Mettons de l'Ordre dans CIC !

Théo Laurent[1] and Kenji Maillard[2]

[1] Inria Paris, Équipe Prosecco, Paris
[2] Inria Rennes, Équipe Gallinette, Nantes

Résumé

Les langages à types dépendant souffrent régulièrement de problèmes de modularité. L'ajout de sous-typage permet d'améliorer l'interopérabilité entre composants. Cependant, l'étude de la métathéorie des langages résultants, cruciale dans le cadre des assistants à la preuve, dépend souvent de propriétés syntaxiques délicates à établir [Luo88 ; Miq01]. Nous proposons une approche à base de modèles syntaxiques [BPT17b] pour capturer différentes notions de sous-typage en théorie des types et en particulier pour le Calcul des Constructions Inductives (CIC). Ces modèles servent de prototypes et de guides pour la conception d'extensions de CIC munies de sous-typage, de types à raffinement ou de cumulativité. Cette approche systématique vise à combler l'écart existant entre les différents systèmes de sous-typage présentés dans la littérature et à proposer des solutions aux limitations parfois artificielles de leurs implémentations dans des assistants à la preuve.

1 Introduction

Les langages de programmation comme les assistants de preuve souffrent parfois de problèmes d'interopérabilité entre composants. Ceux-ci se manifestent sous la forme d'une rigidité importante du système : par exemple une fonction définie sur les entiers naturels $\mathbb{N}$ ne s'applique *a priori* pas au type des entiers naturels pairs Even, bien qu'un utilisateur puisse légitimement considérer qu'il existe une coercion canonique entre les deux types. Les disciplines de sous-typage proposent de pallier ce problème en introduisant dans le système de type de telles approximations. Ces disciplines peuvent prendre des formes variées : cumulativité, sous-typage structurel, types à raffinement, degrés d'effacement (irrelevance, de tailles, ...).

En présence de types dépendants, les coercions $\vdash \mathsf{coe}_X^Y\, t : Y$ sur un terme $\vdash t : X$ induites par l'ordre de sous-typage $X <: Y$ apparaissent aussi dans les types. Pour obtenir une discipline efficace de sous-typage, les différentes constructions du langage doivent interagir avec cet ordre (monotonicité, irrelevance). Les problématiques engendrées par ces interactions non-triviales se retrouvent sous des formes similaires quelle que soit la discipline de sous-typage envisagée.

Dans ce document nous proposons une feuille de route pour établir de manière solide les propriétés métathéoriques de calculs à types dépendants munis de sous-typage. Nous établissons un panorama succinct des nombreuses contributions à ce domaine dans la Section 2 et décrivons certains problèmes ouverts motivant nos travaux. Notre méthodologie, présentée en Section 3, s'appuie sur l'étude de modèles syntaxiques de CIC [BPT17a] équipant les types avec une relation de sous-typage (Section 4). Les pistes de réflexion présentées dans ce papier constituent des travaux de recherche en cours.

2 État de l'art & Applications Envisagées

La littérature autour de CIC abonde d'extensions munissant CIC d'une forme d'ordre : types à raffinement [Soz07], sous-typage structurel [LA08], cumulativité [Ass14 ; TS18], types munis de tailles pour la récursion gardée [Abe08b ; Abe08a ; AP13].

```
Inductive vect A : ℕ → □:=                    Inductive vect A (n : ℕ) : □:=
| nil : vect A 0                              | nil : ∀ n, n = 0 → vect A n
| cons : A → vect A m → vect A (S m).         | cons : ∀ n m, n = S m → A → vect A m → vect A n.
```

FIGURE 1 – Vecteurs (à gauche) et leur traduction de fordisme (à droite).

Sous-typages subsumptif et coercif. L'approche *subsumptive* au sous-typage part du postulat que les coercions sont entièrement implicites et transparentes : aucune information n'apparaît dans les termes. Le Calcul des Constructions Implicite [Miq01 ; Ber15] présente une forme radicale de sous-typage subsumptif où $T <: U$ dès que tout terme $\vdash t : T$ vérifie aussi $\vdash t : U$. Par opposition, l'approche *coercive* repose sur l'utilisation explicite de fonctions de coercion, éventuellement inférées durant une phase d'élaboration [Luo96 ; LS99 ; LL16].

Il est à noter que les approches subsumptives posent souvent des problèmes de décidabilité puisque moins d'information est conservée dans les termes. D'autre part, ces systèmes requièrent une conversion non typée ce qui nécessiterait une adaptation importante de la méthode à base de modèles syntaxiques. Nous prenons donc ici une approche coercive tout en notant que des modèles validant la transparence des coercions dans le cadre subsumptif pourraient permettre de réconcilier ces deux mondes.

Sous-typage via raffinement. Le langage Russel [Soz07] emploie du sous-typage par raffinement pour attacher des invariants à des programmes Coq et générer des conditions de vérification. Il est validé par une transformation de programme vers CIC enrichie de types sous-ensembles suffisamment extensionnels.

Le langage F* propose une approche similaire a la preuve de programme, mais à une échelle plus importante. Il implémente une théorie des types extensionnelle avec des types à raffinement, du sous-typage et du sous-effet. La métathéorie du langage à été étudiée sur des fragments parfois conséquents [Swa+16 ; Ahm+17 ; Ahm+18], mais une approche englobant toutes les différentes facettes du système manque encore à ce jour.

