B$:.
Par le lemme 2.3.15 on a $\Delta \vdash_{CCI} c_e[\rho] : E \rhd_\bullet A$:, de même que précédemment on obtient la condition nécessaire par induction sur la dérivation de $\Gamma \vdash_\bullet A : s$.
On a donc les coercions suivantes dans l'environnement Δ :

$$\begin{array}{ccc} E & \xrightarrow[c_e[\rho]]{} & A \\ {\scriptstyle \alpha_e}\uparrow & & \downarrow{\scriptstyle c_1} \\ E' & \xrightarrow{c_{e'}} & A' \end{array}$$

Par transitivité, il existe donc une coercion $\Delta \vdash_{CCI} c_{E',A} \equiv c_e[\rho] \circ \alpha_e : E' \rhd_\bullet A$:. Par le lemme 2.3.15 en utilisant la coercion de contexte $\bullet, \ldots, c_{E',A} : (\Delta, x : E') \rhd_\bullet \Delta, x : A$ et la dérivation de c_2 on a donc : $\Delta, x : E' \vdash_{CCI} c_2' : B' \rhd_\bullet B$: avec $c_2' \equiv c_2[(c_e[\rho])[\alpha_e[x]]/x]$.
Par le lemme 2.3.11 avec $\Delta \vdash_{CCI} e : E'$ et la dérivation de c_2' on obtient : $\Delta \vdash_{CCI} c_2'' : B'[e/x] \rhd_\bullet B[e/x]$: avec

$$c_2'' \equiv c_2'[[\![e]\!]_\Delta/x] \equiv c_2[(c_e[\rho])[\alpha_e[[\![e]\!]_\Delta]]/x]$$

Soit $\alpha = c_2''$, on peut vérifier :

$$
\begin{array}{ll}
 & \alpha[[\![f\ e]\!]_\Delta] \\
 & \quad \textbf{\{ Définition de l'interprétation \}} \\
\hat{=} & \alpha[(\pi_{F'}[[\![f]\!]_\Delta])[c_{e'}[[\![e]\!]_\Delta]]] \\
 & \quad \textbf{\{ Définitions de } \alpha \textbf{ et } c_{e'} \textbf{ \}} \\
\equiv & c_2[c_e[\rho]\ [(\alpha_e[[\![e]\!]_\Delta])]/x]\ [((\pi_{F'}[[\![f]\!]_\Delta])\ [(c_1 \circ c_e[\rho] \circ \alpha_e)[[\![e]\!]_\Delta]])] \\
 & \quad \textbf{\{ Définition de la composition \}} \\
= & c_2[c_e[\rho]\ [(\alpha_e[[\![e]\!]_\Delta])]/x]\ [((\pi_{F'}[[\![f]\!]_\Delta])\ [c_1[c_e[\rho][\alpha_e[[\![e]\!]_\Delta]]]])] \\
 & \quad \textbf{\{ Définition de } c_f \textbf{ (} = \lambda x.c_2[\bullet\ c_1[x]]\textbf{) \}} \\
= & c_f[(\pi_{F'}[[\![f]\!]_\Delta])\ [c_e[\rho]\ [\alpha_e[[\![e]\!]_\Delta]]]] \\
 & \quad \textbf{\{ Commutation du diagramme pour la fonction \}} \\
\equiv & (\pi_F[\rho][\alpha_f[[\![f]\!]_\Delta]]\ [c_e[\rho][\alpha_e\ [\![e]\!]_\Delta]]) \\
 & \quad \textbf{\{ Définitions de } \alpha_f \textbf{ et } \alpha_e \textbf{ \}} \\
\equiv & (\pi_F[\rho][[[\![f]\!]_\Gamma[\rho]])\ [c_e[\rho][[[\![e]\!]_\Gamma[\rho]]] \\
 & \quad \textbf{\{ Définition de l'interprétation \}} \\
\hat{=} & [\![f\ e]\!]_\Gamma[\rho]
\end{array}
$$

– Prod : On a

$$\frac{\Gamma \vdash_\bullet T : s_1 \qquad \Gamma, x : T \vdash_\bullet U : s_2}{\Gamma \vdash_\bullet \Pi x : T.U : s_3}\ (s_1, s_2, s_3) \in \mathcal{R}$$

Par induction, $\Delta \vdash_\bullet T : s_1$ et $\Delta, x : T \vdash_\bullet U : s_2$ (en utilisant une coercion identité ajoutée à ρ). On peut donc appliquer Prod pour obtenir le résultat désiré avec $\alpha = \bullet$. De même pour Sum, Subset.

– Abs : On a

$$\frac{\Gamma \vdash_\bullet \Pi x : T.U : s \qquad \Gamma, x : T \vdash_\bullet M : U}{\Gamma \vdash_\bullet \lambda x : T.M : \Pi x : T.U}$$

Par induction, $\Delta \vdash_\bullet \Pi x : T.U : s$ et il existe U', $\Delta, x : T \vdash_\bullet M : U'$ avec une coercion $\Delta, x : T \vdash_{CCI} \alpha' : U' \rhd_\bullet U$: telle que $[\![M]\!]_{\Gamma,x:T}[\rho, \bullet] \equiv \alpha'[[\![M]\!]_{\Delta,x:T}]$.
Soit α la coercion : $\Delta \vdash_{CCI} \lambda x : [\![T]\!]_\Delta.\alpha'[\bullet\ x] : \Pi x : T.U' \rhd_\bullet \Pi x : T.U$:. On a bien :

$$
\begin{array}{ll}
 & \alpha[[\![\lambda x : T.M]\!]_\Delta] \\
 & \quad \textbf{\{ Définition de l'interprétation \}} \\
\hat{=} & \alpha[\lambda x : [\![T]\!]_\Delta.[\![M]\!]_{\Delta,x:T}] \\
 & \quad \textbf{\{ Définition de la coercion } \alpha \textbf{ \}} \\
\hat{=} & \lambda x : [\![T]\!]_\Delta.\alpha'[[\![M]\!]_{\Delta,x:T}[x/x]] \\
 & \quad \textbf{\{ Définition de le coercion } \alpha' \textbf{ \}} \\
\equiv & \lambda x : [\![T]\!]_\Delta.[\![M]\!]_{\Gamma,x:T}[\rho] \\
 & \quad \textbf{\{ Hypothèse d'induction, } [\![T]\!]_\Delta = [\![T]\!]_\Gamma[\rho] \textbf{ \}} \\
\equiv & \lambda x : [\![T]\!]_\Gamma[\rho].[\![M]\!]_{\Gamma,x:T}[\rho] \\
 & \quad \textbf{\{ Définition de la substitution \}} \\
= & (\lambda x : [\![T]\!]_\Gamma.[\![M]\!]_{\Gamma,x:T})[\rho]
\end{array}
$$

– Pair : On a :

$$\frac{\Gamma \vdash_\bullet \Sigma x : T.U : s \qquad \begin{array}{c}\Gamma \vdash_\bullet t : T' \quad \Gamma \vdash_\bullet T' \rhd_\bullet T : s \\ \Gamma \vdash_\bullet u : U' \quad \Gamma \vdash_\bullet U' \rhd_\bullet U[t/x] : s\end{array}}{\Gamma \vdash_\bullet (t,u)_{\Sigma x:T.U} : \Sigma x : T.U}$$

On a donc les coercions : $\Gamma \vdash_{CCI} \alpha_t : T' \rhd_\bullet T :$ et $\Gamma \vdash_{CCI} \alpha_u : U' \rhd_\bullet U[t/x] :$. On peut les faire passer dans l'environnement Δ par le lemme 2.3.15 (la condition est vérifiée en utilisant la dérivation de $\Gamma \vdash_\bullet \Sigma x : T.U : s$). On obtient :

$$\Delta \vdash_{CCI} \alpha_t[\rho] : T' \rhd_\bullet T : \wedge \Delta \vdash_{CCI} \alpha_u[\rho] : U' \rhd_\bullet U[t/x] :$$

Par induction :

$$\frac{\Delta \vdash_\bullet t : T'' \quad T'' \rhd_\bullet T \qquad \begin{array}{c}\Delta \vdash_\bullet u : U'' \\ \Delta \vdash_\bullet \Sigma x : T.U : s\end{array} \qquad \Delta \vdash_\bullet U'' : s \quad U'' \rhd_\bullet U[t/x]}{\Delta \vdash_\bullet (t,u)_{\Sigma x:T.U} : \Sigma x : T.U}$$

Avec les coercions :

$$\Delta \vdash_{CCI} \beta_t : T'' \rhd_\bullet T' :, \beta_t[[\![t]\!]_\Delta] \equiv [\![t]\!]_\Gamma[\rho]$$
$$\Delta \vdash_{CCI} \beta_u : U'' \rhd_\bullet U' :, \beta_u[[\![u]\!]_\Delta] \equiv [\![u]\!]_\Gamma[\rho]$$

Par transitivité de la coercion on peut construire les coercions

$$\Delta \vdash_{CCI} \chi_t \equiv \alpha_t[\rho] \circ \beta_t : T'' \rhd_\bullet T : \wedge \Delta \vdash_{CCI} \chi_u = \alpha_u[\rho] \circ \beta_u : U'' \rhd_\bullet U :$$

utilisées dans le jugement précédent.
Par réflexivité de la coercion, on a $\Delta \vdash_{CCI} \alpha \equiv \bullet : \Sigma x : T.U \rhd_\bullet \Sigma x : T.U :$.
On peut vérifier :

$$\begin{array}{rl} & \alpha[[\![(t,u)_{\Sigma x:T.U}]\!]_\Delta] \\ & \quad \textbf{\{ Définition de l'interprétation \}} \\ \hat{=} & (\chi_t[[\![t]\!]_\Delta], \chi_u[[\![u]\!]_\Delta])_{[\![\Sigma x:T.U]\!]_\Delta} \\ & \quad \textbf{\{ Définition des coercions \}} \\ \hat{=} & (\alpha_t[\rho][\beta_t[[\![t]\!]_\Delta]], \alpha_u[\rho][\beta_u[[\![u]\!]_\Delta]])_{[\![\Sigma x:T.U]\!]_\Delta} \\ & \quad \textbf{\{ Hypothèses sur les coercions \}} \\ \hat{=} & (\alpha_t[\rho][[\![t]\!]_\Gamma[\rho]], \alpha_u[\rho][[\![u]\!]_\Gamma[\rho]])_{[\![\Sigma x:T.U]\!]_\Delta} \\ & \quad \textbf{\{ Définition de l'interprétation \}} \\ \hat{=} & [\![(t,u)_{\Sigma x:T.U}]\!]_\Gamma[\rho] \end{array}$$

– Pi-1 :

$$\frac{\Gamma \vdash_\bullet t : S \qquad \mu_\bullet(S) = \Sigma x : T.U}{\Gamma \vdash_\bullet \pi_1\ t : T}$$

On nomme μ_S la coercion de S à $\Sigma x : T.U$. Par le lemme 2.3.15 on obtient : $\Delta \vdash_{CCI} \mu_S[\rho] : S \rhd_\bullet \Sigma x : T.U :$. Par induction, il existe α_t, $\Delta \vdash_{CCI} \alpha_t : S' \rhd_\bullet S :$ et $\alpha_t[[\![t]\!]_\Delta] = [\![t]\!]_\Gamma[\rho]$. Par transitivité de la coercion, on a $\Delta \vdash_{CCI} \alpha_{S'} \equiv \mu_S[\rho] \circ \alpha_t : S' \rhd_\bullet \Sigma x : T.U :$. On en déduit qu'il existe T', U' tels que $\mu_\bullet(S') = \Sigma x : T'.U'$ et qu'il existe une coercion β telle que

$$\Delta \vdash_{CCI} \beta = (c_1[\pi_1\ \bullet], c_2[\pi_2\ \bullet][\pi_1\ \bullet/x])_{\Sigma x:T.U} : \Sigma x : T'.U' \rhd_\bullet \Sigma x : T.U :$$

avec $\beta \circ \mu_{S'} \equiv \mu_S[\rho] \circ \alpha_t$.
Soit $\alpha = c_1$, on a bien $\Delta \vdash_{CCI} \alpha : T' \rhd_\bullet T :$.

