INFO0054 - Programmation fonctionnelle

Projet: Puzzles réguliers

Jean-Michel Begon

2015-2016

Un automate fini déterministe est défini par un tuple $(Q, \Sigma, \delta, s, F)$, où

Q est un ensemble fini d'états ;

Σ est un alphabet fini ;

$\delta : Q \times \Sigma \to Q$ est la fonction de transition ;

s est l'état initial ;

$F \subseteq Q$ est l'ensemble des états accepteurs.

En outre, on appelle "mot" une suite finie de symboles $\{\sigma_i\}_{i=1}^{N}$ ($\sigma_i \in \Sigma$).

On dit qu'un mot $\{\sigma_i\}_{i=1}^{N} = \sigma_1 \dots \sigma_N$ est accepté par l'automate si la propriété suivante est satisfaite :

— $q_1 = s$

— $q_{i+1} = \delta(q_i, \sigma_i) \quad 1 \leq i \leq N$

— $q_{N+1} \in F$

Autrement dit, la suite de symboles permet de passer de l'état initial vers un des états accepteurs.

L'ensemble des mots acceptés par l'automate forme un langage. La classe des langages reconnaissables par des automates finis déterministes s'appelle la classe des langages réguliers.

Dans le cadre de ce projet, nous allons nous intéresser à des puzzles qui peuvent se mettre sous la forme de tels automates. L'ensemble des solutions forme alors un langage, dénoté L. Le sous-langage des mots acycliques est dénoté $L_{\not\circlearrowleft}$.

Par mot acyclique, on entend un mot tel que la séquence de transitions qui lui est liée ne passe jamais deux fois par le même état.

1 Exemple — Le loup, le mouton et le chou

Soit le puzzle suivant :

> "Vous êtes accompagné d'un loup, d'un mouton et d'un chou. Vous devez les faire traverser une rivière mais vous n'avez à votre disposition qu'une barque ne pouvant transporter qu'un seul des trois avec vous. Si le loup est seul avec le mouton, il va le manger. Si le mouton est seul avec le chou, il va le manger. Est-il possible faire traverser le loup, le mouton et le chou ?"

Son automate est illustré à la figure 1. Chacun du passeur (P), du loup (L), du mouton (M) ou du chou (C) peut se trouver du côté initial ou de l'autre côté de la rivière. A ces seize états, on ajoute un état particulier, le *sink*, tel que $sink = \delta(sink, \sigma) \quad \forall \sigma \in \Sigma$.

L'état initial est $s = (\{\}, \{P, L, M, C\})$: tous les quatre sont sur la rive initiale. L'ensemble des états accepteurs est le singleton $F = \{(\{P, L, M, C\}, \{\})\}$.

L'alphabet est composé de quatre symboles $\Sigma = \{p, l, m, c\}$ qui représentent respectivement "le passeur traverse seul", "le passeur traverse avec le loup", "le passeur traverse avec le mouton", "le passeur traverse avec le chou".

Le problème admet deux solutions (*i.e.* mots) sans cycle de sept symboles : *mplmcpm* et *mpcmlpm*. La première se traduit par :

s : On démarre de la rive initiale : $(\{\}, \{P, L, M, C\})$.

$\rightarrow_m$ On transporte le mouton de l'autre côté : $(\{P, M\}, \{L, C\})$.

$\rightarrow_p$ On revient en laissant le mouton : $(\{M\}, \{P, L, C\})$.

$\rightarrow_l$ On transporte le loup de l'autre côté : $(\{M, P, L\}, \{C\})$.

$\rightarrow_m$ On revient avec le mouton : $(\{L\}, \{P, M, C\})$.

$\rightarrow_c$ On transporte le chou de l'autre côté : $(\{L, P, C\}, \{M\})$.

$\rightarrow_p$ On revient en laissant le loup et le chou : $(\{L, C\}, \{P, M\})$.

$\rightarrow_m$ On transporte le mouton de l'autre côté : $(\{P, L, M, C\}, \{\}) \in F$.

La solution à treize symboles *mpcmlcmlcmlpm* contient un cycle car elle repasse par les états $(\{M\}, \{P, L, C\})$, $(\{P, M, C\}, \{L\})$, $(\{C\}, \{P, L, M\})$, $(\{P, L, C\}, \{M\})$.