Sous-typage structurel. Le sous-typage structurel concerne l'aspect congruent des constructions du langage vis-à-vis de l'ordre de sous-typage. Un modèle où toutes les constructions sont monotones validerait donc immédiatement cette discipline de sous-typage. La monotonicité des paramètres des inductifs, c'est à dire le fait que $\mathrm{list}\, A <: \mathrm{list}\, B$ quand $A <: B$, est un cas particulier de sous-typage structurel. C'est d'ailleurs une extension régulièrement sollicitée par les utilisateurs de F* [Hri14], mais dont l'adoption se heurte à l'absence de fondations solides.

Une seconde application potentielle d'un modèle rigoureux serait la validation de la traduction de *fordisme*[1] pour Coq en présence de cumulativité. Celle-ci consiste à transformer un inductif à indices en un inductif ne prenant plus que des paramètres mais en contraignant chaque constructeur avec des preuves d'égalité supplémentaires (voir Fig. 1 pour l'exemple des vecteurs). En présence de cumulativité, cette traduction n'est valide que si l'on internalise la monotonie de l'égalité vis-à-vis de son premier paramètre.

Sous-typage polarisé Le sous-typage ne considère pas uniquement des constructions monotones : dans la plupart des disciplines de sous-typage pour des types simples, le type des fonctions

1. Ce nom semble être une allusion à la citation d'Henri Ford « Le client peut choisir la couleur de sa voiture, pourvu que ce soit noir. » [McB00]

est notoirement contravariant dans le domaine. Le sous-typage polarisé [Abe08a] ajoute des informations de monotonicité aux types de fonctions telles que leur caractère monotone/positif/-covariant, antitone/négatif/contravariant, invariant ou indépendant. Nuyts [Nuy15] esquisse un cadre extrêmement riche à base de modalité dépassant de loin le cadre du sous-typage et servant d'inspiration aux modèles que l'on présente ici.

Le modèle ensembliste de Timany et Sozeau [TS18] ne valide pas la contravariance des produits dépendants vis-à-vis de la cumulativité. Un modèle cumulatif de CIC admettant la contravariance des produits dépendants validerait l'η-réduction dans Coq en permettant de prouver la préservation du typage pour la réduction étendue avec $\lambda x : A.\, t\, x \rightsquigarrow t$.

3 Méthodologie proposée

Nous proposons une méthodologie employant un prototypage à l'aide de modèles de CIC appelés *syntaxiques* car ils vérifient certaines "bonnes propriétés" favorisant une implémentation rapide. Nous expliquons ensuite comment ces modèles peuvent nous aiguiller dans la preuve de propriétés fondamentales que notre langage à types dépendants et sous-typage devrait vérifier.

3.1 Modèles Syntaxiques

Un modèle d'un langage est une interprétation des termes et des types du langage dans un domaine cible fixé. Boulier, Pédrot et Tabareau qualifient de *syntaxique* un modèle de la théorie des types dont la cible est aussi une théorie des types et préservant la présentation syntaxique du langage et notamment la conversion entre termes. Afin de travailler avec plusieurs langages à types dépendants, nous adoptons une présentation basée sur les jugements suivants :

$\Gamma \vdash$	Γ est un contexte bien formé
$\Gamma \vdash A$	A est un type bien formé dans le contexte Γ
$\Gamma \vdash t : A$	t est un terme de type A dans le contexte Γ
$\Gamma \vdash A \equiv B$	A et B sont des types convertibles
$\Gamma \vdash t \equiv u : A$	t et u sont des termes convertibles de type A

Cette présentation des jugements est fondée sur les théories algébriques généralisées de [Car86] et correspond assez directement aux composantes d'une catégorie avec famille (CwF) [Dyb95 ; AK16]. Une théorie des types concrète étend ces CwF avec de nouveaux types, termes et équations. La théorie des types de Martin-Löf requiert des produits Π et sommes Σ dépendants, un type identité Id ainsi qu'un univers $\square$. Pour CIC, on considérera en plus une hiérarchie d'univers $\square_i : \square_{i+1}$, et une signature d'inductifs contenant en particulier $\mathbb{N}, \mathbb{B}, \texttt{list}$ ainsi que leurs éliminateurs.

Dans cette présentation, un modèle est la donnée d'une fonction $[\![-]\!]$ interprétant un type $\Gamma \vdash_{\texttt{src}} A$ de la source en un type $[\![\Gamma]\!] \vdash_{\texttt{tgt}} [\![A]\!]$ de la cible (puis étendu aux contextes), ainsi que d'une fonction $[-]$ interprétant un terme $\Gamma \vdash_{\texttt{src}} t : A$ de la source en un terme $[\![\Gamma]\!] \vdash_{\texttt{tgt}} [t] : [\![A]\!]$ de la cible. On dira que ce modèle est *syntaxique* lorsque cette traduction des contextes, types et termes peut se décrire effectivement sur la syntaxe, et telle que la conversion soit préservée. En pratique, on définit d'abord les deux fonctions de traduction $[\![-]\!]$ et $[-]$ sur la syntaxe des types et des termes, puis on montre par induction sur la dérivation de typage que

$$\Gamma \vdash_{\texttt{src}} t : A \qquad \Rightarrow \qquad [\![\Gamma]\!] \vdash_{\texttt{tgt}} [t] : [\![A]\!]$$

ce qui est grandement facilité si la conversion est préservée $\vdash_{\texttt{src}} A \equiv B \Rightarrow \vdash_{\texttt{tgt}} [\![A]\!] \equiv [\![B]\!]$.

Un certain nombre de modèles et traductions peuvent être raisonnablement considérés comme syntaxiques. Citons à titre d'exemples :

le modèle exceptionel [PT18 ; Péd+19] ou modèle des types pointés ;

les modèles de paramétricité [BL11 ; KL12 ; AGJ14 ; Bou18] équipant les termes avec une preuve de paramétricité ;

le modèle Setoid [Alt+19] équipant chaque type avec une relation d'équivalence ;

les modèles monadiques [PT17] interprétant un type comme une algèbre d'une monade.