On a :

$$
\begin{array}{rl}
 & \alpha[[\![\pi_1\ t]\!]_\Delta] \\
 & \quad \textbf{\{ Définition de l'interprétation \}} \\
\hat{=} & c_1[\pi_1\ \mu_{S'}[[\![t]\!]_\Delta]] \\
 & \quad \{\ \pi_1\ \beta = c_1[\pi_1\ \bullet]\ \} \\
= & (\pi_1\ \beta)[\mu_{S'}[[\![t]\!]_\Delta]] \\
 & \quad \textbf{\{ Définition de la substitution dans les contextes \}} \\
= & \pi_1\ (\beta[\mu_{S'}[[\![t]\!]_\Delta]]) \\
 & \quad \textbf{\{ Transitivité \}} \\
\equiv & \pi_1\ \mu_S[\rho][\alpha_t[[\![t]\!]_\Delta]] \\
 & \quad \textbf{\{ Hypothèse d'induction \}} \\
\equiv & \pi_1\ \mu_S[\rho][[\![t]\!]_\Gamma[\rho]] \\
 & \quad \textbf{\{ Définition de l'interprétation \}} \\
\hat{=} & [[\![\pi_1\ t]\!]_\Gamma[\rho]
\end{array}
$$

– Pi-2 :

$$\frac{\Gamma \vdash_\bullet t : S \qquad \mu_\bullet(S) = \Sigma x : T.U}{\Gamma \vdash_\bullet \pi_2\ t : U[\pi_1\ t/x]}$$

On nomme μ_S la coercion de S à $\Sigma x : T.U$. Par le lemme 2.3.15 on obtient : $\Delta \vdash_{CCI} \mu_S[\rho] : S \rhd_\bullet \Sigma x : T.U :.$
On peut peut appliquer l'hypothèse d'induction sur $\Gamma \vdash_\bullet t : S$ pour dériver :

$$\frac{\Delta \vdash_\bullet t : S' \qquad \mu_\bullet(S') = \Sigma x : T'.U'}{\Delta \vdash_\bullet \pi_1\ t : T'}$$

Par induction, il existe α_t, $\Delta \vdash_{CCI} \alpha_t : S' \rhd_\bullet S :$ et $\alpha_t[[\![t]\!]_\Delta] = [[\![t]\!]_\Gamma[\rho]$. Par transitivité de la coercion, on sait qu'il existe une coercion de S' à $\Sigma x : T.U$. On en déduit qu'il existe T', U' tels que $\mu_\bullet(S') = \Sigma x : T'.U'$ et qu'il existe une coercion β telle que

$$\Delta \vdash_{CCI} \beta = (c_1[\pi_1\ \bullet], c_2[\pi_2\ \bullet][\pi_1\ \bullet/x])_{\Sigma x:T.U} : \Sigma x : T'.U' \rhd_\bullet \Sigma x : T.U :$$

avec $\beta \circ \mu_{S'} \equiv \mu_S[\rho] \circ \alpha_t$.
On a donc les coercions suivantes dans l'environnement Δ :

$$\begin{array}{ccc}
S & \xrightarrow{\quad\mu_S[\rho]\quad} & \Sigma x : T.U \\
{\scriptstyle \alpha_t}\uparrow & & \uparrow{\scriptstyle \beta} \\
S' & \xrightarrow{\quad\mu_{S'}\quad} & \Sigma x : T'.U'
\end{array}$$

Par substitution pour les coercions, avec les dérivation de $\pi_1\ t$ et de c_2 on obtient : $\Delta \vdash_{CCI} c_2' \equiv c_2[[\![\pi_1\ t]\!]_\Delta/x] : U'[\pi_1\ t/x] \rhd_\bullet U[\pi_1\ t/x] :.$

Soit $\alpha = c_2'$. On a :

$$
\begin{array}{rl}
 & \alpha[[\![\pi_2\ t]\!]_\Delta] \\
 & \quad \{\ \textbf{Définition de } c_2'\ \} \\
\equiv & c_2[[\![\pi_1\ t]\!]_\Delta/x][[\![\pi_2\ t]\!]_\Delta] \\
 & \quad \{\ \textbf{Définition de l'interprétation}\ \} \\
\hat{=} & c_2[\pi_1\ \mu_{S'}[[\![t]\!]_\Delta]/x][[\![\pi_2\ t]\!]_\Delta] \\
 & \quad \{\ \textbf{Définition de l'interprétation}\ \} \\
\hat{=} & c_2[\pi_1\ \mu_{S'}[[\![t]\!]_\Delta]/x][\pi_2\ \mu_{S'}[[\![t]\!]_\Delta]] \\
 & \quad \{\ \bullet \notin [[\![t]\!]_\Delta \wedge x \notin \mathcal{FV}(t)\ \} \\
\hat{=} & c_2[\pi_2\ \mu_{S'}[[\![t]\!]_\Delta]][\pi_1\ \mu_{S'}[[\![t]\!]_\Delta]/x] \\
 & \quad \{\ \pi_2\ \beta = c_2[\pi_2\ \bullet][\pi_1\ \bullet/x]\ \} \\
= & (\pi_2\ \beta)[\mu_{S'}[[\![t]\!]_\Delta]] \\
 & \quad \{\ \textbf{Définition de la substitution dans les contextes}\ \} \\
= & \pi_2\ (\beta[\mu_{S'}[[\![t]\!]_\Delta]]) \\
 & \quad \{\ \textbf{Transitivité}\ \} \\
\equiv & \pi_2\ \mu_S[\rho][\alpha_t[[\![t]\!]_\Delta]] \\
 & \quad \{\ \textbf{Hypothèse d'induction}\ \} \\
\equiv & \pi_2\ \mu_S[\rho][[[\![t]\!]_\Gamma[\rho]] \\
 & \quad \{\ \textbf{Définition de l'interprétation}\ \} \\
\hat{=} & [[\![\pi_2\ t]\!]_\Gamma[\rho]
\end{array}
$$

□

On a besoin d'une spécialisation de ce lemme lorsque les types sont équivalents :

Lemme 2.3.20 (Equivalence et interprétation). *Si* $\Gamma \vdash_\bullet U, U' : s$ *et* $U \equiv_{\beta\pi} U'$ *alors :*
- *si* $\Gamma, x : U, \Delta \vdash_\bullet t : T$, *alors* $\Gamma, x : U', \Delta \vdash_\bullet t : T'$ *avec* $T \equiv_{\beta\pi} T'$ *et* $[[\![t]\!]_{\Gamma,x:U,\Delta} \equiv [[\![t]\!]_{\Gamma,x:U',\Delta}$;
- *si* $\Gamma, x : U, \Delta \vdash_\bullet T \rhd_\bullet T' :$ *alors* $\Gamma, x : U', \Delta \vdash_\bullet T \rhd_\bullet T' :$.

Démonstration. Pour la première partie, la preuve suit le même schéma que la précédente. Il suffit de vérifier que l'on a toujours $\alpha \equiv \bullet$. Par exemple, dans le cas de l'application, on a $\alpha_f \equiv \alpha_e \equiv \bullet$. On en déduit que $c_f \equiv \bullet$ et $c_1 \equiv c_2 \equiv \bullet$. Clairement $c_2'' \equiv \bullet$.

Pour la seconde partie, c'est direct par induction en utilisant la première partie. □

On peut combiner les lemmes de substitution et affaiblissement en un lemme puissant.

Lemme 2.3.21 (Substitution et coercion). *Si* $\Gamma \vdash_\bullet u : U$ *et* $\Gamma \vdash_{CCI} c : U \rhd_\bullet V :$ *alors*

$$
\Gamma, x : V, \Delta \vdash_\bullet t : T \Rightarrow \begin{array}{l} \Gamma, \Delta[u/x] \vdash_\bullet t[u/x] : T' \\ \exists \alpha, \Gamma, \Delta[u/x] \vdash_{CCI} \alpha : T' \rhd_\bullet T[u/x] :, \\ \alpha[[\![t[u/x]]\!]_{\Gamma,\Delta[u/x]}] \equiv [[\![t]\!]_{\Gamma,x:V,\Delta}[c[[\![u]\!]_\Gamma]/x] \end{array}
$$

Démonstration. Si $\Gamma, x : V, \Delta \vdash_\bullet t : T$, alors soit ρ la coercion de contextes $\bullet, \ldots, c, \bullet, \ldots : (\Gamma, x : U, \Delta) \vdash_{CCI} (\Gamma, x : V, \Delta)$. Par le lemme 2.3.19, on a $\Gamma, x : U, \Delta \vdash_\bullet t : T'$ et il existe α,

$$
\Gamma, x : U, \Delta \vdash_{CCI} \alpha : T' \rhd_\bullet T : \wedge [[\![t]\!]_{\Gamma,x:V,\Delta}[\rho] \equiv \alpha[[[\![t]\!]_{\Gamma,x:U,\Delta}]
$$

Par substitutivité de la coercion, il existe $\alpha' \equiv \alpha[[[\![u]\!]_\Gamma/x]$ telle que $\Gamma, \Delta[u/x] \vdash_{CCI} \alpha' : T'[u/x] \rhd_\bullet T[u/x] :$.

On a donc, par substitutivité de l'équivalence :

$$[\![t]\!]_{\Gamma,x:V,\Delta}[c[x]/x][[\![u]\!]_\Gamma/x] \equiv \alpha'[[\![t]\!]_{\Gamma,x:U,\Delta}][[\![u]\!]_\Gamma/x]$$

Comme $x \notin \mathcal{FV}(\alpha')$, on a $\alpha'[[\![t]\!]_{\Gamma,x:U,\Delta}][[\![u]\!]_\Gamma/x] = \alpha'[[\![t]\!]_{\Gamma,x:U,\Delta}[[\![u]\!]_\Gamma/x]]$.
Soit, par application du lemme de stabilité par substitution :

$$[\![t]\!]_{\Gamma,x:V,\Delta}[c[[\![u]\!]_\Gamma]/x] \equiv \alpha'[[\![t[u/x]]\!]_{\Gamma,\Delta[u/x]}]$$

On a donc bien construit la coercion α' recherchée. □

Son corrolaire nous intéresse particulièrement pour la preuve de correction :

Corollaire 2.3.22 (Substitution dans un type et interprétation). *Si* $\Gamma \vdash_\bullet u : U$, $\Gamma \vdash_{CCI} c : U \rhd_\bullet V$: *et* $\Gamma, x : V \vdash_\bullet T : s$ *alors* $[\![T[u/x]]\!]_\Gamma \equiv [\![T]\!]_{\Gamma,x:V}[c[[\![u]\!]_\Gamma]/x]$.

On va maintenant s'attacher à montrer que l'équivalence du système algorithmique est préservée par interprétation. Pour cela, on défini une nouvelle relation qui contient à la fois la coercion et l'équivalence. On définit l'équivalence typée figure 2.12.

On considère sa clôture réflexive, symétrique et transitive.

On a la propriété suivante :

Définition 2.3.23 (Equivalence typée). *Si* $\Gamma \vdash_\bullet t : T$, $\Gamma \vdash_\bullet u : U$, *alors* $\Gamma \vdash_\bullet t : T \equiv_{\beta\pi} u : U$ *si et seulement si* $t \equiv_{\beta\pi} u$ *et* $U \rhd_\bullet T$.

