

Rapport TAS - Fiche de lecture

Article: *Compact Bit Encoding Schemes for Simply-Typed Lambda-Terms*, Kotaro Takeda, Naoki Kobayashi, Kazuya Yaguchi, Ayumi Shinohara, *ICFP 2016*

Résumé

L'article présente deux approches de codage de lambda-termes en chaînes binaires pour obtenir une représentation compacte d'un programme fonctionnel.

Contexte

Travaux de Kobayashi sur l'encodage de structures arborescentes en un programme fonctionnel (i.e. traduisible en lambda-termes) produisant la structure en sortie. L'article présente des méthodes examinées pour compresser cette représentation intermédiaire efficacement.

Avantages de la compression d'ordre supérieur:

- En théorie, le ratio de compression peut être *très élevé*.
- Les données compressées peuvent être manipulées *sans décompression*.
- On peut émuler facilement d'autres schémas de compression.

Pour exploiter la puissance de compression de ces idées, il est nécessaire de savoir encoder de manière compacte des lambda-termes. On se restreint au lambda-calcul simplement typé puisqu'il nous permet de vérifier l'intégrité du payload en vérifiant si le typage est correct.

On évite d'encoder simplement les programmes en caractères les représentant puisque ceci rend possible l'encodage de termes mal formés.

Aucun des travaux d'étude de compression de lambda-termes simplement typés ne sont appropriés pour la compression d'ordre supérieur que vise l'article.

Points principaux

Lambda calcul simplement typé

- Garantie de terminaison des programmes si ils sont bien typés
- On peut laisser tomber des caractères dont une représentation classique aurait besoin, puisqu'on peut les retrouver à partir des informations de type - typiquement, (et) sont redondants avec les informations de type contenues dans l'encodage. Par exemple, on peut faire la différence entre $a(b\ c)$ et $(a\ b)\ c$.

Typage

```
tau ::= o
      | tau_1 -> tau_2
```

\circ est l'unique type de base dans ce cas précis, il dénote le type des données de la structure arborescente qu'on cherche à compresser.

Simplification / Jugements

```
M ::= n
      | lambda : tau.M
      | M_1 M_2
```

Un lambda-terme est:

- Un index de de Bruijn
- Une abstraction sur une variable de type tau
- Une application de \$ M_1 \$ à \$ M_2 \$