La traduction de la théorie des types extensionnelle vers la théorie des types intensionnelle munie de l'extensionalité des fonctions (`funext`) et de l'irrélévance des preuves d'égalité (`UIP`) fournit un contre-exemple de modèle non-syntaxique [Hof95 ; WST19].

Alternativement, un modèle syntaxique peut être vu comme une phase de compilation préservant le typage. Cependant l'emploi du vocable *modèle* correspond mieux à notre programme : une traduction explique la théorie source via son encodage dans la théorie cible, mais surtout propose des extensions potentielles de la théorie source. Cette démarche n'est pas récente : la logique linéaire [Gir87] et le λ-calcul différentiel [ER03] proviennent de l'analyse des modèles d'espaces cohérents et de sémantiques quantitatives ; l'introduction de monades [Mog89] ou d'adjonctions [Lev03] pour manipuler des effets dérive directement de l'analyse de modèles catégoriques. L'aspect novateur est de contraindre les modèles à être présentés effectivement afin de guider plus précisément les extensions du langage source. Cette approche permet notamment :

— De valider rapidement certaines idées de nature sémantique sur l'objet d'étude, ici des ordres partiels ;

— De réutiliser les propriétés de l'assistant de preuve dans lequel on travaille (type LF), sans se préoccuper dans un premier temps des aspects purement syntaxiques et algorithmiques ;

— De restreindre l'espace de conception du langage source, en particulier celui des règles de réductions.

Plus particulièrement, les modèles syntaxiques peuvent aider à établir les propriétés métathéoriques clés suivantes :

la consistance relative vis-à-vis de la cible peut s'obtenir en étudiant la traduction du type vide $[\![\bot]\!]$ en supposant la théorie cible consistante ;

la normalisation forte en montrant que chaque réduction dans la source se traduit en une ou plusieurs réductions dans la cible et que celle-ci est fortement normalisante ;

la décidabilité de la conversion, du sous-typage et du typage en réemployant des résultats idoines dans la théorie cible ;

la préservation du typage en proposant des règles de réduction à ajouter à la théorie source validées par la traduction ;

la canonicité en montrant que toutes les conversions entre termes traduits dans la cible – vérifiant elle-même la canonicité – ont déjà lieu dans la source.

3.2 Vers une Implémentation de Référence

L'aspect *traduction de programme* des modèles syntaxiques est particulièrement attrayant pour la formalisation, puisque cette tâche peut être vue comme une phase de compilation. Dans le cadre de l'assistant de preuve Coq, le projet MetaCoq [Soz+20] permet par ailleurs de réifier et manipuler des termes de Gallina, le langage de programmation de Coq. Le processus envisagé est le suivant :

$$[M] := ([M]_0, [M]_1) \qquad [\![A]\!] := \Sigma(x : [A]_0).[A]_1 x$$

$$\begin{array}{rl} [x]_0 := \pi_1\ x & [x]_1 := \pi_2\ x \\ [\lambda(x:A).M]_0 := \lambda(x : [\![A]\!]).[M]_0 & [\lambda(x:A).M]_1 := \lambda(x : [\![A]\!]).[M]_1 \\ [M\ N]_0 := [M]_0\ [N] & [M\ N]_1 := [M]_1\ [N] \\ [\Pi(x:A).B]_0 := \Pi(x : [\![A]\!]).[B]_0 & [\Pi(x:A).B]_1 := \lambda f.\Pi(x : [\![A]\!]).[B]_1\ (f\ (\pi_1\ x)) \\ [\Box_i]_0 := \Box_i & [\Box_i]_1 := \lambda(A : \Box_i).A \to \Box_i \end{array}$$

FIGURE 2 – Transformation de paramétricité généreuse

1. implémenter les modèles considérés composante par composante dans l'assistant de preuve ;
2. étudier les éléments du modèle que l'on peut rajouter à un langage source, syntaxiquement proche de Gallina ;
3. implémenter la traduction de la source vers le modèle sur l'AST fourni par MetaCoq en réifiant les composantes ;
4. enfin, écrire des termes dans le langage source pour expérimenter avec les fonctionnalités.

Dans le cadre du sous-typage, la dernière étape pourrait être étendue avec une phase d'élaboration : MetaCoq serait employé pour récupérer des termes Coq sans coercion avant typage, et l'inférence de types serait instrumentée pour insérer les coercions à la volée. Cette infrastructure permettrait de réutiliser de nombreuses composantes de Coq dont notamment l'algorithme de conversion.

4 Ébauches de Modèles pour le Sous-Typage

Dans cette section, nous proposons deux ébauches de modèles inspirés par la littérature sur le sous-typage qui ont pour vocation de guider la résolution des problèmes présentés en Section 2.

4.1 Modèle de Sous-Ensembles

Notre premier modèle s'inspire des types à raffinement : on souhaite interpréter un type par une paire dépendante $\{C \mid P\}$ d'un type sous-jacent C et d'un prédicat P sur ce type. Ainsi, $\{C \mid P\}$ peut être vu comme un raffinement de C. La paramétricité *généreuse* de [Bou18], présentée en Fig. 2, interprète un type $A : \Box_i$ par une paire $[A] := ([A]_0, [A]_1)$ constituée d'un type $[A]_0 : \Box_i$ et d'une famille $[A]_1 : [A]_0 \to \Box_i$, et fournit donc un modèle proche. Dans l'esprit des systèmes de types à raffinement, on pourrait être naïvement amené à définir une relation $\subset$ sur les types par :

$$A \subset B \qquad \Longleftrightarrow \qquad \Sigma(h : [A]_0 = [B]_0) \quad \forall a : [A]_0, [A]_1\ a \to [B]_1(\texttt{cast}\ h\ a) \tag{1}$$

où $\texttt{cast} : \forall\{X\}\{P : X \to \Box\}\{x\ y : X\}, x = y \to P\,x \to P\,y$.