2 Enoncé

2.1 Le *solver*

Le but du projet est d'implémenter un *solver* générique de puzzles réguliers. La fonction principale à implémenter est :

```
(rp-solve s adj acc-state?)
```

Elle prend trois arguments, tous liés à un même puzzle $P = (Q, \Sigma, \delta, s, F)$:

1. `s` : la représentation de l'état initial ;
2. `adj` : la fonction d'adjacence de P : $adj(q) = \{(\sigma, q_j) \in \Sigma \times Q | q_j = \delta(q, \sigma)\} \quad \forall q \in Q$. On représentera une paire (σ, q_j) par une paire pointée et l'ensemble de ces paires par une liste.
3. `acc-state?` : le prédicat unaire défini sur l'ensemble Q et valant $True$ si et seulement si son argument est issu de F.

Elle renvoie un itérateur paresseux du sous-langage de P constitué des mots sans cycle ($L_{\not\circlearrowright}$). Un mot est représenté par la liste de ses symboles.

Un itérateur paresseux pour une suite $m_1, \ldots, m_p$ est une suite $f_1, \ldots, f_p, f_{p+1}$ telle que les p premiers éléments sont des fonctions vérifiant $(f_k) = (m_k.f_{k+1})$ et $(f_{p+1}) =' ()$.

En mots, l'évaluation de f_k pour $1 \leq k \leq p$ est une paire pointée dont le `car` est m_k et le `cdr` est f_{k+1}. Le programme fonctionne donc même dans le cas d'un langage infini.

La fonction `rp-solve` renvoie bien une fonction et non une paire ! C'est l'évaluation de cette fonction qui produit la paire.

Toutes les informations relatives à un puzzle donné doivent être encapsulées dans les trois paramètres de la fonction.

Bonus : il est en général souhaitable d'obtenir les solutions par ordre croissant de taille. Cette contrainte n'est pas nécessaire mais sera récompensée.

2.2 Cases illuminées

Pour tester votre implémentation, il vous est demandé d'implémenter les paramètres de la fonction `rp-solve` relatifs au puzzle des cases illuminées.

Celui-ci se base sur un échiquier 3x3. Certaines cases sont initialement illuminées et d'autres non. Un pion se trouve sur une des cases. Il ne peut se déplacer que sur une case adjacente à la sienne. Lorsque le pion se déplace sur une nouvelle case, celle de destination ainsi que ses voisines s'illuminent si elles étaient éteintes ou inversement s'éteignent si elles étaient illuminées. Le but est

d'illuminer toutes les cases, peu importe l'emplacement final du pion. Un exemple de configuration est donné à la figure 2.

L'alphabet est donc constitué des quatre symboles $\Sigma = \{u, d, l, r\}$ qui correspondent respectivement à un déplacement du pion vers le haut, vers le bas, vers la gauche et vers la droite.

Pour ce problème, vous devez définir :

— `s-illum` : la représentation de l'état initial correspondant à la figure 2.

— `adj-illum` : la fonction d'adjacence du jeu des cases illuminées.

— `acc-state-illum?` : le prédicat d'acceptation.

3 Rapport

Dans votre rapport, nous vous demandons de répondre aux questions suivantes :

1. Pour le puzzle des cases illuminées :
   (a) Décrivez vos choix d'implémentation (comment l'état est-il représenté, comment le graphe est-il représenté, etc.)
   (b) Combien y a-t-il d'états différents ?
   (c) Fournissez un mot du langage correspondant au puzzle.
2. Pour la fonction `rp-solve`, décrivez vos choix d'implémentation.

4 Soumission et remarques

Le projet est à réaliser par groupes de deux. Il doit être rendu via la plateforme cicada : http://submit.run.montefiore.ulg.ac.be/. sous la forme d'une archive au format `tar.gz` contenant :

— votre rapport au format PDF,

— un fichier intitulé `solver.scm` contenant la fonction `rp-solve`,

— un fichier intitulé `illum.scm` contenant tous les éléments relatifs au puzzle.

Veillez à rendre `rp-solve`, `s-illum`, `adj-illum`, `acc-state-illum?` publiques en ajoutant `(provide s-illum)`, `(provide adj-illum)`, `(provide acc-state-illum)`, `(provide rp-solve)` en en-tête des fichiers correspondants.

Utilisez vos identifiants ULg comme nom de groupe (sxxxxxx-sxxxxxx).

