Formalisme de Réécriture de Passes pour l'élimination de Motifs

Pierre LERMUSIAUX,
Directeurs : Horatiu CIRSTEA, Pierre-Etienne MOREAU

Université de Lorraine – LORIA

Introduction

Contexte :
Une pratique courante pour la transformation de programme consiste à diviser la transformation globale en passes ayant des objectifs restreints. Souvent, de telles passes vont chercher à éliminer un motif dans les constructions du langage d'origine.

Objectifs :

- Fournir un Formalisme de Réécriture adapté à l'écriture de passes
- Permettre la vérification de l'élimination d'un motif par une passe

Exemple

$$Expr = int(Int) \mid str(String) \mid lst(List) \qquad List = nil() \mid cons(Expr, List)$$

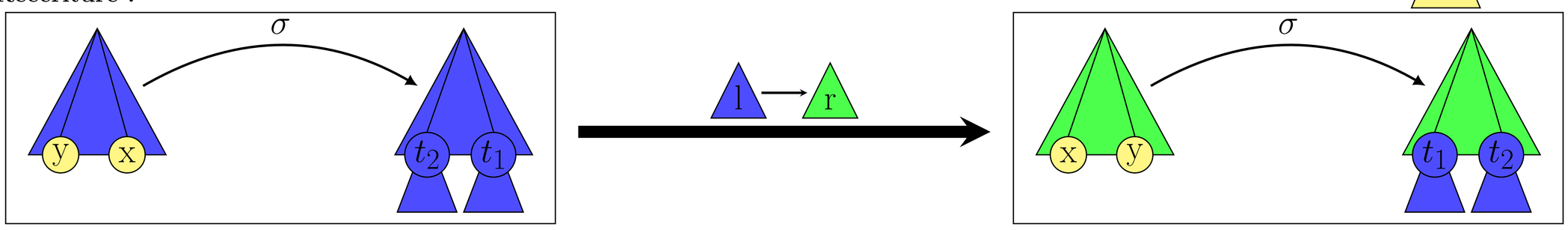
- Une fonction $flatten : List \mapsto List$ doit éliminer le motif $p_{ex} = cons(lst(l_1), l_2)$

Réécriture

Algèbre Multi-sorté : l'ensemble des termes construits à partir de :

- Un ensemble de sortes $\mathcal{S}$ et leurs constructeurs $\mathcal{C}$ (i.e. Algebraic datatypes)
- Un ensemble de symboles définis $\mathcal{D}$ (i.e. fonctions de transformations)
- Un ensemble de variables sortées $\mathcal{X}$

Règle de Réécriture :

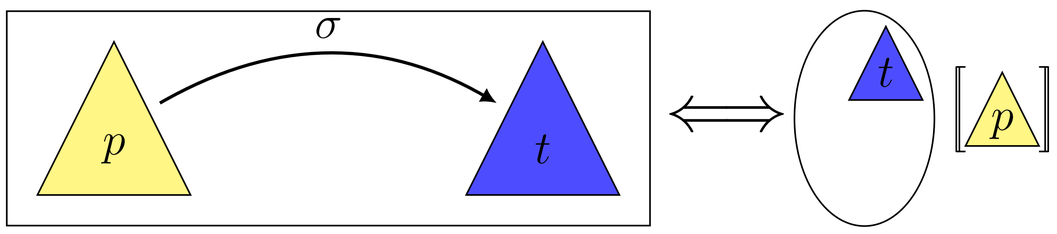


Exemple :

$$\begin{cases} flatten(cons(lst(l_1), l_2)) & \twoheadrightarrow & flatten(concat(l_1, l_2)) \\ concat(cons(e, l_1), l_2) & \twoheadrightarrow & cons(e, concat(l_1, l_2)) \\ concat(nil(), l) & \twoheadrightarrow & l \end{cases}$$

Sémantique de Motif

Définition:
Sémantique d'un motif p : $[\![p]\!] = \{\sigma(p) \mid \sigma(p) \in \mathcal{T}_s(\mathcal{C})\}$