Cependant, quand $[B]_1$ n'est pas à valeur dans des propositions, par exemple $B = \Box_i$, cette définition accepte plusieurs coercions extensionnellement distinctes là où notre intuition de sous-typage en réclame une seule. Plusieurs réponses sont possibles face à cette difficulté. On pourrait être tentés de quotienter les familles de types afin qu'elles prennent leurs valeurs dans des propositions $P \mapsto \| P \|$. KELLER et LASSON [KL12] expliquent en détail en quoi cette option est incompatible avec les univers – on ne peut pas extraire un prédicat sur un type A à partir

de la proposition $\| A \to \square_i \|$ – et résolvent ce problème en modifiant la hiérarchie d'univers. À défaut, on pourrait essayer de quotienter la relation $\subset$, mais on perd alors l'interprétation des coercions comme fonctions.

Ne pouvant pas écraser brutalement cette structure, on peut essayer de l'apprivoiser : à la place d'une structure d'ordre, nous avons une structure plus riche de catégorie. Ce chemin mène tout droit vers l'algèbre supérieure, sujet riche et complexe que nous laisseront ici de côté (voir [LH11 ; Nuy15 ; RS17]).

Prenant en compte ces difficultés, on modifie donc le modèle afin que le prédicat propositionnel [2] équipant chaque type soit internalisé dans l'univers, c'est à dire $[\![\square_i]\!] \cong \Sigma(C : \square_i) C \to \mathrm{Prop}$ (voir Fig. 3). On équipe l'univers du prédicat $\lambda X.\top$ toujours valide. Un terme arbitraire $u : A$ est alors traduit en une paire $[u] := ([u]_0 : [A]_C, [u]_1 : [A]_P\, [u]_0)$ d'un élément du type sous-jacent $[A]_C$ et d'une preuve que le prédicat $[A]_P$ tient pour cet élément.

Dans la traduction des produits dépendants, le codomaine n'est pas raffiné dans le type sous-jacent ce qui permet au produit dépendant d'être covariant vis-à-vis du codomaine. Par contre, le domaine est nécessairement raffiné dans le type sous-jacent – comme dans la paramétricité généreuse – car on ne veut considérer que les éléments du domaine qui valident le prédicat considéré. Par conséquent ce modèle ne valide pas la contravariance dans le domaine.

Théorème 1. *La traduction sous-ensemble est un modèle syntaxique de CIC dans CIC :*

$$\begin{aligned} \Gamma \vdash_{CIC} t : A \quad &\Longrightarrow \quad [\![\Gamma]\!] \vdash_{CIC} [t] : [\![A]\!] \\ \Gamma \vdash_{CIC} A \equiv B \quad &\Longrightarrow \quad [\![\Gamma]\!] \vdash_{CIC} [\![A]\!] \equiv [\![B]\!] \end{aligned}$$

Ce théorème fournit la première étape : on souhaite ensuite étendre la théorie source, c'est à dire CIC, avec un jugement de sous-typage décidable et des types à raffinement. Ce jugement de sous-typage se traduit par la formule (1). Pour un exemple concret de type à raffinement, on peut étendre la source avec un type primitif Even, traduit par $[\mathsf{Even}] := (([\mathbb{N}]_C, \lambda n.\ \texttt{is\_even}\ n), \sharp\sharp)$, ainsi qu'une relation $\Gamma \vdash \mathsf{Even} <: \mathbb{N}$ puisque cette relation est validée par le modèle. Cette théorie étendue serait toujours consistante, puisque $[\![\bot]\!]$ est vide et on conjecture que l'on peut préserver la canonicité tout en obtenant des règles de sous-typage intéressantes. Par exemple, la traduction des listes montre que l'on pourrait valider la monotonicité de `list` vis-à-vis du sous-typage, fournissant ainsi une réponse concrète à [Hri14].

4.2 Modèle de Préordres

Le modèle de préordre équipe chaque type avec une relation de préordre. Il permet d'internaliser la notion de constructions monotones et antitones en théorie des types. Formellement, on interprète un type A par un préordre, c'est à dire un quadruplet $[\![A]\!] = (|A|, \leq^A, \texttt{refl}_A, \texttt{trans}_A)$ constitué d'un type sous-jacent $|A|$, équipé d'une relation $\leq^A : |A| \to |A| \to \mathrm{Prop}$ réflexive et transitive

$$\texttt{refl}_A : \forall (a : |A|), a \leq^A a \qquad \texttt{trans}_A : a_0 \leq^A a_1 \wedge a_1 \leq^A a_2 \to a_0 \leq^A a_2$$

Une famille de type $x : A \vdash B$ s'interprète comme un foncteur de A vu comme une catégorie vers la catégorie des préordres et des applications monotones.

