LANGAGES DE PROGRAMMATION
IFT-3000
HIVER 2019

Travail pratique 2 - TRAVAIL INDIVIDUEL OU EN ÉQUIPE

Pondération de la note finale : 15%

À remettre, par le biais du site Web du cours, et avant 23h59, le vendredi 26 avril 2019

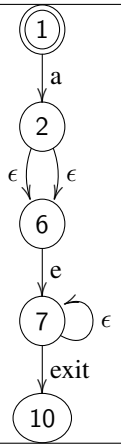
1 Énoncé

Ce travail pratique consiste à implanter un outil pour la comparaison de programmes. Plus précisément, dans ce travail, nous nous intéressons à l'équivalence et à la transformation de programmes. Par exemple, en utilisant la syntaxe ML, il s'agira de définir une fonction «bisimilaire» ayant la signature suivante :

bisimilaire : programme -> programme -> bool

Comme le décrit la table suivante, nous considérons un programme comme la donnée d'un ensemble de transitions et d'un état initial. Ces transitions sont étiquetées par des appels de fonctions (API) ou par une action particulière nommée *Epsilon* (ϵ). Cette dernière abstrait toute instruction du programme qui ne correspondant pas à un appel de fonctions (ex : affectation).

TABLE 1 – Différentes représentations d'un programme.

Code C	Modèle abstrait (graphe)	Description en ML	Version simplifiée (sans ϵ)
0 : `main()` { 1 : int x = a() ; 2 : **if** (x == 0) 3 : x++ ; 4 : **else** 5 : x– ; 6 : e() ; 7 : **while** (i– != 0) 8 : x++ ; 9 : exit() ; 10 : }		([(1, Api("a"), 2) ; (2, Epsilon, 6) ; (2, Epsilon, 6) ; (6, Api("e"), 7) ; (7, Epsilon, 7) ; (7, Api("exit"), 10)], 1)	([(1, Api("a"), 2) ; (2, Api("e"), 7) ; (7, Api("exit"), 10)], 1)

La traduction d'un programme C vers une description abstraite (transitions, etat_initial) peut être effectuée par un analyseur syntaxique (*parseur*). Cette traduction ne fait pas partie de ce TP et pourrait être réalisée dans le cadre d'un futur TP qui complèterait alors celui en cours. Cependant, notez que cette traduction produit, en général, un système de transitions admettant des transitions ϵ. Il sera question, dans ce TP, de transformer la description abstraite d'un programme en éliminant ce type de transitions (ϵ-transitions).

La section qui suit décrit de manière plus précise les structures de données permettant de représenter un programme ainsi que la description des différentes fonctions à implanter.

2 Spécification (Signature)

Comme mentionné précédemment, dans ce TP, nous considérons un programme comme la donnée d'une liste de transitions et d'un état initial. Une transition lie 2 états par une action. Elle correspond donc à un triplet réunissant 2 états et une action. Une action sera représentée par une structure de données permettant de modéliser à la fois l'action ϵ ainsi que les appels de fonctions API (string). Les états seront représentés par des valeurs entières (nous pouvons considérer que ces valeurs correspondent aux numéros d'instructions du programme traité).

En utilisant la syntaxe du langage OCaml, nous obtenons donc les définitions suivantes :

```
type action = Epsilon | Api of string
type etat = int
type transition = etat * action * etat
type programme = transition list * etat
```

Par ailleurs, voici la signature des principales fonctions à implanter :

— supprimeEpsilon : programme -> programme
  Cette fonction génère une nouvelle version d'un programme passé en argument dans laquelle toutes les ϵ-transitions sont éliminées (l'ordre des éléments da la liste des transitions du programme généré n'importe pas).
— similaire : programme -> programme -> bool
  Cette fonction teste si un programme est similaire à un autre programme.
— bisimilaire : programme -> programme -> bool
  Cette fonction teste si un programme est bisimilaire à un autre programme.
— estSousPgm : programme -> programme -> bool
  Cette fonction teste si un programme est «inclus» dans un (sous-programme d'un) autre programme.

