Prévenir les erreurs techniques des analyses dans les data lakes avec la théorie des types

Alexis Guyot, Éric Leclercq, Annabelle Gillet, Nadine Cullot

Laboratoire d'Informatique de Bourgogne (LIB), Université de Bourgogne
9 avenue Alain Savary, F-21078 Dijon CEDEX, France
{prenom}.{nom}@u-bourgogne.fr

REFERENCE DE L'ARTICLE INTERNATIONAL Cet article est une synthèse de l'article :
Alexis Guyot, Éric Leclercq, Annabelle Gillet, Nadine Cullot:
Preventing Technical Errors in Data Lake Analyses with Type Theory. DaWaK 2023: 18-24.

Les *data lakes* sont des plateformes dédiées à l'analyse des données massives (Hai *et al.*, 2023). Ils fournissent un ensemble d'opérateurs pour explorer, transformer, enrichir et analyser les données à la demande. Ces opérateurs proviennent de différents outils spécialisés. Par exemple, des opérateurs de transformation peuvent provenir de SparkSQL ou de Pandas, des opérateurs d'analyse de TensorFlow ou de NetworkX.

Les utilisateurs des *data lakes* mettent en œuvre leurs analyses en composant ces différents opérateurs. On qualifie de telles compositions de *workflows* d'analyse. Les *workflows* d'analyse dans les *data lakes* sont généralement hybrides. Les opérateurs composés reposent sur différents cadres théoriques comme l'algèbre relationnelle, l'algèbre linéaire ou la théorie des graphes. Ils implémentent différents paradigmes de programmation comme l'orienté objet ou le fonctionnel. Ils peuvent s'exécuter dans différents environnements d'exécution comme un CPU, un GPU ou un DPU.

L'hybridité des *workflows* d'analyse est source d'erreurs techniques pouvant nuire à leur exécution. En effet, de nombreuses transformations intermédiaires sont nécessaires pour composer certains opérateurs : filtrage, jointure, changement de modèle, etc. Ces transformations s'appliquent sur différents niveaux d'abstraction, notamment sur les données elles-mêmes, sur leur schéma et sur leur modèle. Elles peuvent introduire des incohérences sur chacun de ces niveaux. Par exemple, une transformation peut retirer un attribut nécessaire à l'exécution d'un opérateur suivant du *workflow*. Une autre peut introduire un attribut dont le type n'est pas supporté par le modèle de données utilisé par un opérateur suivant.

Notre article présente une approche pour prévenir les erreurs techniques dans les *workflows* d'analyse des *data lakes* (Guyot *et al.*, 2023). Nous nous intéressons particulièrement aux erreurs techniques causées par des incohérences au niveau du schéma et du modèle. Notre approche transforme les erreurs techniques en erreurs

de types, de sorte à pouvoir les prévenir dès la compilation. Contrairement aux approches existantes, elle permet une représentation unifiée de différents modèles, de différents niveaux d'abstraction et des liens qui les unissent.

Pour cela, nous proposons un cadre formel basé sur la théorie des types (Martin-Löf et Sambin, 1984). Ce cadre définit un ensemble de types pour représenter les schémas et les modèles des entrées et sorties des opérateurs des *workflows*. La théorie des types est un formalisme constructif permettant la définition de types à partir de règles de construction. Elle fournit un ensemble de types primitifs : entiers (*Integer*), chaînes de caractères (*String*), etc. Elle fournit également un ensemble de constructeurs permettant la définition de types composites, notamment : des produits de types, notés *T1* x *T2* ; des types génériques, notés *T1[T2]* ; ou des types dépendants, notés $\Pi_{(x:T2)}T1(x)$. Un produit de types est un type dont les valeurs sont des paires de valeurs d'autres types, comme le type *Integer* x *String* de la valeur *(1, abc)*. Un type générique est un type paramétré par un autre type, comme le type *Array[Integer]* de la valeur *[1,2]*. Un type dépendant est un type paramétré par une valeur d'un autre type, comme le type $\Pi_{(3:Integer)}Array(3)$ de la valeur *[a,1,true]*.

Dans notre cadre formel, les schémas des données sont représentés par des produits de types dépendants-génériques, et les modèles par des types génériques. Ainsi, une relation Personne composée de deux attributs Nom et Age de types *String* et *Integer* peut être associée au type $Relation[\Pi_{(Nom:String)}Attribut[String](Nom)$ x $\Pi_{(Age:String)}Attribut[Integer](Age)]$. Les valeurs de ce type sont des tuples de chaînes de caractères et d'entiers comme *(guyot, 24)*. Les liens entre les deux niveaux d'abstraction sont exprimés au travers de règles de construction. Notre cadre — combiné avec les mécanismes de preuve de la théorie des types — permet de formellement prouver l'absence d'erreur dans une composition d'opérateurs.

Nous avons réalisé une implémentation des différentes constructions de notre cadre formel en Scala[1]. Celle-ci utilise le compilateur pour techniquement vérifier l'absence d'erreur. Elle repose sur les mécanismes de typage avancés du langage — notamment les implicites qui permettent de modifier le comportement du compilateur — et sur la bibliothèque Shapeless[2]. L'implémentation peut être utilisée pour spécifier des langages de domaine (*Domain Specific Language* ou DSL) permettant la définition de *workflows* d'analyse sûrs dans les *data lakes*.

Bibliographie

Guyot A., Leclercq E., Gillet A., Cullot N. (2023). Preventing Technical Errors in Data Lake Analyses with Type Theory, *International Conference on Big Data Analytics and Knowledge Discovery, DaWaK 2023 (pp. 18-24)*, Springer, Penang, Malaysia.

Hai R., Koutras C., Quix C., Jarke M. (2023). Data Lakes : A Survey of Functions and Systems, *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, n°12, p. 12571-12590.

Martin-Löf P., Sambin G. (1984). *Intuitionistic type theory*, Bibliopolis, Naples.

[1] https://github.com/AlexisGuyot/type_safe_compo
[2] https://github.com/milessabin/shapeless