Étant donnés un préordre A et une famille $x : A \vdash B$, on peut construire deux produits dépendants, le produit monotone $\Pi^{\mathrm{mon}}(x : A).B$ et le produit antitone $\Pi^{\mathrm{anti}}(x : A).B$ qui

2. Dans ce contexte, une proposition $p : \mathrm{Prop}$ est considérée comme un type avec au plus un habitant, un sous-singleton. Afin d'obtenir des modèles syntaxiques, on demandera parfois qu'il y ait au plus un habitant *définitionnellement*, c'est à dire que la proposition soit stricte [Gil+19].

$$
\begin{array}{rlrl}
[\Pi(a:A).\ B]_C &:= \Pi(a:[\![A]\!]).\ \pi_1\ [B]_0 & [\Pi(a:A).\ B]_P &:= \lambda f.\ \Pi(a:[\![A]\!]).\ \pi_2\ [B]_0\ (f\ a) \\
[\square_i]_C &:= \Sigma(X:\square_i).\ X \to \mathrm{Prop} & [\square_i]_P &:= \lambda X.\ \top \\
[\mathbb{N}]_C &:= \mathbb{N} & [\mathbb{N}]_P &:= \ \ldots\ \equiv \lambda X.\ \top
\end{array}
$$

$$
\begin{array}{rlrl}
[\Pi(a:A).\ B]_0 &:= ([\Pi(a:A).\ B]_C, [\Pi(a:A).\ B]_P) & [\Pi(a:A).\ B]_1 &:= \not\Vdash \\
[\square_i]_0 &:= ([\square_i]_C, [\square_i]_P) & [\square_i]_1 &:= \not\Vdash \\
[\mathbb{N}]_0 &:= ([\mathbb{N}]_C, [\mathbb{N}]_P) & [\mathbb{N}]_1 &:= \not\Vdash \\
[M\ N]_0 &:= [M]_0\ [N] & [M\ N]_1 &:= [M]_1\ [N] \\
[\lambda(a:A).\ M]_0 &:= \lambda(a:[\![A]\!]).\ [M]_0 & [\lambda(a:A).\ M]_1 &:= \lambda(a:[\![A]\!]).\ [M]_1 \\
[x]_0 &:= \pi_1\ x & [x]_1 &:= \pi_2\ x
\end{array}
$$

FIGURE 3 – Traduction sous-ensemble

contiennent les fonctions $\Pi(x : A).B$ respectivement monotones et antitones. Équipés de la relation induite par l'ordre points-à-points, ces types forment des préordres. Ces constructions s'étendent fonctoriellement : étant données une application monotone $f_A : A_1 \to A_0$ et une transformation naturelle $f_B : B_0 \circ f_A \dot{\to} B_1$, on construit des applications monotones $\Pi^\star\,(x : A_0).B_0 \to \Pi^\star(x : A_1).B_1$. Nous laissons le détail de ces constructions à l'ébauche de formalisation accompagnant cette note [LM20], et notons que cette construction est contravariante vis-à-vis du domaine et covariante vis-à-vis du codomaine.

Les types inductifs [3] s'interprètent munis de leur relation de paramétricité [BL11]. Sur des inductifs sans paramètre tels que $\mathbb{N}$ ou $\mathbb{B}$, cela revient à considérer l'ordre discret, c'est à dire l'égalité intensionnelle. Sur des listes list A, l'ordre de A est étendu composantes à composantes entre listes de même taille. Sur des sommes dépendantes $\Sigma(x : A).B$, on obtient naturellement l'ordre lexicographique.

La traduction doit équiper les univers $[\square_i] := \mathbb{U}_i$ avec un ordre de manière compatible avec les familles de types : si $X \leq^{\mathbb{U}_i} Y$ on doit pouvoir construire une fonction monotone *canonique* de X dans Y. En particulier, on ne peut pas refléter toutes les fonctions (monotones) entre les types de notre univers : si $\mathbb{B}$ représente les booléens munis de l'ordre discret, il y a 4 applications $\mathbb{B} \to \mathbb{B}$ dont deux bijections, l'identité et la négation, que l'on ne peut pas distinguer à priori dans CIC – cela fournirait une réfutation de l'univalence [Uni13]. Il est donc nécessaire de rajouter une composante intensionnelle à nos types afin de pouvoir trier entre les "bons" morphismes – qui seront capturés par l'ordre sur l'univers – et les autres.

Techniquement, il s'agit d'interpréter un type par un *code*, un élément d'un inductif conservant les "plans de constructions" du type. Les codes sont ensuite interprétés comme des préordres à l'aide d'une fonction de décodage, voir Fig. 4. De manière similaire, une relation est définie inductivement sur la structure des codes puis décodée en un morphisme entre le décodage respectif des codes.

Extension du sous-typage Si le modèle permet de parler de monotonie, les types introduits jusqu'ici ne font que propager un ordre. En enrichissant l'univers de types avec des relations particulières, on peut tenter de modéliser plusieurs disciplines de sous-typage de la Section 2 :

Cumulativité explicite on ajoute des relations $\mathfrak{u}_i \leq^{\mathbb{U}} \mathfrak{u}_j$ dès que $i \leq j$, et on interprète ces relations par l'application monotone plongeant $\mathbb{U}_i$ dans $\mathbb{U}_j$.