Démonstration. De gauche à droite, c'est trivial par induction. De droite à gauche : il suffit de considérer la relation $\to_{\beta\rho}$. En effet, si $t \equiv_{\beta\pi} u$ alors il existe v tel que $t \twoheadrightarrow_{\beta\rho} v$ et $u \twoheadrightarrow_{\beta\rho} v$ par confluence de la réduction. Pour tout Γ, t, T tels que $\Gamma \vdash_\bullet t : T$ on a par préservation du typage $\Gamma \vdash_\bullet t' : T'$ avec $T' \rhd_\bullet T$ pour tout t' tel que $t \to_{\beta\rho} t'$. Il suffit alors de montrer que $\Gamma \vdash_\bullet t : T \equiv_{\beta\pi} t' : T'$. Par applications répétées de ce résultat et en utilisant la transitivité de l'équivalence typée, on obtient $\Gamma \vdash_\bullet t : T \equiv_{\beta\pi} v : V$ et $\Gamma \vdash_\bullet u : U \equiv_{\beta\pi} v : V$. On applique enfin symétrie et transitivité pour obtenir le résultat $\Gamma \vdash_\bullet t : T \equiv_{\beta\pi} u : U$.

On va donc montrer que pour tout Γ, t, T, $\Gamma \vdash_\bullet t : T$, pour tout u tel que $t \to_{\beta\rho} u$ et $\Gamma \vdash_\bullet u : U$ on a $\Gamma \vdash_\bullet t : T \equiv_{\beta\pi} u : U$. On a déjà par réduction du sujet que $U \rhd_\bullet T$. Par induction sur la dérivation de typage de t.

- $(\lambda x : X.v)\ e \to_\beta v[e/x]$:. C'est la première règle, direct.
- π_1, π_2 : De même.
- $f\ e \to f'\ e$: Par inversion de $\Gamma \vdash_\bullet f\ e : T$ on a $\Gamma \vdash_\bullet f : F$, $\mu_\bullet(F) = \Pi x : A.B$ où $T = B[e/x]$, $\Gamma \vdash_\bullet e : E$ et $\Gamma \vdash_\bullet E \rhd_\bullet A$:. Par induction on a donc $\Gamma \vdash_\bullet f : F \equiv_{\beta\pi} f' : F'$ où $\Gamma \vdash_\bullet F' \rhd_\bullet F$:. Par le lemme sur la coercion et $\mu_\bullet()$ on a $\mu_\bullet(F') = \Pi x : A'.B'$ et $\Gamma \vdash_\bullet \Pi x : A'.B' \rhd_\bullet \Pi x : A.B$:. On a donc $\Gamma \vdash_\bullet A \rhd_\bullet A'$: et par transitivité de la coercion $\Gamma \vdash_\bullet E \rhd_\bullet A'$:. On peut donc appliquer la règle App-≡. Pour obtenir :

$$\frac{\begin{array}{c}\Gamma \vdash_\bullet f : F \equiv_{\beta\pi} f' : F' \\ \mu_\bullet(F) = \Pi x : A.B \quad \mu_\bullet(F') = \Pi x : A'.B' \qquad \Gamma \vdash_\bullet e : E \equiv_{\beta\pi} e : E \quad \Gamma \vdash_\bullet E \rhd_\bullet A : \qquad \Gamma \vdash_\bullet E \rhd_\bullet A' :\end{array}}{\Gamma \vdash_\bullet f\ e : B[e/x] \equiv_{\beta\pi} f'\ e : B'[e/x]}$$

- $f\ e \to f\ e'$: Par App-≡ on obtient $\Gamma \vdash_\bullet f\ e : B[e/x] \equiv_{\beta\pi} f\ e' : B[e'/x]$.
- $\lambda x : X.v \to \lambda x : X'.v$: Direct par LambdaTeq.
- $\lambda x : X.v \to \lambda x : X.v'$: Direct par LambdaTeq. On a $\Gamma, x : X \vdash_\bullet v : Y \equiv_{\beta\pi} v' : Y'$ par induction sur la prémisse $\Gamma, x : X \vdash_\bullet v : Y$.

$$\frac{\Gamma \vdash_\bullet X : s_1 \equiv_{\beta\pi} X : s_1 \qquad \Gamma, x : X \vdash_\bullet v : Y \equiv_{\beta\pi} v' : Y'}{\Gamma \vdash_\bullet \lambda x : X.v : \Pi x : X.Y \equiv_{\beta\pi} \lambda x : X.v' : \Pi x : X.Y'}$$

– $(e_1, e_2)_{\Sigma x:A.B} \to (e'_1, e_2)_{\Sigma x:A.B}$: Par application de Pair-≡. On a $\Gamma \vdash_\bullet e_1 : A' \equiv_{\beta\pi} e'_1 : C'$ par induction sur la prémisse $\Gamma \vdash_\bullet e_1 : A'$. On a $\Gamma \vdash_\bullet C' \rhd_\bullet A'$: en utilisant l'autre sens de l'équivalence qu'on cherche à prouver. On en déduit $\Gamma \vdash_\bullet C' \rhd_\bullet A$: par transitivité de la coercion avec $\Gamma \vdash_\bullet A' \rhd_\bullet A$:. D'autre part, comme $e_1 \equiv_{\beta\pi} e'_1$ et $B \equiv_{\beta\pi} B'$ on a $B[e_1/x] \equiv_{\beta\pi} B'[e'_1/x]$. Il existe donc une coercion de B' à $B[e_1/x]$.

$$\frac{\Gamma \vdash_\bullet e_1 : A' \equiv_{\beta\pi} e'_1 : C' \qquad \Gamma, x : A' \vdash_\bullet e_2 : B' \equiv_{\beta\pi} e_2 : B' \qquad \Gamma \vdash_\bullet A' \rhd_\bullet A : \qquad \Gamma \vdash_\bullet C' \rhd_\bullet C : \qquad \Gamma \vdash_\bullet B' \rhd_\bullet B[e_1/x] : \qquad \Gamma \vdash_\bullet B' \rhd_\bullet B[e'_1/x] :}{\Gamma \vdash_\bullet (e_1, e_2)_{\Sigma x:A.B} : \Sigma x : A.B \equiv_{\beta\pi} (e'_1, e_2)_{\Sigma x:A.B} : \Sigma x : A.B}$$

De façon équivalente pour la réduction dans la deuxième composante.

– Direct par induction pour les types produits, sommes et sous-ensemble.

□

Théorème 2.3.24 (Conservation de l'équivalence). *Si* $\Gamma \vdash_\bullet t : T$, $\Gamma \vdash_\bullet u : U$, $t \equiv_{\beta\pi} u$ *et* $T \rhd_\bullet U$ *alors il existe* α, $\alpha[[\![t]\!]_\Gamma] \equiv [\![u]\!]_\Gamma$ *où* $\Gamma \vdash_{CCI} \alpha : T \rhd_\bullet U$:.

Démonstration. Par induction sur la dérivation de $\Gamma \vdash_\bullet t : T \equiv_{\beta\pi} u : U$.

– Transitivité : On a $\Gamma \vdash_\bullet t : T \equiv_{\beta\pi} v : V$ et $\Gamma \vdash_\bullet v : V \equiv_{\beta\pi} t : T$, donc par induction, il existe α, β telles que : $\alpha[[\![t]\!]_\Gamma] \equiv [\![v]\!]_\Gamma$ et $\beta[[\![v]\!]_\Gamma] \equiv [\![u]\!]_\Gamma$ où $\Gamma \vdash_{CCI} \alpha : T \rhd_\bullet V$: et $\Gamma \vdash_{CCI} \beta : V \rhd_\bullet U$:. Par transitivité de la coercion il existe $c \equiv \beta \circ \alpha$ telle que $\Gamma \vdash_{CCI} c : T \rhd_\bullet U$:. Clairement, $c[[\![t]\!]_\Gamma] \equiv \beta[\alpha[[\![t]\!]_\Gamma]] \equiv \beta[[\![v]\!]_\Gamma] \equiv [\![u]\!]_\Gamma$.

– Symétrie : On a $\Gamma \vdash_\bullet u : U \equiv_{\beta\pi} t : T$. Par induction, il existe α, $\alpha[[\![u]\!]_\Gamma] \equiv [\![t]\!]_\Gamma$ et $\Gamma \vdash_{CCI} \alpha : U \rhd_\bullet T$:. Par symmétrie de la coercion, il existe $c \equiv \alpha^{-1}$, $\Gamma \vdash_{CCI} c : T \rhd_\bullet U$: et $c \circ \alpha \equiv \alpha \circ c \equiv \bullet$. On a donc $c[\alpha[[\![u]\!]_\Gamma]] \equiv [\![u]\!]_\Gamma \equiv \alpha^{-1}[[\![t]\!]_\Gamma]$. Par symétrie de l'équivalence, on obtient le résultat désiré : $\alpha^{-1}[[\![t]\!]_\Gamma \equiv [\![u]\!]_\Gamma$.

– Réflexivité : On a $\Gamma \vdash_\bullet t : T \equiv_{\beta\pi} t : T$. Par réflexivité de la coercion, c'est trivial.

– VarTeq : De même, $\alpha = \bullet$.

– SortTeq : Trivial.

– LambdaTeq : Par induction on a $\Gamma \vdash_{CCI} \alpha = \bullet : s_1 \rhd_\bullet s_1$:, $[\![X]\!]_\Gamma \equiv [\![X']\!]_\Gamma$ et $\Gamma, x : X' \vdash_{CCI} \beta : Y \rhd_\bullet Y'$: avec $\beta[[\![v]\!]_{\Gamma,x:X'}] = [\![v']\!]_{\Gamma,x:X'}$.
On obtient donc le jugement $\Gamma \vdash_{CCI} c = (\lambda x : [\![X']\!]_\Gamma.\beta[\bullet\ \bullet[x]]) : \Pi x : X.Y \rhd_\bullet \Pi x : X'.Y'$: par $\rhd$-Prod.
On a bien :

$$\begin{array}{lll}
 & c[[\![\lambda x : X.v]\!]_\Gamma] & \\
 & \quad \{ \textbf{Définition de l'interprétation} \} & \\
\hat{=} & c[\lambda x : [\![X]\!]_\Gamma.[\![v]\!]_{\Gamma,x:X}] & \\
 & \quad \{ \textbf{Définition de } c \} & \\
\equiv & \lambda x : [\![X']\!]_\Gamma.\beta[(\lambda x : [\![X]\!]_\Gamma.[\![v]\!]_{\Gamma,x:X})\ x] & \\
 & \quad \{ \textbf{Réduction} \} & \\
\to_\beta & \lambda x : [\![X']\!]_\Gamma.\beta[[\![v]\!]_{\Gamma,x:X}] & \\
 & \quad \{ \textbf{Lemme 2.3.20 :} [\![v]\!]_{\Gamma,x:X} \equiv [\![v]\!]_{\Gamma,x:X'} \} & \\
\equiv & \lambda x : [\![X']\!]_\Gamma.\beta[[\![v]\!]_{\Gamma,x:X'}] & \\
 & \quad \{ \textbf{Hypothèse sur } \beta \} & \\
\equiv & \lambda x : [\![X']\!]_\Gamma.[\![v']\!]_{\Gamma,x:X'} & \\
 & \quad \{ \textbf{Définition de l'interprétation} \} & \\
\equiv & [\![\lambda x : X'.v']\!]_\Gamma &
\end{array}$$

– β-$\equiv$: On a $[\![(\lambda x : X.v)\ e]\!]_\Gamma = (\lambda x : [\![X]\!]_\Gamma.[\![v]\!]_{\Gamma,x:X})\ c[[\![e]\!]_\Gamma] \rightarrow_\beta [\![v]\!]_{\Gamma,x:X}[c[[\![e]\!]_\Gamma]/x]$ où $\Gamma \vdash_\bullet e : E$, $\Gamma \vdash_{CCI} c : E \rhd_\bullet X$:. On doit montrer qu'il existe α tel que $[\![v]\!]_{\Gamma,x:X}[c[[\![e]\!]_\Gamma]/x] \equiv \alpha[[\![v[e/x]]\!]]$. Par inversion de $\Gamma \vdash_\bullet (\lambda x : X.v)\ e : T$ on a $\Gamma, x : X \vdash_\bullet v : V$ tel que $V[e/x] = T$.
On applique lemme de substitution et coercion 2.3.21 avec $\Gamma \vdash_\bullet e : E$, $\Gamma \vdash_{CCI} c : E \rhd_\bullet X$: et cette dérivation. On obtient : $\alpha[[\![v[e/x]]\!]_\Gamma \equiv [\![v]\!]_{\Gamma,x:X}[c[[\![e]\!]_\Gamma]/x]$, soit le résultat désiré.

