

Ré-écriture

2h : avec documents

Année 2018-2019

Préambule

- Les contrats (entête + tests unitaires) de chaque fonction doivent être donnés.
- Vous devez tout écrire dans le fichier `boa.ml`.
- Les noms de modules et fonctions doivent être respectés (tests automatiques).
- Il ne faut pas modifier le fichier `test.ml`. Ce fichier est uniquement là pour vérifier que votre architecture est compatible avec les tests automatiques. Les tests unitaires doivent être réalisés dans le fichier `boa.ml`.
- Pour tester dans `utop`, vous devez ouvrir les modules **Be** et **Boa** (`open Be.Boa;;`) pour avoir accès aux modules et interfaces que vous définirez.
- Le code rendu doit impérativement compiler.
- Les exercices 1 et exercices 2 sont indépendants.

Sujet d'étude

Le but est de coder un algorithme qui à partir :

- d'un *terme*
- d'un ensemble de règles permettant de générer un ou plusieurs termes à partir d'un terme
- et d'un terme cible

donne la suite de règles et de termes permettant d'obtenir le terme cible à partir du terme initial.

En particulier, nous nous étudierons le système, nommé BOA, suivant :

- les termes sont des listes de caractères utilisant uniquement trois lettres : 'B', 'O' et 'A'
- les règles de transformation sont les suivantes :
 - **Règle I** : À partir d'une chaîne se terminant par un 'O' vous pouvez générer une chaîne identique à laquelle un 'A' est ajouté à la fin.
Par exemple ['B'; 'O'] permet de générer ['B'; 'O'; 'A'].
 - **Règle II** : À partir d'une chaîne commençant par un 'B' vous pouvez générer une chaîne identique à laquelle est ajoutée à la fin la chaîne initiale privée du 'B'.
Par exemple ['B'; 'O'; 'A'] permet de générer ['B'; 'O'; 'A'; 'O'; 'A'].
 - **Règle III** : À partir d'une chaîne contenant les sous-chaînes ['O'; 'O'; 'O'] ou ['A'; 'O'; 'A'], vous pouvez générer une chaîne où ces sous-chaînes ont été remplacées par un 'A'.
Par exemple ['B'; 'O'; 'O'; 'O'; 'O'] permet de générer ['B'; 'O'; 'A'] et ['B'; 'A'; 'O'].
 - **Règle IV** : À partir d'une chaîne contenant la sous-chaîne ['A'; 'A'], vous pouvez générer une chaîne où cette sous-chaîne a été supprimée.

Par exemple $[' B ' ; ' O ' ; ' A ' ; ' A ' ; ' O ']$ permet de générer $[' B ' ; ' O ' ; ' O ']$.

1 Les règles du système BOA

Les règles du système de réécriture sont caractérisées par un identificateur et le comportement de l'application de celles-ci.

L'interface correspondant à une règle de réécriture est la suivante :

```
module type Regle =
sig
  type tid = int
  type td
  val id : tid

  val appliquer : td -> td list
end
```

où **tid** est le type des identifiants de règles et **td** est le type des termes.