3. Dans un premier temps, on ne considère pas les inductifs avec indices.

Univers $\mathbb{U}_i$ et fonction de décodage $\mathrm{El} : \mathbb{U}_i \to \Box$

$$\frac{}{\mathfrak{nat} \in \mathbb{U}} \qquad \frac{}{\mathfrak{bool} \in \mathbb{U}} \qquad \frac{A \in \mathbb{U}}{\mathfrak{list}\, A \in \mathbb{U}} \qquad \frac{i < j}{\mathfrak{u}_i \in \mathbb{U}_j}$$

$$\frac{A \in \mathbb{U} \qquad B \in \Pi^{\mathrm{mon}}(a : \mathrm{El}\, A)\, \mathbb{U}}{\mathfrak{s}\, A\, B \in \mathbb{U}} \qquad \frac{\star \in \{\mathtt{mon}, \mathtt{anti}\} \qquad A \in \mathbb{U}_i \qquad B \in \Pi^{\mathrm{mon}}(a : \mathrm{El}\, A)\, \mathbb{U}}{\pi\, \star\, A\, B \in \mathbb{U}}$$

$$\begin{aligned} \mathrm{El}\, \mathfrak{nat} &= \mathbb{N} & \mathrm{El}\, \mathfrak{bool} &= \mathbb{B} \\ \mathrm{El}\, (\mathfrak{list}\, A) &= \mathrm{list}\, (\mathrm{El}\, A) & \mathrm{El}\, \mathfrak{u}_k &= \mathbb{U}_k \\ \mathrm{El}\, (\mathfrak{s}\, A\, B) &= \Sigma(x : A).B & \mathrm{El}\, (\pi\, \star\, A\, B) &= \Pi^{\star}(a : \mathrm{El}\, A)\ \mathrm{El}(B\, a) \end{aligned}$$

Ordre $\leq^{\mathbb{U}}$ sur les codes

$$\frac{}{\mathfrak{nat} \leq^{\mathbb{U}} \mathfrak{nat}}\ \mathfrak{nat}\text{-}\leq^{\mathbb{U}} \qquad \frac{}{\mathfrak{u}_k \leq^{\mathbb{U}} \mathfrak{u}_k}\ \mathfrak{u}\text{-}\leq^{\mathbb{U}} \qquad \frac{A \leq^{\mathbb{U}} A'}{\mathfrak{list}\, A \leq^{\mathbb{U}} \mathfrak{list}\, A'}\ \mathfrak{list}\text{-}\leq^{\mathbb{U}}$$

$$\frac{A_\varepsilon : A' \leq^{\mathbb{U}} A \qquad a : \mathrm{El}\, A, a' : \mathrm{El}\, A', a_\epsilon : \mathrm{El}_\varepsilon\, A_\varepsilon\ a \leq^{A'} a' \vdash B\, a \leq^{\mathbb{U}} B'\, a'}{\pi\, \star\, A\, B \leq^{\mathbb{U}} \pi\, \star\, A'\, B'}\ \pi\text{-}\leq^{\mathbb{U}}$$

FIGURE 4 – Univers de code, relation d'ordre et décodage

Sous-typage en largeur étant fixé un type $\mathcal{L}$ d'étiquettes muni d'une égalité décidable, on étend l'univers avec des types sommes étiquetés et des types enregistrements avec champs étiquetés. La relation de sous-typage entre ces types est induite par l'inclusion des ensembles d'étiquettes combinées au sous-typage structurel, par exemple :

$$A <: B \qquad \Longrightarrow \qquad \{\ \ell_1 : A;\ \ell_0 : X\ \} <: \{\ \ell_1 : B\ \}$$

Comme pour le modèle des sous-ensembles présenté en Section 4.1, les éléments ci-dessus permettent d'assembler un modèle syntaxique de CIC. Cependant, afin de tirer pleinement parti de l'aspect polarisé du modèle de préordre, la théorie source devrait être largement étendue. NUYTS [Nuy15] propose d'employer 3 modalités `discr`/`op`/`codiscr` pour manipuler l'ordre sur les types, néanmoins une expérience préliminaire en Agda laisse penser que la théorie résultante semble difficile à utiliser en l'état et sera au cœur des recherches ultérieures sur ce sujet.

5 Conclusion

Nous avons présenté et motivé une approche à base de modèles syntaxiques pour étudier plusieurs disciplines de sous-typage en présence de types dépendants. Des formalisation en Coq et en Agda des deux modèles ébauchés dans la section précédente sont en cours de développement, première étape concrète de cette étude. Nous espérons que notre démarche pourra aider à consolider la littérature déjà conséquente mais morcelée du sous-typage en présence de types dépendants, et plus généralement qu'elle montrera l'intérêt d'employer des assistants à la preuve tout au long de la conception de langages de programmation.

Références

[Abe08a] Andreas Abel. « Polarised subtyping for sized types ». In : *Math. Struct. Comput. Sci.* 18.5 (2008), p. 797-822. DOI : 10.1017/S0960129508006853. URL : https://doi.org/10.1017/S0960129508006853.

[Abe08b] Andreas Abel. « Semi-continuous Sized Types and Termination ». In : *Logical Methods in Computer Science* Volume 4, Issue 2 (avr. 2008). DOI : 10.2168/LMCS-4(2:3)2008. URL : https://lmcs.episciences.org/1236.

[AGJ14] Robert Atkey, Neil Ghani et Patricia Johann. « A relationally parametric model of dependent type theory ». In : *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*. 2014, p. 503-516. DOI : 10.1145/2535838.2535852. URL : https://doi.org/10.1145/2535838.2535852.

[Ahm+17] Danel Ahman et al. « Dijkstra Monads for Free ». In : *44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*. ACM, jan. 2017, p. 515-529. DOI : 10.1145/3009837.3009878. URL : https://www.fstar-lang.org/papers/dm4free/.

[Ahm+18] Danel Ahman et al. « Recalling a Witness : Foundations and Applications of Monotonic State ». In : *PACMPL* 2.POPL (jan. 2018), 65 :1-65 :30. URL : https://arxiv.org/abs/1707.02466.

[AK16] Thorsten Altenkirch et Ambrus Kaposi. « Type theory in type theory using quotient inductive types ». In : *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*. 2016, p. 18-29. DOI : 10.1145/2837614.2837638. URL : https://doi.org/10.1145/2837614.2837638.