– Π_1-$\equiv$: On a $[\![\pi_1\ (e_1, e_2)_{\Sigma x:T.V}]\!]_\Gamma \hat{=} \pi_1\ [\![(e_1, e_2)_{\Sigma x:T.V}]\!]_\Gamma = \pi_1\ ([\![e_1]\!]_\Gamma, [\![e_2]\!]_\Gamma)_{[\![\Sigma x:T.V]\!]_\Gamma} \rightarrow_{\pi_1} [\![e_1]\!]_\Gamma$. On peut donc prendre $\Gamma \vdash_{CCI} \alpha : T \rhd_\bullet T :\equiv \bullet$.

– Π_2-$\equiv$: On a $[\![\pi_2\ (e_1, e_2)_{\Sigma x:T.V}]\!]_\Gamma \hat{=} \pi_2\ [\![(e_1, e_2)_{\Sigma x:T.V}]\!]_\Gamma = \pi_2\ ([\![e_1]\!]_\Gamma, [\![e_2]\!]_\Gamma)_{[\![\Sigma x:T.V]\!]_\Gamma} \rightarrow_{\pi_2} [\![e_1]\!]_\Gamma$. On peut donc prendre $\Gamma \vdash_{CCI} \alpha : T \rhd_\bullet T :\equiv \bullet$.

– App-$\equiv$: On a

$$\frac{\Gamma \vdash_\bullet M : S \equiv_{\beta\pi} M' : T \qquad \mu_\bullet(S) = \Pi x : A.B \quad \mu_\bullet(T) = \Pi x : C.D \qquad \Gamma \vdash_\bullet N : U \equiv_{\beta\pi} N' : V \quad \Gamma \vdash_\bullet U \rhd_\bullet A : \qquad \Gamma \vdash_\bullet V \rhd_\bullet C :}{\Gamma \vdash_\bullet M\ N : B[N/x] \equiv_{\beta\pi} M'\ N' : D[N'/x]}$$

On a :

$$[\![M\ N]\!]_\Gamma \hat{=} \pi_S[[\![M]\!]_\Gamma]\ c[[\![N]\!]_\Gamma] \text{ et } [\![M'\ N']\!]_\Gamma \hat{=} \pi_T[[\![M']\!]_\Gamma]\ c'[[\![N']\!]_\Gamma]$$

où

$$\begin{aligned} \pi_S &= \mathbf{coerce}_\Gamma\ S\ (\Pi x : A.B) \\ \pi_T &= \mathbf{coerce}_\Gamma\ T\ (\Pi x : C.D) \\ c &= \mathbf{coerce}_\Gamma\ U\ A \\ c' &= \mathbf{coerce}_\Gamma\ V\ C \end{aligned}$$

Par induction, il exite α, β, telles que :

$$\Gamma \vdash_{CCI} \alpha : S \rhd_\bullet T : \text{ et } \alpha[[\![M]\!]_\Gamma] \equiv [\![M']\!]_\Gamma$$

$$\Gamma \vdash_{CCI} \beta : U \rhd_\bullet V : \text{ et } \beta[[\![N]\!]_\Gamma] \equiv [\![N']\!]_\Gamma$$

On a donc les coercions suivantes :

$$\begin{array}{ccc} S & \xrightarrow{\pi_S} & \Pi x : A.B \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\alpha'} \\ T & \xrightarrow{\pi_T} & \Pi x : C.D \end{array} \qquad \begin{array}{ccc} U & \xrightarrow{c} & A \\ {\scriptstyle\beta}\downarrow & & \uparrow{\scriptstyle\beta'} \\ V & \xrightarrow{c'} & C \end{array}$$

La coercion α' est nécessairement de la forme $\lambda x : [\![C]\!]_\Gamma.c_2[\bullet\ c_1[x]]$ où $\Gamma \vdash_{CCI} c_1 = \beta' : C \rhd_\bullet A$: et $\Gamma, x : C \vdash_{CCI} c_2 : B \rhd_\bullet D$:. Par le lemme 2.3.11 avec $\Gamma \vdash_\bullet N' : V$, $\Gamma \vdash_{CCI} c' : V \rhd_\bullet C$: et $\Gamma, x : C \vdash_{CCI} c_2 : B \rhd_\bullet D$: on a : $\Gamma \vdash_{CCI} c'_2 \equiv c_2[c'[[\![N']\!]_\Gamma]/x] : B[N'/x] \rhd_\bullet D[N'/x]$:. Or comme $B[N'/x] \equiv_{\beta\pi} B[N/x]$ on a $\Gamma \vdash_{CCI} c'_2 : B[N/x] \rhd_\bullet D[N'/x]$:. Soit $\alpha = c'_2$.

On a bien :

$$
\begin{array}{rl}
& \alpha[[\![M\ N]\!]_\Gamma] \\
& \quad \{ \textbf{Définition de l'interprétation} \} \\
\hat{=} & \alpha[\pi_S[[\![M]\!]_\Gamma]\ c[[\![N]\!]_\Gamma]] \\
& \quad \{ \textbf{Définition de } \alpha \} \\
\hat{=} & c_2[c'[[\![N']\!]_\Gamma]/x][\pi_S[[\![M]\!]_\Gamma]\ c[[\![N]\!]_\Gamma]] \\
& \quad \{ \textbf{Hypothèse sur } \beta \} \\
\equiv & c_2[c'[\beta[[\![N]\!]_\Gamma]]/x][\pi_S[[\![M]\!]_\Gamma]\ c[[\![N]\!]_\Gamma]] \\
& \quad \{ \textbf{Commutation du diagrame pour l'argument} \} \\
\equiv & c_2[c'[\beta[[\![N]\!]_\Gamma]]/x][\pi_S[[\![M]\!]_\Gamma]\ (\beta'[c'[\beta[[\![N]\!]_\Gamma]]])] \\
& \quad \{ \textbf{Définition de } \alpha' \} \\
\equiv & \alpha'[\pi_S[[\![M]\!]_\Gamma]]\ (c'[\beta[[\![N]\!]_\Gamma]]) \\
& \quad \{ \textbf{Commutation du diagramme pour la fonction} \} \\
\equiv & \pi_T[\alpha[[\![M]\!]_\Gamma]]\ (c'[\beta[[\![N]\!]_\Gamma]]) \\
& \quad \{ \textbf{Hypothèses sur } \alpha \textbf{ et } \beta \} \\
\equiv & \pi_T[[\![M']\!]_\Gamma]\ (c'[[\![N']\!]_\Gamma]) \\
& \quad \{ \textbf{Définition de l'interprétation} \} \\
\hat{=} & [[\![M'\ N']\!]_\Gamma
\end{array}
$$

– Pair-≡ :

$$
\frac{\Gamma \vdash_\bullet e_1 : A' \equiv_{\beta\pi} e'_1 : C' \qquad \Gamma, x : A' \vdash_\bullet e_2 : B' \equiv_{\beta\pi} e'_2 : D' \qquad \Gamma \vdash_\bullet A' \rhd_\bullet A : \quad \Gamma \vdash_\bullet C' \rhd_\bullet C : \qquad \Gamma \vdash_\bullet B' \rhd_\bullet B[e_1/x] : \quad \Gamma \vdash_\bullet D' \rhd_\bullet D[e'_1/x] :}{\Gamma \vdash_\bullet (e_1, e_2)_{\Sigma x:A.B} : \Sigma x : A.B \equiv_{\beta\pi} (e'_1, e'_2)_{\Sigma x:C.D} : \Sigma x : C.D}
$$

On a $[[\![(e_1, e_2)_{\Sigma x:A.B}]\!]_\Gamma = (c_1[[\![e_1]\!]_\Gamma], c_2[[\![e_2]\!]_\Gamma])_{[\![\Sigma x:A.B]\!]_\Gamma}$ et $[[\![(e'_1, e'_2)_{\Sigma x:C.D}]\!]_\Gamma = (c'_1[[\![e'_1]\!]_\Gamma], c'_2[[\![e'_2]\!]_\Gamma])_{[\![\Sigma x:C.D]\!]_\Gamma}$ où :

$$
\begin{array}{rcl}
c_1 & = & \textbf{coerce}_\Gamma\ A'\ A \\
c'_1 & = & \textbf{coerce}_\Gamma\ C'\ C \\
c_2 & = & \textbf{coerce}_\Gamma\ B'\ B[e_1/x] \\
c'_2 & = & \textbf{coerce}_\Gamma\ D'\ D[e'_1/x]
\end{array}
$$

Par induction il existe α, β telles que :

$$\Gamma \vdash_{CCI} \alpha : A' \rhd_\bullet C' : \text{ et } \alpha[[\![e_1]\!]_\Gamma] \equiv [[\![e'_1]\!]_\Gamma$$

$$\Gamma \vdash_{CCI} \beta : B' \rhd_\bullet D' : \text{ et } \beta[[\![e_2]\!]_\Gamma] \equiv [[\![e'_2]\!]_\Gamma$$

On a $\Gamma \vdash_\bullet \Sigma x : A.B \rhd_\bullet \Sigma x : C.D$:, on en déduit qu'il existe la dérivation suivante :

$$
\frac{\Gamma \vdash_{CCI} d_1 : A \rhd_\bullet C : \qquad \Gamma, x : A \vdash_{CCI} d_2 : B \rhd_\bullet D :}{\Gamma \vdash_{CCI} d = (:, \Sigma)_{[\![\Sigma x:C.D]\!]_{d_1[\pi_1\ \bullet]d_2[\pi_2\ \bullet][\pi_1\ \bullet/x]}} x : A.B \rhd_\bullet \Sigma x : C.D :
$$

Par symétrie et transitivité de la coercion on a $d_1 \equiv c'_1 \circ \alpha \circ c_1^{-1} \vdash_{CCI} A : C \rhd_\bullet :$.
Par transitivité, il existe aussi une coercion d'_2 telle que $\Gamma \vdash_{CCI} d_2 : B[e_1/x] \rhd_\bullet D[e'_1/x] :$ et $d_2 \equiv c'_2 \circ \beta \circ c_2^{-1}$. Or par affaiblissement des coercions on a : $\Gamma, x : A' \vdash_{CCI} d_2[c_1[x]/x] : B \rhd_\bullet D :$. On peut appliquer le lemme de substitutivité pour les coercion afin d'obtenir une dérivation de $\Gamma, x : A \vdash_{CCI} d_2[c_1[[\![e_1]\!]_\Gamma]/x] : B[e_1/x] \rhd_\bullet D[e_1/x] :$. Par unicité des coercions, on a $d'_2[c_1[[\![e_1]\!]_\Gamma]/x] \equiv d_2$.