En outre, deux autres fonctions (l'ordre des éléments dans les listes résultantes n'importe pas) sont à implanter :

— transitionsImmediates : programme -> etat -> transition list * transition list
  Cette fonction retourne respectivement les api-transitions immédiates d'un état du programme passé en argument et ses ϵ-transitions immédiates.
— epsilonAtteignable : programme -> (etat * etat list) list
  Cette fonction retourne pour chaque état d'un programme, la liste de tous les états qu'il peut atteindre en effectuant une séquence de 1 ou plusieurs ϵ-transitions.

Dans ce TP, nous considérons une relation permettant de comparer des programmes entre eux. Cette relation est définie comme suit :

$$P \sim Q \ \ ssi \ \ \forall a, P \xrightarrow{a} P' \ \Rightarrow \ (Q \xrightarrow{a} Q' \ \wedge \ P' \sim Q')$$

Autrement dit, et de manière plus informelle, un programme P est *similaire* à un programme Q ($P \sim Q$) si pour chaque action a (différente de ϵ) que peut effectuer P ($P \xrightarrow{a} P'$), Q peut aussi l'effectuer ($Q \xrightarrow{a} Q'$) et l'état du programme P', résultant de cette action, est *similaire* à l'état du programme Q', résultant de cette action aussi.

La *bisimilarité* entre deux programmes est définie comme suit :

$$P \approx Q \ \ ssi \ \ (P \sim Q \ \wedge \ Q \sim P)$$

Un programme P_1 est un sous-programme d'un programme P si ce dernier admet un état e_pivot à partir duquel le programme P_1 lui est *similaire*, c'est-à-dire que pour chaque action a (différente de ϵ) que peut effectuer P_1, P, à l'état e_pivot, peut aussi l'effectuer et l'état du programme P'_1, résultant de cette action, est *similaire* à l'état du programme P', résultant de cette action aussi.

Pour terminer, afin d'éliminer les ϵ-transition d'un programme donné P et obtenir une nouvelle version, P', il est possible de procéder comme suit (notez que nous nous référons à l'exemple présenté au début de cette énoncé, page 1) :

1. On calcule les états de P'. Ces états correspondent à ceux de P qui sont les cibles d'une api-transition. À cet ensemble, il faut ajouter l'état initial de P.
   En se référant à l'exemple, les états de P' sont [2 ;7 ;10 ;1].
2. Pour chaque état du programme P, on calcule la liste des états qui peuvent être atteints par une séquence de 1 ou plusieurs ϵ-transitions. Nous considérons le résultat de ce traitement comme un dictionnaire (notez que la fonction «epsilonAtteignable» réalise ce traitement !).
   En se référant à l'exemple, le résultat serait : [(1,[]) ;(2,[6]) ;(6,[]) ;(7,[7]) ;(10,[])].

3. On calcule les nouvelles transitions de P', en procédant comme suit. Tout d'abord, en considérant les transitions de P, on retient toutes les api-transitions immédiates des états de P'. En se référant à l'exemple, nous aurions les transitions suivantes :

   [(1,Api("a"),2) ;(7,Api("exit"),10)].

   À cela, il faut ajouter d'autres transitions qui sont déterminées comme suit : Pour chaque état s de P', on considère la liste des états qui lui est associée dans le dictionnaire. Pour chaque état de cette liste, s'il admet une api(a)-transition qui mène vers un état s', alors on ajoute la transition (s, Api(a), s') à celles déjà retenues.

   En se référant à l'exemple, et au dictionnaire [(1,[]) ;(2,[6]) ;(6,[]) ;(7,[7]) ;(10,[])], les seuls états auxquels correspondent des listes non vides sont 2 et 7. Puisque l'état 6 admet une api(e)-transition qui aboutit à l'état 7, on ajoute donc une transition (2,Api("e"),7) aux transitions de P'. Aussi, l'état 7 admet une api(exit)-transition qui mène à l'état 10. Par conséquent, il faudrait ajouter une transition (7,Api("exit"),10). Notez que cette transition a déjà été retenu à l'étape précédente (voir ci-dessus).

Au final, et en se référant au même exemple, nous obtenons un programme défini comme suit :

([(1, Api("a"), 2) ; (2, Api("e"), 7) ; (7, Api("exit"), 10)], 1).