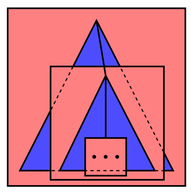
Calcul:

- complément de 2 motifs u et v : $u \setminus v \Longrightarrow [\![u \setminus v]\!] = [\![u]\!] \setminus [\![v]\!]$
- conjonction de 2 motifs u et v : $u \times v \Longrightarrow [\![u \times v]\!] = [\![u]\!] \cap [\![v]\!]$

$\Longrightarrow$ comme une somme de motifs $[\![\sum_i u_i]\!] = \bigcup_i [\![u_i]\!]$

Exemption de Motif

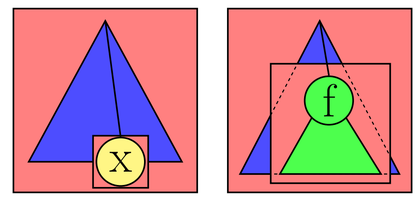
Définition :
On dit qu'une valeur $c(v_1, \ldots, v_n)$ est **exempt** d'un motif p si et seulement si $p \not\twoheadrightarrow c(v_1, \ldots, v_n)$ et $\forall i, v_i$ est **exempt** de p



Exemple :

- $cons(str(\text{"}Bonjour\text{"}), cons(int(42), nil()))$ est **exempt** de p_{ex}
- $cons(lst(cons(str(\text{"}Bonjour\text{"}), nil())), cons(int(42), nil()))$ n'est pas **exempt** de p_{ex}

Définition étendue :
On dit qu'un terme t est **exempt** d'un motif p si et seulement si $\forall v \in [\![t]\!]$, v est **exempt** de p



Exemple :

- $cons(str(\text{"}Bonjour\text{"}), cons(int(42), l_x))$ n'est pas **exempt** de p_{ex}
- $flatten(cons(lst(cons(str(\text{"}Bonjour\text{"}), nil())), cons(int(42), nil())))$ est **exempt** de p_{ex}

Génération de règles

Grâce à la notion de sémantique de motif, l'analyse des règles définissantes permet d'identifier les motifs manquants pour obtenir un système complet. Les règles adéquates peuvent alors être générées pour ces motifs en fonction de la stratégie de traverse.

Exemple :

$$\begin{cases} concat(conc(e, l_1), l_2) & \twoheadrightarrow & conc(e, concat(l_1, l_2)) \\ concat(nil(), l) & \twoheadrightarrow & l \\ flatten(str(s)) & \twoheadrightarrow & \boxed{str(s)} \\ flatten(int(n)) & \twoheadrightarrow & \boxed{int(n)} \\ flatten(lst(l)) & \twoheadrightarrow & \boxed{lst(flatten(l))} \\ flatten(nil()) & \twoheadrightarrow & \boxed{nil()} \\ flatten(conc(str(s), l)) & \twoheadrightarrow & \boxed{conc(str(s), flatten(l))} \\ flatten(conc(lst(l_1), l_2)) & \twoheadrightarrow & \boxed{flatten(concat(l_1, l_2))} \end{cases}$$

Vérification

Soit une règle $f(l_1, \ldots, l_n) \twoheadrightarrow r$ avec f éliminant le motif p :

- si r est **exempt** de p, alors $\forall u_1, \ldots, u_n, v$,

$$f(u_1, \ldots, u_n) \xrightarrow{f(l_1, \ldots, l_n) \twoheadrightarrow r} v \Longrightarrow v \text{ est } \textbf{exempt} \text{ de } p$$

Exemple :

Références

- Pierre Lermusiaux, Horatiu Cirstea, Pierre-Etienne Moreau. Pattern eliminating transformations. CIEL 2019 - 8ème Conférence en IngénieriE du Logiciel, Jun 2019.
- Horatiu Cirstea and Pierre-Etienne Moreau. 2019. Generic Encodings of Constructor Rewriting Systems. In Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages 2019 (PPDP '19).