On peut vérifier :

$$
\begin{aligned}
& d[[\![(e_1, e_2)_{\Sigma x:A.B}]\!]_\Gamma] \\
& \quad \{ \textbf{Définition de l'interprétation} \} \\
\hat{=}\ & d[(c_1[[\![e_1]\!]_\Gamma], c_2[[\![e_2]\!]_\Gamma])_{[\![\Sigma x:A.B]\!]_\Gamma}] \\
& \quad \{ \textbf{Définition de } d \} \\
\hat{=}\ & (d_1[c_1[[\![e_1]\!]_\Gamma]], d_2[c_2[[\![e_2]\!]_\Gamma]][c_1[[\![e_1]\!]_\Gamma/x]])_{[\![\Sigma x:C.D]\!]_\Gamma} \\
& \quad \{ \textbf{Définition de } d_1 \} \\
\hat{=}\ & ((c'_1 \circ \alpha \circ c_1^{-1})[c_1[[\![e_1]\!]_\Gamma]], d_2[c_2[[\![e_2]\!]_\Gamma]][c_1[[\![e_1]\!]_\Gamma/x]])_{[\![\Sigma x:C.D]\!]_\Gamma} \\
& \quad \{ c_1^{-1} \circ c_1 \equiv \bullet \} \\
\hat{=}\ & (c'_1[\alpha[[\![e_1]\!]_\Gamma]], d_2[c_2[[\![e_2]\!]_\Gamma]][c_1[[\![e_1]\!]_\Gamma/x]])_{[\![\Sigma x:C.D]\!]_\Gamma} \\
& \quad \{ \textbf{Hypothèse sur } \alpha \} \\
\hat{=}\ & (c'_1[[\![e'_1]\!]_\Gamma], d_2[c_2[[\![e_2]\!]_\Gamma]][c_1[[\![e_1]\!]_\Gamma/x]])_{[\![\Sigma x:C.D]\!]_\Gamma} \\
& \quad \{ \textbf{Définition de la substitution, } x \notin e_2 \} \\
\hat{=}\ & (c'_1[[\![e'_1]\!]_\Gamma], d_2[c_1[[\![e_1]\!]_\Gamma/x]][c_2[[\![e_2]\!]_\Gamma]])_{[\![\Sigma x:C.D]\!]_\Gamma} \\
& \quad \{ \textbf{Définition de } d_2 \} \\
\hat{=}\ & (c'_1[[\![e'_1]\!]_\Gamma], d'_2[c_2[[\![e_2]\!]_\Gamma]])_{[\![\Sigma x:C.D]\!]_\Gamma} \\
& \quad \{ \textbf{Définition de } d'_2 \} \\
\hat{=}\ & (c'_1[[\![e'_1]\!]_\Gamma], (c'_2 \circ \beta \circ c_2^{-1})[c_2[[\![e_2]\!]_\Gamma]])_{[\![\Sigma x:C.D]\!]_\Gamma} \\
& \quad \{ c_2^{-1} \circ c_2 \equiv \bullet \} \\
\hat{=}\ & (c'_1[[\![e'_1]\!]_\Gamma], c'_2[\beta[[\![e_2]\!]_\Gamma]])_{[\![\Sigma x:C.D]\!]_\Gamma} \\
& \quad \{ \textbf{Hypothèse sur } \beta \} \\
\hat{=}\ & (c'_1[[\![e'_1]\!]_\Gamma], c'_2[[\![e'_2]\!]_\Gamma])_{[\![\Sigma x:C.D]\!]_\Gamma} \\
& \quad \{ \textbf{Définition de l'interprétation} \} \\
\hat{=}\ & [\![(e'_1, e'_2)_{\Sigma x:C.D}]\!]_\Gamma
\end{aligned}
$$

– ProdTeq, Sigma-≡, Subset-≡ : La traduction est un homomorphisme sur ces termes, c'est donc direct par induction.

□

Corollaire 2.3.25 (Conservation de l'équivalence pour les types). *Si* $\Gamma \vdash_\bullet T, U : s$ *et* $T \equiv_{\beta\pi} U$ *alors* $[\![T]\!]_\Gamma \equiv [\![U]\!]_\Gamma$.

On peut maintenant montrer notre théorème de correction.

Théorème 2.3.26 (Correction de l'interprétation). *Si* $\Gamma \vdash_\bullet t : T$ *alors on a* $[\![\Gamma]\!] \vdash_{CCI} [\![t]\!]_{[\![\Gamma]\!]} : [\![T]\!]_{[\![\Gamma]\!]}$. *Si* $\vdash_\bullet \simeq \mathbf{wf} G$ *alors* $\vdash_\bullet [\![\Gamma]\!]\ \mathbf{wf}$. *Si* $\Gamma \vdash_\bullet T, U : s$ *et* $T \rhd_\bullet U$ *alors il existe* $c, \Gamma \vdash_{CCI} c : T \rhd_\bullet U$ *et* $[\![\Gamma]\!] \vdash_{CCI} \lambda x : [\![T]\!].c[x] : [\![T]\!]_\Gamma \to [\![U]\!]_\Gamma$.

Démonstration. Par induction mutuelle sur les dérivations de typage, bonne formation et coercion.

– Wf-Empty : Trivial.

– Wf-Var : Par induction $[\![\Gamma]\!] \vdash_{CCI} [\![A]\!]_{[\![\Gamma]\!]} : [\![s]\!]_{[\![\Gamma]\!]}$. Par inversion du jugement de bonne formation dans Cci, $\vdash [\![\Gamma]\!]$. Or, $[\![s]\!]_{[\![\Gamma]\!]} = s$ ($s \in \{\texttt{Prop}, \texttt{Set}, \texttt{Type}\}$), donc on peut appliquer Wf-Var dans Cci pour obtenir : $\vdash_\bullet [\![\Gamma]\!]\ \mathbf{wf}, x : [\![A]\!]_{[\![\Gamma]\!]}$, soit $\vdash_\bullet [\![\ ]\!]\mathbf{wf}\Gamma, x : A$.

– PropSet : Direct par induction, $[\![\Gamma]\!] \vdash_{CCI} [\![s]\!]_{[\![\Gamma]\!]} = s : [\![\texttt{Type}]\!]_{[\![\Gamma]\!]} = \texttt{Type}$.

– Var : On a :

$$\frac{\vdash_\bullet \Gamma \ \mathbf{wf} \qquad x : A \in \Gamma}{\Gamma \vdash_\bullet x : A}$$

Par induction, $\vdash_\bullet [\![\Gamma]\!]$ **wf** et par simple inspection de la définition de l'interprétation des contextes, si $x : A \in \Gamma$ alors $x : [\![A]\!]_\Delta \in [\![\Gamma]\!]$ pour $\Delta \subset \Gamma$. Par affaiblissement dans Cci, on obtient aisément $[\![\Gamma]\!] \vdash_{CCI} x : [\![A]\!]_\Delta$ à partir de $[\![\Delta]\!] \vdash_{CCI} x : [\![A]\!]_\Delta$. Or il est clair par la définition de l'interprétation que $[\![A]\!]_\Delta = [\![A]\!]_\Gamma$ puisque les variables libres de A sont strictement contenues dans Δ.

– Prod : On a :

$$\frac{\Gamma \vdash_\bullet T : s_1 \qquad \Gamma, x : T \vdash_\bullet U : s_2}{\Gamma \vdash_\bullet \Pi x : T.U : s_3} (s_1, s_2, s_3) \in \mathcal{R}$$

Par induction $[\![\Gamma]\!] \vdash_{CCI} [\![T]\!]_{[\![\Gamma]\!]} : [\![s_1]\!]_{[\![\Gamma]\!]} = s_1$ et $[\![\Gamma, x : T]\!] \vdash_{CCI} [\![U]\!]_{[\![\Gamma,x:T]\!]} : [\![s_2]\!]_{[\![\Gamma,x:T]\!]} = s_2$. On déplie l'interprétation pour obtenir : $[\![\Gamma]\!], x : [\![T]\!]_\Gamma \vdash_{CCI} [\![U]\!]_{\Gamma,x:T} : s_2$.
Par Prod dans Cci, on obtient : $[\![\Gamma]\!] \vdash_{CCI} \Pi x : [\![T]\!]_\Gamma.[\![U]\!]_{\Gamma,x:T} : s_3$. Or $[\![\Pi x : T.U]\!]_\Gamma = \Pi x : [\![T]\!]_\Gamma.[\![U]\!]_{\Gamma,x:T}$, donc on a bien : $[\![\Gamma]\!] \vdash_{CCI} [\![\Pi x : T.U]\!]_\Gamma : s_3 = [\![s_3]\!]_\Gamma$.

– Abs : On a :

$$\frac{\Gamma \vdash_\bullet \Pi x : T.U : s \qquad \Gamma, x : T \vdash_\bullet M : U}{\Gamma \vdash_\bullet \lambda x : T.M : \Pi x : T.U}$$

Par induction $\Gamma \vdash_{CCI} [\![\Pi x : T.U]\!]_\Gamma : [\![s]\!]_\Gamma$ et $[\![\Gamma, x : T]\!] \vdash_{CCI} [\![M]\!]_{[\![\Gamma,x:T]\!]} : [\![U]\!]_{[\![\Gamma,x:T]\!]}$. On déplie l'interprétation pour obtenir : $[\![\Gamma]\!], x : [\![T]\!]_\Gamma \vdash_{CCI} [\![M]\!]_{\Gamma,x:[\![T]\!]_{[\![\Gamma]\!]}} : [\![U]\!]_{[\![\Gamma,x:T]\!]}$.
Par Abs dans Cci, on obtient : $[\![\Gamma]\!] \vdash_{CCI} \lambda x : [\![T]\!]_\Gamma.[\![M]\!]_{\Gamma,x:T} : [\![\Pi x : T.U]\!]_\Gamma$. C'est équivalent à : $[\![\Gamma]\!] \vdash_{CCI} [\![\lambda x : T.U]\!]_\Gamma : [\![\Pi x : T.U]\!]_\Gamma$.

– App : On a :

$$\frac{\Gamma \vdash_\bullet f : T \quad \mu_\bullet(T) = \Pi x : V.W : s \qquad \Gamma \vdash_\bullet u : U \qquad \Gamma \vdash_\bullet U \rhd_\bullet V : s'}{\Gamma \vdash_\bullet (fu) : W[u/x]}$$

C'est le cas intéressant puisqu'on ajoute ici des coercions. Par induction, $[\![\Gamma]\!] \vdash_{CCI} [\![f]\!]_{[\![\Gamma]\!]} : [\![T]\!]_\Gamma$, $[\![\Gamma]\!] \vdash_{CCI} [\![u]\!]_\Gamma : [\![U]\!]_\Gamma$ et $[\![\Gamma]\!] \vdash_{CCI} [\![U]\!]_\Gamma, [\![V]\!]_\Gamma : [\![s']\!]_\Gamma = s'$.
On procéde par étapes : d'abord la construction d'une fonction $[\![\Gamma]\!] \vdash_{CCI} \pi[[\![f]\!]_\Gamma] : [\![\Pi x : V.W]\!]_\Gamma$ puis la construction de l'argument $[\![\Gamma]\!] \vdash_{CCI} c[[\![u]\!]_\Gamma] : [\![V]\!]_\Gamma$. On a ces deux résultats par application de l'hypothèse de récurrence pour les coercions. On n'a plus qu'à les appliquer pour obtenir le jugement : $[\![\Gamma]\!] \vdash_{CCI} [\![f\ u]\!]_\Gamma : [\![W]\!]_{\Gamma,x:V}[c[[\![u]\!]_\Gamma]/x]$. Par le lemme 2.3.21, on a $[\![W]\!]_{\Gamma,x:V}[c[[\![u]\!]_\Gamma]/x] \equiv [\![W[u/x]]\!]_\Gamma$ puisque les coercions de sorte à sorte ne peuvent être que l'identité. On peut donc déduire : $[\![\Gamma]\!] \vdash_{CCI} [\![f\ u]\!]_\Gamma : [\![W[u/x]]\!]_\Gamma$ par Conv.

– Sum, Sum, Let-Sum, Subset : Direct par induction ou un raisonnement similaire à App.