3 Implantation (module)

En considérant les programmes définis [1] comme suit (pgm1 correspond à celui de la page 1) :

let pgm1 = ([(1, Api "a", 2) ; (2, Epsilon, 6) ; (2, Epsilon, 6) ; (6, Api "e", 7) ; (7, Epsilon, 7) ;
(7, Api "exit", 10)],1) ; ;
let pgm2 = ([(1, Api "a", 2) ; (2, Epsilon, 5) ; (5, Epsilon, 6) ; (6, Api "c", 7) ; (7, Epsilon, 7) ; (1, Epsilon, 3) ;
(3, Api "a", 4) ; (4, Epsilon, 4) ;(4, Api "b", 8)],1) ; ;
let pgm3 = ([(1, Epsilon, 2) ; (2, Api "a", 6) ; (6, Epsilon, 7) ; (7, Api "b", 10) ; (10, Epsilon, 10) ;
(6, Api "c", 8) ; (1, Epsilon, 4) ; (4, Epsilon, 1)], 1) ; ;
let pgm4 = ([(0, Api "a", 6) ; (0, Epsilon, 3) ; (6, Api "b", 10) ; (6, Api "c", 8) ; (8, Epsilon, 8)],0) ; ;

Voici les résultats des différentes fonctions que vous devez implanter :

```
# transitionsImmediates pgm1 1;;
_ : transition list * transition list = ([(1,Api "a",2)],[])
# transitionsImmediates pgm3 1;;
- : transition list * transition list = ([], [(1, Epsilon, 2); (1, Epsilon, 4)])
# transitionsImmediates pgm1 2;;
_ : transition list * transition list = ([],[(2,Epsilon,6);(2,Epsilon,6)])
# transitionsImmediates pgm3 6;;
- : transition list * transition list = ([(6, Api "c", 8)], [(6, Epsilon, 7)])
# transitionsImmediates pgm1 7;;
_ : transition list * transition list = ([(7,Api "exit",10)],[(7,Epsilon,7)])
# transitionsImmediates pgm1 10;;
_ : transition list * transition list = ([],[])

# epsilonAtteignable pgm1;;
- : (etat * etat list) list =
  [(10, []); (7, [7]); (6, []); (2, [6]); (1, [])]
# epsilonAtteignable pgm2;;
- : (etat * etat list) list =
  [(8, []); (4, [4]); (3, []); (7, [7]); (6, []); (5, [6]); (2, [5; 6]);(1, [3])]
# epsilonAtteignable pgm3;;
- : (etat * etat list) list =
  [(4, [1; 4; 2]); (8, []); (10, [10]); (7, []); (6, [7]); (2, []);(1, [2; 4; 1])]
```

1. La représentation graphique des principaux programmes définis dans ces exemples est présentée en annexe.

```
# let pgm1' = supprimeEpsilon pgm1;;
val pgm1' : programme =
    ([(1, Api "a", 2); (7, Api "exit", 10); (2, Api "e", 7)], 1)
# let pgm2' = supprimeEpsilon pgm2;;
val pgm2' : programme =
    ([(1, Api "a", 2); (1, Api "a", 4); (4, Api "b", 8); (2, Api "c", 7)], 1)
# let pgm3' = supprimeEpsilon pgm3;;
val pgm3' : programme =
    ([(6, Api "c", 8); (1, Api "a", 6); (6, Api "b", 10)], 1)
# supprimeEpsilon pgm4;;
- : programme = ([(6, Api "c", 8); (6, Api "b", 10); (0, Api "a", 6)], 0)

# similaire pgm2' pgm3';;
- : bool = true
# similaire pgm3' pgm2';;
- : bool = false

# bisimilaire pgm2' pgm3';;
- : bool = false
# bisimilaire pgm3' pgm4;;
_ : bool = true
# bisimilaire pgm4 pgm3';;
_ : bool = true

# estSousPgm ([(0, Api "e", 1); (1, Api "exit", 2)],0) pgm1;;
_ : bool = true
# estSousPgm ([(0, Api "a", 1); (1, Api "c", 2)],0) pgm3;;
_ : bool = true
# estSousPgm pgm1' pgm1';;
_ : bool= true
# List.map (estSousPgm ([],0)) [pgm1;pgm2;pgm3;pgm4;pgm1';pgm2';pgm3'];;
- : bool list = [true; true; true; true; true; true; true]
```