[Alt+19] Thorsten Altenkirch et al. « Setoid Type Theory - A Syntactic Translation ». In : *Mathematics of Program Construction - 13th International Conference, MPC 2019, Porto, Portugal, October 7-9, 2019, Proceedings*. 2019, p. 155-196. DOI : 10.1007/978-3-030-33636-3_7. URL : https://doi.org/10.1007/978-3-030-33636-3_7.

[AP13] Andreas M. Abel et Brigitte Pientka. « Wellfounded Recursion with Copatterns : A Unified Approach to Termination and Productivity ». In : *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*. ICFP '13. Boston, Massachusetts, USA : Association for Computing Machinery, 2013, 185–196. ISBN : 9781450323260. DOI : 10.1145/2500365.2500591. URL : https://doi.org/10.1145/2500365.2500591.

[Ass14] Ali Assaf. « A Calculus of Constructions with Explicit Subtyping ». In : *20th International Conference on Types for Proofs and Programs, TYPES 2014, May 12-15, 2014, Paris, France*. 2014, p. 27-46. DOI : 10.4230/LIPIcs.TYPES.2014.27. URL : https://doi.org/10.4230/LIPIcs.TYPES.2014.27.

[Ber15] Bruno Bernardo. « Un Calcul des Constructions implicite avec sommes dépendantes et à inférence de type décidable. » Version soutenance. Theses. École polytechnique, oct. 2015. URL : https://hal.inria.fr/tel-01197380.

[BL11] Jean-Philippe Bernardy et Marc Lasson. « Realizability and Parametricity in Pure Type Systems ». In : *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*. 2011, p. 108-122. DOI : 10.1007/978-3-642-19805-2_8. URL : https://doi.org/10.1007/978-3-642-19805-2_8.

[Bou18] Simon Boulier. « Extending type theory with syntactic models ». Theses. Ecole nationale supérieure Mines-Télécom Atlantique, nov. 2018. URL : https://tel.archives-ouvertes.fr/tel-02007839.

[BPT17a] Simon Boulier, Pierre-Marie Pédrot et Nicolas Tabareau. « Modèles de la théorie des types donnés par traduction de programme ». In : *28ièmes Journées Francophones des Langages Applicatifs*. Gourette, France, jan. 2017. URL : https://hal.archives-ouvertes.fr/hal-01503089.

[BPT17b] Simon Boulier, Pierre-Marie Pédrot et Nicolas Tabareau. « The next 700 syntactical models of type theory ». In : *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*. 2017, p. 182-194. DOI : 10.1145/3018610.3018620. URL : https://doi.org/10.1145/3018610.3018620.

[Car86] John Cartmell. « Generalised algebraic theories and contextual categories ». In : *Annals of Pure and Applied Logic* 32 (1986), p. 209 -243. ISSN : 0168-0072. DOI : https://doi.org/10.1016/0168-0072(86)90053-9. URL : http://www.sciencedirect.com/science/article/pii/0168007286900539.

[Dyb95] Peter Dybjer. « Internal Type Theory ». In : *Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5-8, 1995, Selected Papers*. 1995, p. 120-134. DOI : 10.1007/3-540-61780-9_66. URL : https://doi.org/10.1007/3-540-61780-9_66.

[ER03] Thomas Ehrhard et Laurent Regnier. « The differential lambda-calculus ». In : *Theor. Comput. Sci.* 309.1-3 (2003), p. 1-41. DOI : 10.1016/S0304-3975(03)00392-X. URL : https://doi.org/10.1016/S0304-3975(03)00392-X.

[Gil+19] Gaëtan Gilbert et al. « Definitional proof-irrelevance without K ». In : *Proc. ACM Program. Lang.* 3.POPL (2019), 3 :1-3 :28. DOI : 10.1145/3290316. URL : https://doi.org/10.1145/3290316.

[Gir87] Jean-Yves Girard. « Linear logic ». In : *Theoretical Computer Science* 50.1 (1987), p. 1 -101. ISSN : 0304-3975. DOI : https://doi.org/10.1016/0304-3975(87)90045-4. URL : http://www.sciencedirect.com/science/article/pii/0304397587900454.

[Hof95] Martin Hofmann. « Conservativity of Equality Reflection over Intensional Type Theory ». In : *Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5-8, 1995, Selected Papers*. 1995, p. 153-164. DOI : 10.1007/3-540-61780-9_68. URL : https://doi.org/10.1007/3-540-61780-9_68.

[Hri14] Catalin Hritcu. *Issue #65 on Fstar's bug tracker "Polarities : subtyping for datatypes"*. https://github.com/FStarLang/FStar/issues/65. 2014. URL : https://github.com/FStarLang/FStar/issues/65.

[KL12] Chantal Keller et Marc Lasson. « Parametricity in an Impredicative Sort ». In : *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*. 2012, p. 381-395. DOI : 10.4230/LIPIcs.CSL.2012.381. URL : https://doi.org/10.4230/LIPIcs.CSL.2012.381.

[LA08] Zhaohui Luo et Robin Adams. « Structural subtyping for inductive types with functorial equality rules ». In : *Math. Struct. Comput. Sci.* 18.5 (2008), p. 931-972. DOI : 10.1017/S0960129508006956. URL : https://doi.org/10.1017/S0960129508006956.

[Lev03] Paul Blain Levy. *Call-By-Push-Value*. en. Dordrecht : Springer Netherlands, 2003. ISBN : 978-94-010-3752-5 978-94-007-0954-6. DOI : 10.1007/978-94-007-0954-6. URL : http://link.springer.com/10.1007/978-94-007-0954-6 (visité le 15/07/2020).

