Travail pratique #2

IFT-2035

18 novembre 2022

1 Survol

Ce TP a pour but de vous familiariser avec le langage Prolog et la programmation logique.

Comme pour les TP précédents, les étapes sont les suivantes :

1. Parfaire sa connaissance de Prolog.
2. Lire et comprendre cette donnée. Cela prendra probablement une partie importante du temps total.
3. Lire, trouver, et comprendre les parties importantes du code fourni.
4. Compléter le code fourni.
5. Écrire un rapport. Il doit décrire **votre** expérience pendant les points précédents : problèmes rencontrés, surprises, choix que vous avez dû faire, options que vous avez sciemment rejetées, etc... Le rapport ne doit pas excéder 5 pages.

Ce travail est à faire en groupes de 2 étudiants. Le rapport, au format LaTeX exclusivement (compilable sur `ens.iro`) et le code sont à remettre par remise électronique avant la date indiquée. Aucun retard ne sera accepté. Indiquez clairement votre nom au début de chaque fichier.

Ceux qui veulent faire ce travail seul(e)s doivent d'abord en obtenir l'autorisation, et l'évaluation de leur travail n'en tiendra pas compte. Des groupes de 3 ou plus sont **exclus**.

2 Les expressions régulières

Vous allez écrire un module qui implante le filtrage par expressions régulières. Les expressions régulières sont une famille de motifs très utilisée pour faire de la recherche dans du texte, entre autre. Une des raisons est qu'il est possible de faire cette recherche de manière très efficace en compilant ces motifs vers ce que l'on appelle des machines à états finis déterministes (DFA). Cette compilation se fait souvent en passant par une phase intermédiaire qui transforme l'expression régulière en une machine à états finis non-déterministe (NFA). Dans ce TP, vous allez implanter cette première phase, ainsi qu'un interpréteur de ces machines NFA.

Les expressions régulières on généralement la forme suivante :

C	motif constant qui accepte le charactère C
.	motif acceptant n'importe quel caractère
$[C_1...C_n]$	accepte n'importe quel caractère parmis $C_1...C_n$
$[\hat{}C_1...C_n]$	accepte n'importe quel caractère pas parmis $C_1...C_n$
ϵ	accepte la chaîne vide
$Re_1\ Re_2$	concaténation : accepte Re_1 suivi de Re_2
$Re_1 \mid Re_2$	disjonction de motifs
$Re?$	motif optionnel
$Re*$	répétition de motif
$Re+$	répétition non-nulle
$(?Name : Re)$	motif nommé

Pour faire de la recherche dans du texte, on ajoute souvent toutes sortes d'extensions tels que des motifs qui reconnaissent le début ou la fin d'une ligne. D'un côté plus théorique, le langage des expressions régulières inclus aussi la conjonction et la négation. Cependant dans le cadre de ce travail, nous nous limiterons aux primitives listées ci-dessus.

3 Les NFA

Une machine a états finis non-déterministe est un graphe dirigé où chaque nœud représente un état et chaque arc indique une transition possible d'un état vers un autre. Les arcs peuvent soit être liés à un événement, auquel cas cette transition n'est possible que si l'événement a lieu, ou sinon ils peuvent être de type ϵ, c'est à dire que la transition peut se faire à n'importe quel moment.

Pour notre application, les événements seront simplement des caractères, et la machine va donc avancer sur une chaîne de caractères en utilisant le caractère courant pour décider quelles transitions sont possibles. Il y aura un état de départ et certains états seront considérés comme finaux, ce qui signifie que lorsque la machine arrive sur un tel état elle a fini sont travail de filtrage et la chaîne qu'elle a parcourue est la chaîne acceptée par la machine (et donc acceptée par l'expression régulière qu'elle représente).