– $\rhd$-Conv : Par le lemme 2.3.24 on a $[\![T]\!]_\Gamma \equiv [\![U]\!]_\Gamma$, on peut donc dériver :

$$\frac{[\![\Gamma]\!] \vdash_{CCI} t : [\![T]\!]_\Gamma \qquad [\![T]\!]_\Gamma \equiv [\![U]\!]_\Gamma}{[\![\Gamma]\!] \vdash_{CCI} c[t] = t : [\![U]\!]_\Gamma}$$

– $\rhd$-$\downarrow$: Par induction on a $[\![\Gamma]\!] \vdash_{CCI} t : [\![T^\downarrow]\!]_\Gamma \Rightarrow [\![\Gamma]\!] \vdash_{CCI} c[t] : [\![U^\downarrow]\!]_\Gamma$. A l'aide du lemme 2.3.24 on peut dériver :

$$\dfrac{\dfrac{\dfrac{[\![\Gamma]\!] \vdash_{CCI} t : [\![T]\!]_\Gamma \qquad [\![T]\!]_\Gamma \equiv [\![T^\downarrow]\!]_\Gamma}{[\![\Gamma]\!] \vdash_{CCI} t : [\![T^\downarrow]\!]_\Gamma}}{[\![\Gamma]\!] \vdash_{CCI} c[t] : [\![U^\downarrow]\!]_\Gamma} \qquad [\![U^\downarrow]\!]_\Gamma \equiv [\![U]\!]_\Gamma}{[\![\Gamma]\!] \vdash_{CCI} c[t] : [\![U]\!]_\Gamma}$$

– $\rhd$-Prod : On a :

$$\frac{\Gamma \vdash_{CCI} c_1 : U \rhd_\bullet T : s_1 \qquad \Gamma, x : U \vdash_{CCI} c_2 : V \rhd_\bullet W : s_2}{\Gamma \vdash_{CCI} \lambda x : [\![U]\!]_\Gamma . c_2[\bullet\ (c_1[x])] : \Pi x : T.V \rhd_\bullet \Pi x : U.W : s_2}\ (s_1, s_2) \in \mathcal{R}$$

Supposons $[\![\Gamma]\!] \vdash_{CCI} t : [\![\Pi x : T.V]\!]_\Gamma$. Alors $c[t] = \lambda x : [\![U]\!]_\Gamma . c_2[t\ c_1[x]]$.
Par induction, $[\![G, x : U]\!] \vdash_{CCI} c_1[x] : [\![T]\!]_\Gamma$, donc par App, $[\![G, x : U]\!] \vdash_{CCI} t\ c_1[x] : [\![V]\!]_{\Gamma, x:T}[c_1[x]/x]$. Par stabilité par affaiblissement (2.3.19), $[\![V]\!]_{\Gamma,x:T}[c_1[x]/x] \equiv [\![V]\!]_{\Gamma,x:U}$. On peut donc dériver par $\rhd$-Conv :

$$[\![G, x : U]\!] \vdash_{CCI} t\ c_1[x] : [\![V]\!]_{\Gamma,x:V}$$

Par induction, $[\![G, x : U]\!] \vdash_{CCI} c_2[t\ c_1[x]] : [\![W]\!]_\Gamma$, donc par Abs,

$$[\![\Gamma]\!] \vdash_{CCI} \lambda x : [\![U]\!]_\Gamma . c_2[t\ c_1[x]] : [\![\Pi x : U.W]\!]_\Gamma$$

– $\rhd$-Sum : On a :

$$\frac{\Gamma \vdash_{CCI} c_1 : T \rhd_\bullet U : s \qquad \Gamma, x : T \vdash_{CCI} c_2 : V \rhd_\bullet W : s}{\Gamma \vdash_{CCI} (c_1[\pi_1\ \bullet], c_2[\pi_2\ \bullet][\pi_1\ \bullet/x])_{[\![\Sigma x:U.W]\!]_\Gamma} : \Sigma x : T.V \rhd_\bullet \Sigma x : U.W : s}$$

Ici, $c[t] = (c_1[\pi_1\ t], c_2[\pi_1\ t/x][\pi_2\ t])_{[\![\Sigma x:U.W]\!]_\Gamma}$. Par application des règles de typage pour les projections : $[\![\Gamma]\!] \vdash_{CCI} \pi_1\ t : [\![T]\!]_\Gamma$ et $[\![\Gamma]\!] \vdash_{CCI} \pi_2\ t : [\![V]\!]_{\Gamma,x:T}[\pi_1\ t/x]$.
Par induction on obtient $[\![\Gamma]\!] \vdash_{CCI} c_1[\pi_1\ t] : [\![U]\!]_\Gamma$. Par induction on a aussi $[\![\Gamma, x : T]\!] \vdash_{CCI} \lambda y : [\![V]\!]_\Gamma . c_2[\pi_2\ y] : [\![V]\!]_\Gamma \to [\![W]\!]_\Gamma$, où y n'apparait pas dans c_2.
Par affaiblissement, on passe dans l'environnement $\Gamma, t : \Sigma x : T.V, x : T$:

$$[\![\Gamma, t : \Sigma x : T.V, x : T]\!] \vdash_{CCI} \lambda y : [\![V]\!]_\Gamma . c_2[\pi_2\ y] : [\![V]\!]_\Gamma \to [\![W]\!]_\Gamma$$

On peut alors substituer pour obtenir : $[\![\Gamma, t : \Sigma x : T.V]\!] \vdash_{CCI} \lambda y : [\![V]\!]_\Gamma[\pi_1\ t/x] . c_2[\pi_1\ t/x][\pi_2\ y] : [\![V]\!]_\Gamma[\pi_1\ t/x] \to [\![W]\!]_\Gamma[\pi_1\ t/x]$.
Enfin on applique à $\pi_2\ t$ pour obtenir : $[\![\Gamma, t : \Sigma x : T.V]\!] \vdash_{CCI} c_2[\pi_1\ t/x][\pi_2\ t] : [\![W]\!]_\Gamma[\pi_1\ t/x]$.
On a bien :

$$[\![\Gamma]\!] \vdash_{CCI} \lambda t : [\![\Sigma x : T.V]\!]_\Gamma . (c_1[\pi_1\ t], c_2[\pi_1\ t/x][\pi_2\ t])_{[\![\Sigma x:U.W]\!]_\Gamma} : [\![\Sigma x : T.V]\!]_\Gamma \to [\![\Sigma x : U.W]\!]_\Gamma$$

– $\rhd$-Proof : On a :

$$\frac{\Gamma \vdash_{CCI} c : T \rhd_\bullet U : \mathtt{Set}}{\Gamma \vdash_{CCI} \mathrm{elt}\ [\![U]\!]_\Gamma\ [\![\lambda x : U.P]\!]_\Gamma\ c\ ?_{[\![P]\!]_{\Gamma,x:U}[c/x]} : T \rhd_\bullet \{\ x : U \mid P\ \} : \mathtt{Set}}\ T = T^\downarrow$$

Par induction, $[\![\Gamma]\!] \vdash_{CCI} \lambda x : [\![T]\!]_\Gamma . c[x] : [\![T]\!]_\Gamma \to [\![U]\!]_\Gamma$ et $[\![\Gamma]\!] \vdash_{CCI} \{\ x : [\![U]\!]_\Gamma \mid [\![P]\!]_{\Gamma,x:U}\ \} : \mathtt{Set}$.
On a donc clairement :

$$[\![\Gamma]\!] \vdash_{CCI} \lambda t : [\![T]\!]_\Gamma . \mathrm{elt}\ [\![U]\!]_\Gamma\ [\![\lambda x : U.P]\!]_\Gamma\ c[t]\ ?_{[\![P]\!]_{\Gamma,x:U}[c[t]/x]} : [\![T]\!]_\Gamma \to [\![\{\ x : U \mid P\ \}]\!]_\Gamma$$

– $\rhd$-Subset : On a :

$$\frac{\Gamma \vdash_{CCI} c : U \rhd_\bullet T : \mathtt{Set}}{\Gamma \vdash_{CCI} c[\sigma_1\ \bullet] : \{\ x : U \mid P\ \} \rhd_\bullet T : \mathtt{Set}}\ T = T^\downarrow$$

Par induction, $[\![\Gamma]\!] \vdash_{CCI} \lambda x : [\![U]\!]_\Gamma . c[x] : [\![U]\!]_\Gamma \to [\![T]\!]_\Gamma$, on a bien :

$$[\![\Gamma]\!] \vdash_{CCI} \lambda t : [\![\{\ x : U \mid P\ \}]\!]_\Gamma . c[\sigma_1\ t] : [\![\{\ x : U \mid P\ \}]\!]_\Gamma \to [\![T]\!]_\Gamma$$

□

$$
\begin{array}{lcll}
[\![x]\!]_\Gamma & = & x & \\
[\![s]\!]_\Gamma & = & s & s \in \{\mathtt{Set}, \mathtt{Prop}, \mathtt{Type}\} \\
[\![\Pi x : T.U]\!]_\Gamma & = & \Pi x : [\![T]\!]_\Gamma.[\![U]\!]_{\Gamma,x:T} & \\
[\![\lambda x : \tau.v]\!]_\Gamma & = & \textbf{let } \tau' = [\![\tau]\!]_\Gamma \textbf{ in} & \\
 & & \textbf{let } v' = [\![v]\!]_{\Gamma,x:\tau} \textbf{ in} & \\
 & & (\lambda x : \tau'.v') & \\
[\![f\ u]\!]_\Gamma & = & \textbf{let } F\ = \textbf{type}_\Gamma(f) \textbf{ and } U = \textbf{type}_\Gamma(u) \textbf{ in} & \\
 & & \textbf{let } (\Pi x : V.W) = \mu_\bullet(F) \textbf{ in} & \\
 & & \textbf{let } \pi = \textbf{coerce}_\Gamma\ F\ (\Pi x : V.W) \textbf{ in} & \\
 & & \textbf{let } c = \textbf{coerce}_\Gamma\ U\ V \textbf{ in} & \\
 & & (\pi[[\![f]\!]_\Gamma])\ (c[[\![u]\!]_\Gamma]) & \\
[\![\Sigma x : T.U]\!]_\Gamma & = & \Sigma x : [\![T]\!]_\Gamma.[\![U]\!]_{\Gamma,x:T} & \\
[\![(t,u)_{\Sigma x:T.U}]\!]_\Gamma & = & \textbf{let } t' = [\![t]\!]_\Gamma \textbf{ in} & \\
 & & \textbf{let } T' = \textbf{type}_\Gamma(t) \textbf{ in} & \\
 & & \textbf{let } ct = \textbf{coerce}_\Gamma\ T'\ T \textbf{ in} & \\
 & & \textbf{let } U' = \textbf{type}_\Gamma(u) \textbf{ in} & \\
 & & \textbf{let } u' = [\![u]\!]_\Gamma \textbf{ in} & \\
 & & \textbf{let } cu = \textbf{coerce}_\Gamma\ U'\ U[t/x] \textbf{ in} & \\
 & & (ct[t'], cu[u'])_{[\![\Sigma x:T.U]\!]_\Gamma} & \\
[\![\pi_i\ t]\!]_\Gamma & = & \textbf{let } t' = [\![t]\!]_\Gamma \textbf{ in} & i \in \{1,2\} \\
 & & \textbf{let } T = \textbf{type}_\Gamma(t) \textbf{ in} & \\
 & & \textbf{let } \Sigma x : V.W = \mu_\bullet(T) \textbf{ in} & \\
 & & \textbf{let } c = \textbf{coerce}_\Gamma\ T\ (\Sigma x : V.W) \textbf{ in} & \\
 & & \pi_i\ c[t'] & \\
[\![\{\ x : U \mid P\ \}]\!]_\Gamma & = & \{\ x : [\![U]\!]_\Gamma \mid [\![P]\!]_{\Gamma,x:U}\ \} &
\end{array}
$$

FIG. 2.8 – Interprétation dans CCI

$$
\begin{array}{lcll}
((\lambda x : T.e)\ v)^\downarrow & = & e[v/x]^\downarrow & \\
\pi_1\ (x,y)^\downarrow & = & x^\downarrow & \\
\pi_2\ (x,y)^\downarrow & = & y^\downarrow & \\
e^\downarrow & = & e & \{\text{si } e \text{ est d'une autre forme}\}
\end{array}
$$