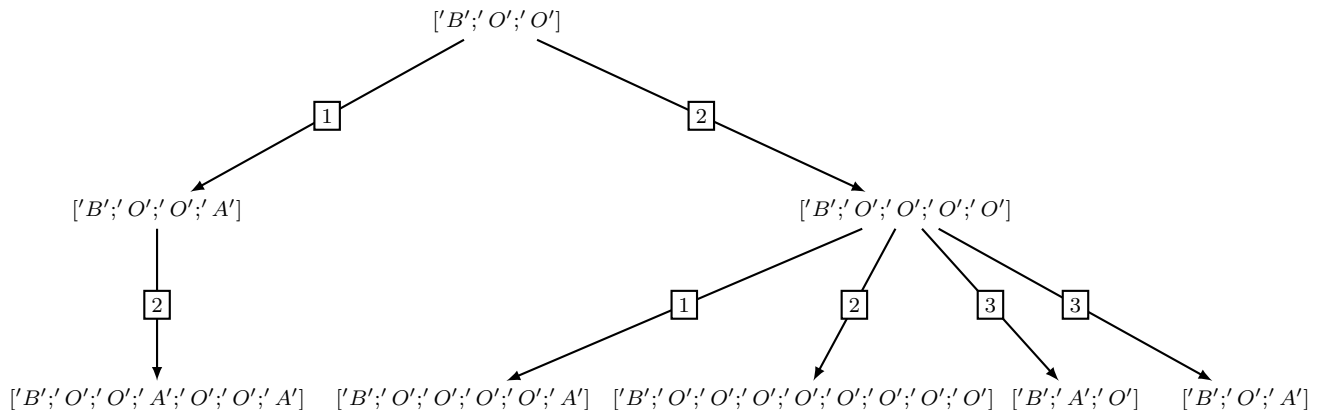
► Exercice 1

1. Compléter l'interface **Regle** pour ajouter les contrats.
2. Pour chacune des règles du système BOA, écrire un module conforme à l'interface **Regle**. Ils devront se nommer **Regle1**, **Regle2**, **Regle3** et **Regle4**. Les termes retournés par la fonction **appliquer** doivent d'être obtenus avec une unique application de la règle. Tous les termes obtenables par une application doivent être renvoyés.

2 Arbre de réécriture

Pour représenter l'ensemble des termes générables à partir d'un terme et la façon de les obtenir, nous utiliserons un arbre n-aire avec des termes dans les nœuds et des identifiants de règles sur les branches.

Par exemple, l'arbre de réécriture, engendré par les règles 1 à 4, de profondeur 2, à partir du terme $[' B ' ; ' O ' ; ' O ']$ est le suivant :



L'interface correspondant aux arbres de réécriture est la suivante :

```
module type ArbreReecriture =  
sig  
  type tid = int  
  type td  
  type arbre_reecriture = ...  
  
  val creer_noeud : ...  
  
  val racine : ...  
  val fils : ...  
  
  val appartient : td -> arbre_reecriture -> bool  
end
```

où `tid` est le type des identifiants de règles, `td` est le type des termes, `arbre_reecriture` est le type des arbres n-aires décrits précédemment et les fonctions `creer_noeud`, `racine` et `fils` sont des constructeurs et destructeurs du type `arbre_reecriture`.

▷ Exercice 2

1. Compléter cette interface pour ajouter les contrats, définir le type `arbre_reecriture` et donner les types de `creer_noeud`, `racine` et `fils`.
2. Définir un module `ArbreReecritureBOA` conforme à `ArbreReecriture` pour la cas particulier du système BOA
3. À l'intérieur du module `ArbreReecritureBOA`, définir 2 exemples d'arbre de réécriture et écrire les tests unitaires des fonctions du module.

3 Le système BOA

▷ Exercice 3

Écrire un module `SystemeBOA` contenant :

1. une fonction `construit_arbre` qui à partir d'une liste de caractères et d'un entier n , crée l'arbre de réécriture contenant les solutions obtenues en appliquant au maximum n règles (règle I à IV de l'exercice 1).
2. une fonction `chemin` qui à partir d'une liste de caractères de départ, d'une liste de caractères cible et d'un entier n , renvoie la liste des règles appliquées et la liste des listes de caractères successives, pour passer de la liste de caractères de départ à la liste de caractères cible (toutes deux incluses), en maximum n transformations. Le type `'a option` sera utilisé pour différencier les cas où la liste de caractères cible appartient à l'arbre de réécriture de ceux où elle ne lui appartient pas.
3. une vérification qu'il y a bien une solution pour passer de la liste de caractères `['B'; 'O']` à la liste de caractères `['B'; 'A']` en cinq transformations au plus.

4 Généralisation (BONUS)

Dans le cas général, un système de réécriture est caractérisé par une ensemble de règles (couple : identifiant et comportement de la règle).

L'interface correspondant à un système de réécriture est la suivante :

```
module type ReglesReecriture =  
sig  
  type tid = int  
  type td  
  
  val regles : (tid * (td -> td list)) list  
end
```

où `tid` est le type des identifiants de règles et `td` est le type des termes.

▷ Exercice 4

1. Écrire un foncteur `SystemeReecriture` paramétré par un `ArbreReecriture` et un `ReglesReecriture` qui définit les fonctions `construit_arbre` et `chemin` dans le cas général.
2. Écrire un module `ReglesReecritureBOA` conforme à `ReglesReecriture` qui regroupe les règles du système BOA.
3. Écrire un module `SystemeReecritureBOA` qui instancie le foncteur `SystemeReecriture` dans le cas particulier du système BOA.
4. Vérifier que les tests du module `SystemeBOA` sont également valides avec le module `SystemeReecritureBOA`.