4 Démarche à suivre

Il faut créer un fichier "tp2.ml" comprenant la définition d'un module "Tp2" qui devra respecter la signature précisée dans le fichier "tp2.mli" qui vous a été remis. Au niveau de la correction, le code suivant ne devra pas provoquer d'erreurs :

```
# #use "tp2.mli";; (* On utilisera la version du fichier remise avec l'énoncé *)
   ...
# #use "tp2.ml";;  (* Ce sera le fichier que vous aurez remis *)
                   (* -10 pts si cette ligne provoque une erreur *)
   ...
# module TestConformite = (Tp2 : TP2);; (* -10 pts si cette ligne provoque une erreur *)
module TestConformite : TP2
```

Notez qu'au besoin, vous pouvez utiliser le code du corrigé du Tp1 (évidemment, il faudra l'adapter (ex : fonction show_graph) pour tenir compte de la nouvelle définition des types, notamment celle des actions). Vous pouvez aussi utiliser et combiner les paradigmes de programmation de votre choix.

5 À remettre

Il faut remettre un fichier .zip contenant uniquement un fichier "tp2.ml". Le code doit être clair et bien structuré [2].

2. Suggestion : http://ocaml.org/learn/tutorials/guidelines.html

6 Remarques importantes

Plagiat : Tel que décrit dans le plan de cours, le plagiat est interdit. Une politique stricte de tolérance zéro est appliquée en tout temps et sous toutes circonstances. Tous les cas seront référés à la direction de la Faculté.

Travail remis en retard : Tout travail remis en retard se verra pénalisé de 50%. Un retard excédant 1 jours (24h) provoquera le rejet du travail pour la correction et la note de 0 pour ce travail. La remise doit se faire par la boite de dépôt du TP2 dans la section «Évaluation et résultats».

Barème : Au niveau du code à implanter, le barème est précisé dans le fichier "tp2.mli". Notez cependant que :

- (-10pts), si vous ne remettez pas un unique fichier nommé "tp2.ml" et que ce dernier ne comprend pas la définition d'un module nommé "Tp2".
- (-10pts), si votre code ne compile pas (provoque une erreur/exception lors du chargement du fichier "tp2.ml" dans l'interpréteur), ou n'est pas conforme à la spécification TP2, précisé dans le fichier "tp2.mli" (voir section 4, page 4).
- tout code en commentaire ne sera pas corrigé.

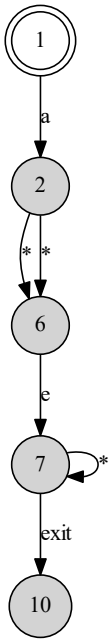
Bon travail.

Annexe

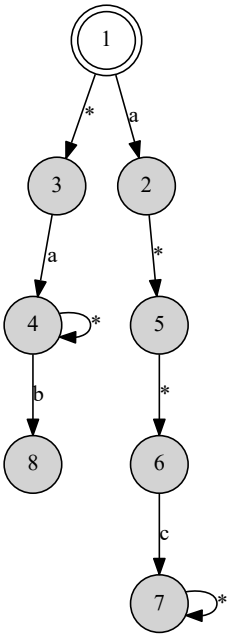
7 Représentation graphique de programmes

Dans cet annexe, on présente graphiquement les exemples de programmes utilisés dans l'énoncé (le symbole étoile, «*», représente l'action ϵ).

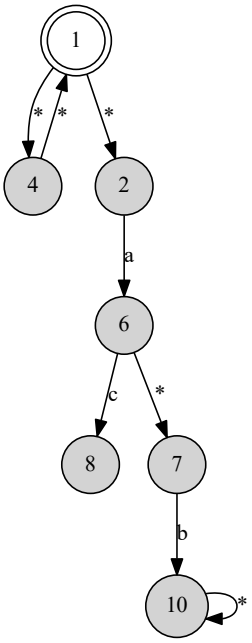
pgm1



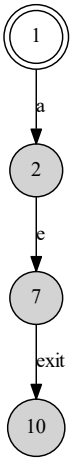
pgm2



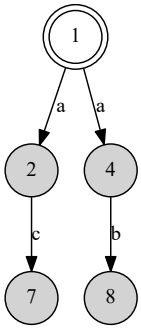
pgm3



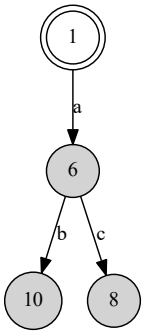
pgm1'



pgm2'



pgm3'



pgm4