FIG. 2.9 – Définition de la réduction de tête

$$
\begin{array}{llcll}
(\beta) & (\lambda x : X.e)\ v & = & e[v/x] & \\
(\pi_i) & \pi_i\ (e_1, e_2)_T & = & e_i & \\
(\sigma_i) & \sigma_i\ (\mathrm{elt}\ E\ P\ e_1\ e_2) & = & e_i & \\
(\eta) & (\lambda x : X.e\ x) & = & e & \text{si } x \notin FV(e) \\
(\rho) & (\pi_1\ e, \pi_2\ e)_{\Sigma x:X.Y} & = & e & \text{si } e : \Sigma x : X.Y \\
 & \mathrm{elt}\ E\ P\ (\sigma_1\ e)\ (\sigma_2\ e) & = & e & \text{si } e : \{\ x : E \mid P\ \} \\
(\sigma) & \mathrm{elt}\ E\ P\ t\ p & = & \mathrm{elt}\ E\ P\ t'\ p' & \text{si } t \equiv t'
\end{array}
$$

FIG. 2.10 – Théorie équationnelle de CC?

$$
\frac{T \equiv_{\beta\pi} U \quad \Gamma \vdash_\bullet T, U : s}{\Gamma \vdash_{CCI} \bullet : T \rhd_\bullet U : s}\ T = T^\downarrow \wedge T \neq \Pi, \Sigma, \{|\} \wedge U = U^\downarrow
$$

$$
\frac{\Gamma \vdash_{CCI} c : T^\downarrow \rhd_\bullet U^\downarrow : s}{\Gamma \vdash_{CCI} c : T \rhd_\bullet U : s}\ T \neq T^\downarrow \vee U \neq U^\downarrow
$$

$$
\frac{\Gamma \vdash_{CCI} c_1 : U \rhd_\bullet T : s_1 \qquad \Gamma, x : U \vdash_{CCI} c_2 : V \rhd_\bullet W : s_2}{\Gamma \vdash_{CCI} \lambda x : [\![U]\!]_\Gamma . c_2[\bullet\ (c_1[x])] : \Pi x : T.V \rhd_\bullet \Pi x : U.W : s_2}\ (s_1, s_2) \in \mathcal{R}
$$

$$
\frac{\Gamma \vdash_{CCI} c_1 : T \rhd_\bullet U : s \qquad \Gamma, x : T \vdash_{CCI} c_2 : V \rhd_\bullet W : s}{\Gamma \vdash_{CCI} (c_1[\pi_1\ \bullet], c_2[\pi_2\ \bullet][\pi_1\ \bullet / x])_{[\![\Sigma x:U.W]\!]_\Gamma} : \Sigma x : T.V \rhd_\bullet \Sigma x : U.W : s}
$$

$$
\frac{\Gamma \vdash_{CCI} c : U \rhd_\bullet T : \mathtt{Set}}{\Gamma \vdash_{CCI} c[\sigma_1\ \bullet] : \{\ x : U \mid P\ \} \rhd_\bullet T : \mathtt{Set}}\ T = T^\downarrow
$$

$$
\frac{\Gamma \vdash_{CCI} c : T \rhd_\bullet U : \mathtt{Set}}{\Gamma \vdash_{CCI} \mathrm{elt}\ [\![U]\!]_\Gamma\ [\![\lambda x : U.P]\!]_\Gamma\ c\ ?_{[\![P]\!]_{\Gamma, x:U}[c/x]} : T \rhd_\bullet \{\ x : U \mid P\ \} : \mathtt{Set}}\ T = T^\downarrow
$$

FIG. 2.11 – Réécriture de la coercion vers CCI

$$\textsc{VarTeq}\ \frac{}{\Gamma \vdash_\bullet x : X \equiv_{\beta\pi} x : X}$$

$$\textsc{SortTeq}\ \frac{}{\Gamma \vdash_\bullet s : \texttt{Type} \equiv_{\beta\pi} s : \texttt{Type}}\ s \in \{\texttt{Set}, \texttt{Prop}\}$$

$$\beta\text{-}\equiv\ \frac{\Gamma \vdash_\bullet U \rhd_\bullet T :}{\Gamma \vdash_\bullet (\lambda x : X.v)\ e : T \equiv_{\beta\pi} v[e/x] : U}$$

$$\Pi_1\text{-}\equiv\ \frac{\Gamma \vdash_\bullet U \rhd_\bullet T :}{\Gamma \vdash_\bullet \pi_1\ (e_1, e_2)_{\Sigma x:T.V} : T \equiv_{\beta\pi} e_1 : U}$$

$$\Pi_2\text{-}\equiv\ \frac{\Gamma \vdash_\bullet U \rhd_\bullet T :}{\Gamma \vdash_\bullet \pi_2\ (e_1, e_2)_{\Sigma x:V.T} : T \equiv_{\beta\pi} e_2 : U}$$

$$\textsc{LambdaTeq}\ \frac{\Gamma \vdash_\bullet X : s_1 \equiv_{\beta\pi} X' : s_1 \qquad \Gamma, x : X' \vdash_\bullet v : Y \equiv_{\beta\pi} v' : Y'}{\Gamma \vdash_\bullet \lambda x : X.v : \Pi x : X.Y \equiv_{\beta\pi} \lambda x : X'.v' : \Pi x : X'.Y'}$$

$$\textsc{App-}\equiv\ \frac{\begin{array}{c}\Gamma \vdash_\bullet M : S \equiv_{\beta\pi} M' : T \\ \mu_\bullet(S) = \Pi x : A.B \quad \mu_\bullet(T) = \Pi x : C.D \qquad \Gamma \vdash_\bullet N : U \equiv_{\beta\pi} N' : V \quad \Gamma \vdash_\bullet U \rhd_\bullet A : \qquad \Gamma \vdash_\bullet V \rhd_\bullet C :\end{array}}{\Gamma \vdash_\bullet M\ N : B[N/x] \equiv_{\beta\pi} M'\ N' : D[N'/x]}$$

$$\textsc{Pair-}\equiv\ \frac{\begin{array}{c}\Gamma \vdash_\bullet e_1 : A' \equiv_{\beta\pi} e_1' : C' \qquad\qquad\qquad\qquad\qquad\qquad \Gamma \vdash_\bullet B' \rhd_\bullet B[e_1/x] : \\ \Gamma, x : A' \vdash_\bullet e_2 : B' \equiv_{\beta\pi} e_2' : D' \qquad \Gamma \vdash_\bullet A' \rhd_\bullet A : \quad \Gamma \vdash_\bullet C' \rhd_\bullet C : \qquad \Gamma \vdash_\bullet D' \rhd_\bullet D[e_1/x] :\end{array}}{\Gamma \vdash_\bullet (e_1, e_2)_{\Sigma x:A.B} : \Sigma x : A.B \equiv_{\beta\pi} (e_1', e_2')_{\Sigma x:C.D} : \Sigma x : C.D}$$

$$\textsc{Prod-}\equiv\ \frac{\Gamma \vdash_\bullet C : s_1 \equiv_{\beta\pi} A : s_1 \qquad \Gamma, x : A \vdash_\bullet B : s_2 \equiv_{\beta\pi} D : s_2}{\Gamma \vdash_\bullet \Pi x : A.B : s_3 \equiv_{\beta\pi} \Pi x : C.D : s_3}$$

$$\textsc{Sigma-}\equiv\ \frac{\Gamma \vdash_\bullet A : s_1 \equiv_{\beta\pi} C : s_1 \qquad \Gamma, x : A \vdash_\bullet B : s_2 \equiv_{\beta\pi} D : s_2}{\Gamma \vdash_\bullet \Sigma x : A.B : s_3 \equiv_{\beta\pi} \Sigma x : C.D : s_3}$$

$$\textsc{Subset-}\equiv\ \frac{\Gamma \vdash_\bullet T : \texttt{Set} \equiv_{\beta\pi} A : \texttt{Set} \qquad \Gamma, x : T \vdash_\bullet P : \texttt{Prop} \equiv_{\beta\pi} P' : \texttt{Prop}}{\Gamma \vdash_\bullet \{\ x : T \mid P\ \} : \texttt{Set} \equiv_{\beta\pi} \{\ x : U \mid P'\ \} : \texttt{Set}}$$

Fig. 2.12 – Définition du jugement $\Gamma \vdash_\bullet t : T \equiv_{\beta\pi} u : U$

Chapitre 3

SUBTAC

Nous avons développé la contribution SUBTAC (pour "subset-tactics") disponible dans la version CVS courante de COQ (`http://coq.inria.fr`). Elle permet de typer un programme en RUSSELL et générer un terme incomplet correspondant (voir annexe A.2).

3.1 Existentielles

La génération des buts correspondant aux variables existentielles et la formation du terme final devaient originellement être laissées à la tactique REFINE et au système de gestion des existentielles de COQ. Certaines limitations dans l'implantation du raffinement (le mécanisme permettant de manipuler des termes "à trous") nous ont empêché d'utiliser REFINE. En particulier, la gestion des définitions récursives et la présence de variables existentielles dans les types d'autres existentielles n'étaient pas supportées. En conséquence, nous avons développé une nouvelle tactique COQ permettant de gérer les termes avec existentielles de façon plus générale.

3.1.1 La tactique eterm

L'idée de départ de la tactique REFINE est de prendre un terme à trous et d'en faire une traduction en une séquence de tactiques. Par exemple, lorsque REFINE rencontre une abstraction, il fait une introduction, lorsque c'est un cast, on applique l'identité et ainsi de suite. Intuitivement, la séquence de tactiques engendrée va construire le terme de départ implicitement.

La tactique **eterm** fonctionne différemment. À partir d'un terme t contenant des existentielles, **eterm** va généraliser le terme par rapport à celles-ci, et généraliser chaque existentielle par rapport à son contexte, créant ainsi un objet $(\lambda ex_1 : T_1, \ldots, ex_n : T_n, t[?_1 := ex_1, \ldots, ?_n := ex_n])$, où chaque ex_i est appliqué aux variables introduites dans son contexte par les abstractions. Habituellement, on propose t comme habitant d'un type T donné (le but), par exemple on peut proposer $\lambda x : \texttt{nat}.x$ comme preuve du but $\texttt{nat} \rightarrow \texttt{nat}$. Plutôt que de donner directement t, on applique le nouveau terme, et COQ va automatiquement nous demander d'instancier les arguments $ex_1 \ldots ex_n$ correspondant aux existentielles du terme t. Cette technique permet d'avoir des dépendances entre existentielles (par exemple, ex_1 peut apparaître dans tous les types $T_2 \ldots T_n$) et de ne pas reposer entièrement sur la gestion des existentielles de COQ qui n'est pas très flexible à l'heure actuelle. En particulier, si l'on applique une produits où il y a des dépendances entre arguments, COQ va recréer des existentielles.

Il nous faut nous pencher un peu plus avant sur la généralisation des existentielles pour comprendre le mécanisme d'**eterm**. Puisqu'on veut pouvoir avoir des dépendances entres les n existentielles d'un terme et qu'on sérialise celles-ci en un produit n-aire, il nous faut être très attentifs à l'ordre dans lequel on généralise les variables existentielles. Si $?_3 : T_3$ où T_3 référence $?_4$, il faut que l'existentielle ex_4 apparaisse *avant* ex_3 dans notre produit. Il est toujours possible de trouver un ordre compatible avec ces dépendances puisqu'il est impossible de créer un cycle où $?_i$ référencerait $?_j$ et vice-versa (ceci est assuré par le caractère fonctionnel

des objets implantant les existentielles dans Coq). La tactique actuelle est codée avec l'hypothèse que toute existentielle $?_i$ ne dépend pas des existentielles d'indice supérieur à i. Il est cependant envisageable de réécrire tout terme contenant des existentielles comme un terme équivalent avec des indices respectant cet ordre.

3.2 Traitement de la récursion

Lorsque l'on développe un programme récursif dans un système tel que Coq, on est forcé de fournir une preuve de sa terminaison. Pour cela, on montre généralement qu'on a un ordre bien fondé sur le type de l'argument de récursion et que chaque appel respecte cet ordre. Nous avons ajouté des facilités d'écriture de fonctions récursives à notre langage ; on ajoute les existentielles correspondant aux preuves que l'ordre est bien fondé ou qu'il est bien respecté par les termes. Ainsi lors du raffinement on obtient naturellement les buts correspondants à prouver.

