

4 Résumé de la thèse (2 pages maxi)

Contexte Dans les dernières années, XML s'est imposé comme un format standard et générique pour représenter, stocker, échanger des données semi-structurées. En dehors des outils génériques, les applications travaillent avec des classes bien définies de documents XML, souvent spécifiées d'une manière formelle par soucis d'interopérabilité ou de documentation, au moyen de DTD ou de XML-Schema, que l'on peut voir comme des définitions de types.

Le développement d'applications XML avec des langages de programmations généralistes est souvent mal aisé. La raison principale est que le modèle de données XML (les documents et leurs types) ne se laisse pas facilement représenter de manière fidèle, sans introduire de structure superflue et sans oublier d'information, dans ces langages. De plus, des opérations simples (navigation, requête), ne s'expriment pas de manière idiomatique dans ces langages. Il est bien entendu possible d'utiliser des outils externes et/ou des bibliothèques spécialisées, mais on perd alors la souplesse des types XML, et les opérations deviennent lourdes et perdent en sûreté (par exemple, les bibliothèques génériques qui implémentent XPath « oublient » les types XML).

Pour faciliter le développement d'applications XML, il est naturel de chercher à prendre en compte de manière native dans le langage de programmation la notion de type XML. Les bénéfices attendus sont une plus grande expressivité et une plus grande sûreté des applications. La thèse d'Haruo Hosoya, qui a débouché sur la définition du langage XDuce, est un des travaux fondateurs dans cette ligne de recherche. XDuce propose de considérer les documents XML comme des termes (arbres), manipulés de manière purement fonctionnelle. Les types dans XDuce constituent une généralisation des DTD (ce sont des automates d'arbres), et ils donnent naturellement naissance à une relation de sous-typage ensembliste, utilisée librement dans les programmes (sous-entendu implicite). Des propriétés de clôture de l'algèbre de types permettent de typer d'une manière très précise des opérations sur les documents XML, telle une puissante opération de filtrage par motifs expressions régulières (*pattern-matching*). XDuce a eu une grande influence dans la communauté de recherche autour des langages pour XML, et en dehors, par exemple dans la définition du langage de requêtes XQuery, ou de Relax-NG, une alternative à XML-Schema.

Aperçu de la thèse Ma thèse s'inscrit dans la continuité des travaux sur XDuce. XDuce partage de nombreux traits avec des langages fonctionnels : manipulation de termes sans effet de bords, filtrage, fonctions récursives. Pourtant, ce langage ne dispose pas de fonctions de première classe. Le point de départ de mon travail de DEA, qui a débouché sur la thèse, a été de combler cette lacune. L'ajout d'un type flèche à l'algèbre de types, tout en préservant ses propriétés de clôture et une définition ensembliste du sous-typage, n'a pas été chose aisée. Il a fallu développer un cadre théorique pour traiter d'une algèbre de types avec constructeurs (produits, flèches), combinaisons booléennes (union, intersection, complémentaire), types récursifs. La thèse présente plusieurs variantes de ce cadre. En particulier, il est apparu que la prise en compte de fonctions surchargées (avec plusieurs types flèches) venait naturellement. Cela a donné un éclairage nouveau sur des travaux antérieurs portant sur des calculs

avec fonctions surchargées, tel le $\lambda^{\&}$ calcul, en donnant une justification sémantique à des relations de sous-typage jusqu'alors définies de manière axiomatique. Évidemment, ces développements sémantiques s'éloignent un peu du cadre des langages de programmation pour XML : ils ont été guidés par des considérations théoriques, des soucis esthétiques. Néanmoins, ils fournissent un cadre solide pour comprendre comment intégrer les caractéristiques de XDuce dans un vrai langage fonctionnel d'ordre supérieur. Les fonctions surchargées trouvent d'ailleurs une utilisation naturelle dans le cadre de manipulations de documents XML.

À coté de ces travaux de sémantique et de typage, je me suis intéressé à la conception et à l'implémentation effective de CDuce, une extension de XDuce avec ordre supérieur fonctionnel et fonctions surchargées. La contribution principale de la thèse dans ce domaine porte sur la compilation efficace du filtrage par motifs ; j'ai développé un formalisme pour exprimer diverses stratégies correctes de compilation, et j'en ai étudié plus particulièrement une qui utilise de manière maximale des informations fournies par le système de types pour produire des automates efficaces. Les autres contributions portent sur divers aspects de l'implémentation : cadre catégorique pour rendre compte de plusieurs stratégies de partage dans l'algèbre de types ; évaluation efficace de prédicats inductifs ou coinductifs au moyen d'un solveur léger de contraintes booléennes (utilisé en particulier pour calculer efficacement la relation de sous-typage) ; techniques de représentation des valeurs XML à l'exécution.

Survol des chapitres La première partie de la thèse présente les fondations théoriques et sémantiques de CDuce.

Chap. 2,3 Cadre catégorique pour définir l'algèbre de types, ses stratégies de partage.

Chap. 4 Définition de la relation de sous-typage, via une notion abstraite de modèle ensembliste.

Chap. 5 Formalisation du noyau fonctionnel de CDuce, sous la forme d'un λ -calcul typé, preuve de sûreté, inférence de types.

Chap. 6 Opération de filtrage (généralisation de celle de XDuce, fermeture d'un problème ouvert sur l'inférence exacte).

La deuxième partie s'intéresse aux aspects algorithmiques.

Chap. 7 Stratégies d'évaluation de prédicats (co)inductifs, application au calcul du sous-typage.

Chap. 8 Compilation du filtrage.

Enfin, la troisième partie présente le langage CDuce lui-même et son implémentation.

Chap. 9 Ajout d'enregistrements extensibles au formalisme, pour rendre compte des attributs XML.

Chap. 10 Présentation du langage : types de base, opérateurs, sucre syntaxique pour XML, traits impératifs.

Chap. 11 Techniques d'implémentation.