[LH11] Daniel R. Licata et Robert Harper. « 2-Dimensional Directed Type Theory ». In : *Twenty-seventh Conference on the Mathematical Foundations of Programming Semantics, MFPS 2011, Pittsburgh, PA, USA, May 25-28, 2011*. 2011, p. 263-289. DOI : 10.1016/j.entcs.2011.09.026. URL : https://doi.org/10.1016/j.entcs.2011.09.026.

[LL16] Georgiana E. Lungu et Zhaohui Luo. « On Subtyping in Type Theories with Canonical Objects ». In : *22nd International Conference on Types for Proofs and Programs, TYPES 2016, May 23-26, 2016, Novi Sad, Serbia*. 2016, 13 :1-13 :31. DOI : 10.4230/LIPIcs.TYPES.2016.13. URL : https://doi.org/10.4230/LIPIcs.TYPES.2016.13.

[LM20] Théo Laurent et Kenji Maillard. *Développement Coq accompagnant cet article*. 2020. URL : https://gitlab.inria.fr/kmaillar/subcic.

[LS99] Zhaohui Luo et Sergei Soloviev. « Dependent Coercions ». In : *Conference on Category Theory and Computer Science, CTCS 1999, Edinburgh, UK, December 10-12, 1999*. 1999, p. 152-168. DOI : 10.1016/S1571-0661(05)80314-7. URL : https://doi.org/10.1016/S1571-0661(05)80314-7.

[Luo88] Zhaohui Luo. *CCômega_subset and its metatheory*. https://www.lfcs.inf.ed.ac.uk/reports/88/ECS-LFCS-88-58/. 1988.

[Luo96] Zhaohui Luo. « Coercive Subtyping in Type Theory ». In : *Computer Science Logic, 10th International Workshop, CSL '96, Annual Conference of the EACSL, Utrecht, The Netherlands, September 21-27, 1996, Selected Papers*. 1996, p. 276-296. DOI : 10.1007/3-540-63172-0_45. URL : https://doi.org/10.1007/3-540-63172-0_45.

[McB00] Conor McBride. « Dependently typed functional programs and their proofs ». Thèse de doct. University of Edinburgh, UK, 2000. URL : http://hdl.handle.net/1842/374.

[Miq01] Alexandre Miquel. « Le Calcul des Constructions implicite : syntaxe et sémantique ». Theses. Université Paris 7, déc. 2001. URL : https://www.fing.edu.uy/~amiquel/publis/these.pdf.

[Mog89] Eugenio Moggi. « Computational Lambda-Calculus and Monads ». In : *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*. 1989, p. 14-23. DOI : 10.1109/LICS.1989.39155. URL : https://doi.org/10.1109/LICS.1989.39155.

[Nuy15] Andreas Nuyts. « Toward a Directed Homotopy Type Theory based on 4 Kinds of Variance ». Mém. de mast. Katholieke Universiteit Leuven, 2015. URL : https://anuyts.github.io/files/mathesis.pdf.

[Péd+19] Pierre-Marie Pédrot et al. « A reasonably exceptional type theory ». In : *Proc. ACM Program. Lang.* 3.ICFP (2019), 108 :1-108 :29. DOI : 10.1145/3341712. URL : https://doi.org/10.1145/3341712.

[PT17] Pierre-Marie Pédrot et Nicolas Tabareau. « An effectful way to eliminate addiction to dependence ». In : *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. 2017, p. 1-12. DOI : 10.1109/LICS.2017.8005113. URL : https://doi.org/10.1109/LICS.2017.8005113.

[PT18] Pierre-Marie Pédrot et Nicolas Tabareau. « Failure is Not an Option - An Exceptional Type Theory ». In : *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*. 2018, p. 245-271. DOI : 10.1007/978-3-319-89884-1_9. URL : https://doi.org/10.1007/978-3-319-89884-1_9.

[RS17] Emily Riehl et Michael Shulman. « A type theory for synthetic ∞-categories ». en. In : *Higher Structures* 1.1 (2017), p. 78. URL : https://journals.mq.edu.au/index.php/higher_structures/article/view/36.

[Soz07] Matthieu Sozeau. « Subset coercions in Coq ». In : *Types for Proofs and Programs*. T. 4502. 2007, p. 237-252.

[Soz+20] Matthieu Sozeau et al. « Coq Coq correct ! verification of type checking and erasure for Coq, in Coq ». In : *Proc. ACM Program. Lang.* 4.POPL (2020), 8 :1-8 :28. DOI : 10.1145/3371076. URL : https://doi.org/10.1145/3371076.

[Swa+16] Nikhil Swamy et al. « Dependent Types and Multi-Monadic Effects in F* ». In : *43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. ACM, jan. 2016, p. 256-270. ISBN : 978-1-4503-3549-2. URL : https://www.fstar-lang.org/papers/mumon/.

[TS18] Amin Timany et Matthieu Sozeau. « Cumulative Inductive Types In Coq ». In : *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*. 2018, 29 :1-29 :16. DOI : 10.4230/LIPIcs.FSCD.2018.29. URL : https://doi.org/10.4230/LIPIcs.FSCD.2018.29.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory : Univalent Foundations of Mathematics*. Institute for Advanced Study : https://homotopytypetheory.org/book, 2013.

[WST19] Théo Winterhalter, Matthieu Sozeau et Nicolas Tabareau. « Eliminating reflection from type theory ». In : *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*. 2019, p. 91-103. DOI : 10.1145/3293880.3294095. URL : https://doi.org/10.1145/3293880.3294095.