La possibilité de faire des fonctions récursives à l'intérieur des termes devraient être facile à introduire, il suffit d'avoir un combinateur **fix** comme constante avec une forme pratique pour le langage : par exemple la fonction utilisable pour l'appel récursif devrait avoir un type de la forme : $\{ x : T \mid R\ x\ a \} \to B$ où R est la relation bien fondée et T le type de l'argument de récursion ainsi on génèrera les obligations automatiquement lors des appels récursifs.

3.3 Traitement des inductifs

Notre langage ne prend pas encore en compte les définitions inductives générales. Au-delà du traitement des types sous-ensemble, on a un support minimal pour les inductifs à deux constructeurs qui correspondent à des booléens annotés par des propriétés logiques (voir traitement de la conditionnelle figure A.2). A long terme on devrait pouvoir traiter les inductifs dans `Set` prédicatif, qui ne peuvent embarquer des propositions qu'en utilisant des types sous-ensemble avec le même mécanisme de coercion et conserver l'inférence.

3.4 `Program` et `Recursive program`

On peut utiliser notre contribution à l'aide des deux tactiques `Program` et `Recursive program`.

`Program` La syntaxe pour l'appel de cette tactique est la suivante : `Program` **name** $: \tau := \alpha.$ où τ, α dénottent les catégories syntaxiques des types et des termes respectivement. La tactique fonctionne ainsi : On infère le type du terme, on applique la coercion du terme vers le type spécifié puis on réécrit le terme obtenu dans Cci. On réécrit ensuite le type spécifié dans Cci et on le pose comme but. On utilise **eterm** qui va s'occuper du terme à trous et l'appliquer au but. On obtient alors automatiquement les obligations de preuves.

`Recursive program` Cette deuxième tactique permet d'écrire des définitions récursives, on va donc demander plus d'informations à l'utilisateur lors de la définition :

$$\texttt{Recursive program}\ \textbf{name}\ (a : \tau_a)\{\textbf{wf}\ R\ (\textbf{auto} \mid \textbf{proof}\ p)?\} :\ \tau := \alpha.$$

L'argument a de type τ_a est l'argument de récursion, R est la relation d'ordre qu'on doit montrer bien fondée. Si l'on utilise **auto** la tactique va chercher dans une base une preuve de bonne fondation. Si l'on utilise **proof** alors p doit référencer une preuve de la bonne fondation de R dans l'environnement courant. Sinon on génère une obligation de preuve que R est bien fondé. La tactique fait ensuite à peu près la même chose que `Program`. Le typage et la réécriture se font dans un contexte où la fonction $name : \Pi a : \tau_a.\tau$ apparaît et l'on vérifie à l'application si l'on fait un appel récursif auquel cas on insère une existentielle correspondant à la preuve que l'ordre est bien respecté (dans Coq, la fonction permettant de faire la récursion aura un

type de la forme $\Pi x : \tau_a.R\ x\ a \to \tau$). Un exemple de l'utilisation de `Recursive program` est donné figures A.2 et A.3.

Chapitre 4

Conclusion

Nous avons développé un langage de programmation plus souple que le langage de Coq mais conservant sa richesse d'expression (types dépendants). Il permet de découpler la description algorithmique de la vérification. La correction des termes engendrés est garantie par le système sous-jacent qui offre ensuite la possibilité d'extraire un programme correct par construction dans un langage de type ML. D'autre part, cette méthode s'intègre bien dans l'environnement Coq et ouvre la voie à la réalisation de travaux plus complexes par des utilisateurs non-experts. Cela constitue la première étape vers un environnement de programmation sûre utilisable dans Coq.

Travaux futurs

Nous avons de multiples directions dans lesquelles étendre notre système. Le support des inductifs, de la récursion à l'intérieur des termes, des univers cumulatifs et des définitions dans les contextes semblent des problèmes techniques abordables. La possibilité d'utiliser Russell dans d'autres parties de Coq (énoncés de lemmes, termes de tactiques) est aussi un objectif intéressant. Modifier Coq pour qu'il gère mieux les existentielles est aussi un problème crucial dont nous avons fait l'expérience durant le développement de Subtac.

Bibliographie

[1] Lennart Augustsson. Cayenne—A language with dependent types. In *ACM SIGPLAN International Conference on Functional Programming (ICFP), Baltimore, Maryland*, pages 239–250, 1998.

[2] B. Barras. Coq en coq. Rapport de Recherche 3026, INRIA, October 1996.

[3] B. Barras. *Auto-validation d'un système de preuves avec familles inductives*. Thèse de doctorat, Université Paris 7, November 1999.

[4] Giuseppe Castagna. Covariance and contravariance : Conflict without a cause. *ACM Trans. Program. Lang. Syst.*, 17(3) :431–447, 1995.

[5] Gang Chen. *Sous-typage, Conversion de Types et Élimination de la Transitivité*. PhD thesis, Université Paris VII, Laboratoire d'Informatique de l'Ecole Normale Supérieure, Paris, Décembre 1998.

[6] Gang Chen. Coercive subtyping for the calculus of constructions. In *POPL*, pages 150–159, 2003.

[7] Georges Gonthier and Benjamin Werner. A computer-checked proof of the four-coulour theorem. April 2005, `http://research.microsoft.com/~gonthier/4colproof.pdf`.

[8] Xavier Leroy. Formal certification of a compiler back-end, or : programming a compiler with a proof assistant, 2005. Draft. `http://pauillac.inria.fr/~xleroy/publi/compiler-certif.pdf`.

[9] Pierre Letouzey. *Programmation fonctionnelle certifiée – L'extraction de programmes dans l'assistant Coq*. PhD thesis, Université Paris-Sud, July 2004.

[10] Zhaohui Luo. *An Extended Calculus of Constructions*. PhD thesis, Department of Computer Science, University of Edinburgh, June 1990.

[11] Zhaohui Luo. Coercive subtyping in type theory. In Dirk van Dalen and Marc Bezem, editors, *CSL*, volume 1258 of *Lecture Notes in Computer Science*, pages 276–296. Springer, 1996.

[12] Sam Owre and Natarajan Shankar. The formal semantics of PVS. Technical Report SRI-CSL-97-2, Computer Science Laboratory, SRI International, Menlo Park, CA, August 1997.

[13] Catherine Parent. Synthesizing proofs from programs in the calculus of inductive constructions. In Bernhard Möller, editor, *MPC*, volume 947 of *Lecture Notes in Computer Science*, pages 351–379. Springer, 1995.

[14] John Rushby, Sam Owre, and N. Shankar. Subtypes for specifications : Predicate subtyping in PVS. *IEEE Transactions on Software Engineering*, 24(9) :709–720, September 1998.

[15] Amokrane Saïbi. Typing algorithm in type theory with inheritance. In *24th Annual Symposium on Principles of Programming Languages*, pages 292–, La Sorbonne, Paris, France, January 15-17 1997. ACM. `http://pauillac.inria.fr/~saibi/ClassCoq4.ps`.

[16] Natarajan Shankar and Sam Owre. Principles and pragmatics of subtyping in PVS. In Didier Bert, Christine Choppy, and Peter Mosses, editors, *Recent Trends in Algebraic Development Techniques, WADT '99*, volume 1827 of *Lecture Notes in Computer Science*, pages 37–52, Toulouse, France, September 1999. Springer-Verlag.

[17] Tim Sheard. Languages of the future. `http://www.cs.pdx.edu/~sheard/papers/LangOfTheFuture.ps`.

[18] Hongwei Xi and Frank Pfenning. Dependent types in practical programming. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), San Antonio, Texas*, pages 214–227, January 1999.

Ce rapport a été préparé sous LaTeX avec la fonte `Concrete Math`

Annexe A

Exemples

```
Definition div : forall a : nat, forall b : nat,
  b <> 0 -> { q : nat & { r : nat | r < b /\ a = b * q + r } }.
Proof.
intros a ; pattern a ; apply lt_wf_rec ; intros. (* Récursion *)
elim (lt_ge_dec n b). (* If then else *)
intros. (* Première branche *)
(* Structure du terme *)
refine (existS _ 0 _) ; refine (exist _ n _) ; refine (conj _ _) ;
[ assumption | rewrite mult_0_r ; rewrite plus_0_l ; reflexivity ]. (* Preuve *)
(* Seconde branche *)
intros ; assert (n - b < n). (* Preuve pour l'appel *)
apply lt_minus ; [ apply (ge_le _ _ b0) | apply (nat_neq_0_gt_0 b H0) ].
induction (H (n - b) H1 b H0). (* Appel récursif *)
induction p ; induction p. (* Destruction du résultat *)
refine (existS _ (S x) _) ; refine (exist _ x0 _). (* Structure du terme *)
(* Preuve *)
split.
assumption.
pose (eq_plus_eq _ _ H3 b).
assert (n - b + b = n) ; try omega.
rewrite <- H4 ; rewrite e ; rewrite plus_comm ; rewrite plus_assoc.
replace (b + b * x) with (b * S x).
reflexivity.
rewrite mult_comm ; simpl ; pattern (x * b) ; rewrite mult_comm.
reflexivity.
Qed.
```

FIG. A.1 – Script de preuve de la division euclidienne

```
(* Subtac ne gère pas encore les notations de Coq *)
Definition neq (A : Type) (x y : A) : Prop := x <> y.
Definition div_prop (a b q r : nat) := a = (b * q) + r /\ r < b.
Definition lt_ge_dec (x y : nat) : { x < y } + { x >= y }.
Proof.
  intros ; elim (le_lt_dec y x) ; intros ; auto with arith.
Defined.

Recursive program mydiv (a : nat) { well_founded lt a lt_wf } :
  { b : nat | neq nat b O } ->
  [ q : nat ] { r : nat | div_prop a b q r } :=
  fun { y : nat | neq nat y O } =>
    if lt_ge_dec a y
    then (q := O, a : { r : nat | div_prop a y q r })
    else let (q', r) = mydiv (minus a y) y in
        (q := S q', r : { r : nat | div_prop a y q r }).

(* Dans Coq, mydiv aura le type:
forall a : nat, forall b : { b : nat | b <> 0 },
{ q : nat & { r : nat | div_prop a (proj1_sig b) q r } } *)

(* Obligations de preuves engendrées *)
(* Hypothèses communes: *)
a : nat
mydiv : (n : nat) n < a -> forall b : { b : nat | b <> 0 },
 { q : nat & { r : nat | div_prop n (proj1_sig b) q r } }
y : { b : nat | b <> 0 }

(* (q := 0, a ...)
[ H : a < proj1_sig y, |- div_prop a (proj1_sig y) 0 a]

(* Argument de récursion *)
[H : a >= proj1_sig y |- a - proj1_sig y < a]

(* (q := S q', r) *)
[ H : a >= proj1_sig y, q' : nat,
  r : { r : nat | div_prop (a - proj1_sig y) (proj1_sig y) q' r }
|- div_prop a (proj1_sig y) (S q')  (proj1_sig r)]
```

FIG. A.2 – La division euclidienne avec SUBTAC

```
Recursive program mydiv (a : nat) using lt proof lt_wf  :
  { b : nat | neq nat b 0 } -> [ q : nat ] { r : nat | div_prop a b q r } :=
  fun { b : nat | neq nat b 0 } =>
    if lt_ge_dec a b
      then (q := 0, a : { r : nat | div_prop a b q r })
      else let (q', r) = mydiv (minus a b) b in
        (q := S q', r : { r : nat | div_prop a b q r }).
unfold neq ; simpl ; intros.
induction b ; simpl ; simpl in H ; omega.

unfold neq, div_prop ; simpl ; intros.
induction b ; induction r ; simpl ; simpl in H, p0 ; intuition.
rewrite mult_comm ; simpl.
rewrite mult_comm ; simpl.
omega.

unfold neq, div_prop ; simpl ; induction b ; simpl ; intros.
intuition.
Qed.
```

FIG. A.3 – La division euclidienne avec SUBTAC : script