N'oubliez pas de spécifier toutes les fonctions que vous définissez (même à l'aide d'un `letrec`).

Si nécessaire, vous pouvez utiliser l'implémentation de base des ensembles avec les commandes `set`, `set-member?` et `set-add` (à ne pas confondre avec `set-add!`).

Bon travail !

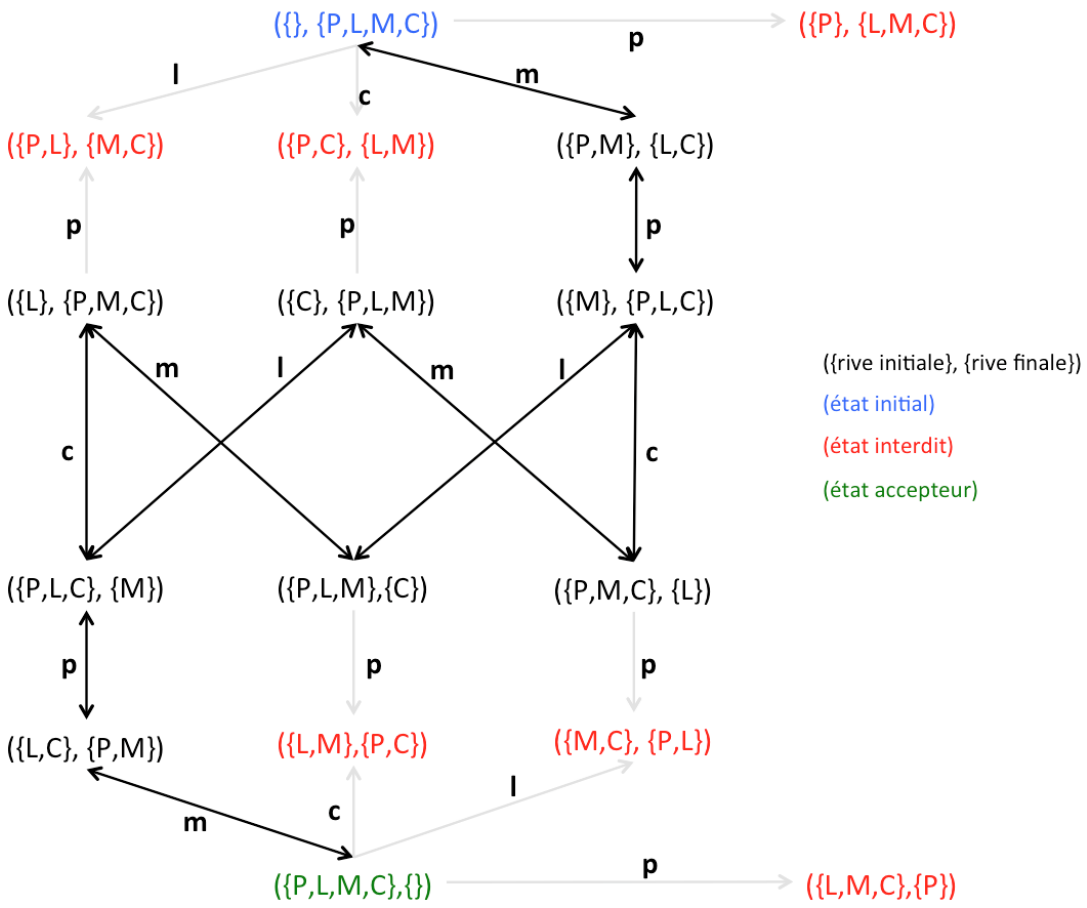

FIGURE 1 – Graphe du loup, du mouton et du chou. Les transitions manquantes sont toutes dirigées vers le *sink*.

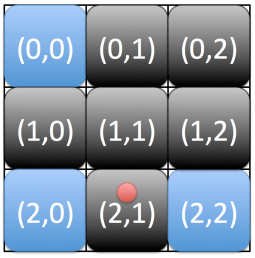

FIGURE 2 – Les cases (0,0), (2,0) et (2,2) sont illuminées. Le pion est en (2,1).

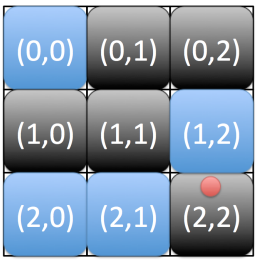

FIGURE 3 – Le pion est maintenant en (2,2)