Par exemple, la machine suivante (avec $s1$ comme état de départ) peut être utilisée pour filtrer le motif `"a*a"`, c'est à dire pour reconnaître une chaîne de caractères qui contient une répétition arbitraire de `a` suivie d'une lettre `a` supplémentaire :

$$\begin{aligned} s1 &: \epsilon \to s2, \mathtt{a} \to s1 \\ s2 &: \mathtt{a} \to s3 \\ s3 &: \mathsf{success} \end{aligned}$$

Un détail plus important est que depuis l'état $s1$ on peut toujours sauter à l'état $s2$, même si le caractère courant est `a` et qu'il y a donc souvent plusieurs transitions possibles (d'où le "non-déterministe"). Donc l'exécution d'une telle machine doit en général essayer toutes les transitions possibles afin de trouver lesquelles permettent à la machine d'atteindre un état final.

4 Le code

Le code fourni se compose des parties suivantes :

— Un ensemble de prédicats qui définissent ce qu'est une NFA valide. Ces prédicats ne sont pas directement utiles, sauf comme documentation du format à utiliser (et occasionnellement pour déboguer) : votre code doit utiliser des machines qui adhèrent à ces règles. Ces prédicats ont des noms qui se terminent par `_wf` (pour *well-formed*).

— Même chose pour les expressions régulières.

— Une implantation directe de la recherche d'expressions régulières, sans passer par une machine NFA. Considérer ce code comme une sorte de spécification qui vous permettra de comparer l'exécution de votre code à une implantation de référence. Le point d'entrée principal est `search`.

— Quelques autres relations qui complètent le code que vous devez écrire.

5 Ce qu'il faut faire

Vous devez implanter un algorithme de recherche d'expressions régulières qui utilise une NFA de manière interne. Cela se fait en deux parties :

1. Vous devez implanter le code de `nfa_match`, qui exécute une NFA donnée sur une chaîne donnée.
2. Vous devez implanter le code de `re_comp`, qui prend une expression régulière et la traduit (*compile*) en une NFA.

Vous devez implanter ces parties manquantes du code de sorte que la règle `re_search`, qui compile une expression régulière en une NFA puis la passe a `nfa_search`, se comporte de manière aussi fidèle que possible à `search`. Vous pouvez bien sûr ajouter autant de règles auxiliaires que vous le voulez.

Faites attention à implanter `re_comp` et `nfa_match` de manière suffisamment indépendante pour qu'il soit possible de tester votre `re_comp` avec notre `nfa_match` et vice versa. C'est aussi à cela que servent les tests `nfa_wf` et `state_wf`.

6 Remise

Vous devez remettre deux fichiers (`re.pl` et `rapport.tex`), qui sont à remettre par Moodle/StudiUM.

7 Détails

La note est basée d'une part sur des tests automatiques, d'autre part sur la lecture du code, ainsi que sur le rapport. Le critère le plus important, est que votre code doit se comporter de manière correcte. En règle générale, une solution simple est plus souvent correcte qu'une solution complexe. Ensuite, vient la

qualité du code : plus c'est simple et clair, mieux c'est. S'il y a beaucoup de commentaires, c'est généralement un symptôme que le code n'est pas clair ; mais bien sûr, sans commentaires le code (même simple) et souvent incompréhensible. L'efficacité de votre code est sans importance, sauf s'il utilise un algorithme vraiment particulièrement ridiculement inefficace.

Les 22 points seront répartis comme suit : 4 sur le rapport, 5 sur les tests de `nfa_match`, 5 sur les tests de `re_comp`, 4 sur la qualité du code de `nfa_match`, et 4 sur la qualité du code de `re_comp`.

- Le code ne doit en aucun cas dépasser 80 colonnes.
- Le code doit être bien indenté.
- Tout usage de matériel (code ou texte) emprunté à quelqu'un d'autre (trouvé sur le web, ...) doit être dûment mentionné, sans quoi cela sera considéré comme du plagiat.
- Vérifiez la page web du cours, pour d'éventuels errata, et d'autres indications supplémentaires.