DM 1
Langages

Version du 16 septembre 2013

Ce devoir à la maison est à rendre demain, mardi, au début du TD. C'est un vrai devoir, à rédiger proprement sur copie, et à rendre impérativement en TD. Une absence de rendu est une absence tout court ; cela nous évite de faire l'appel. Vous aurez chaque jour un DM à rendre pour le lendemain, et nous estimons qu'ils demandent une à deux heures de travail chacun (relecture du cours incluse). Certains exercices de ces DM peuvent être plus faciles à résoudre en utilisant des résultats qui n'ont pas encore été vus en cours : vous avez le droit (de lire le poly pour prendre de l'avance et) d'utiliser ces résultats.

Exercice 1 – Foire aux opérateurs

Cette question utilise les notations introduites dans la section « Opérations ensemblistes » du polycopié. L'alphabet utilisé est toujours $\Sigma = \{a, b\}$.

Redéfinissez chacun des langages L_i suivants en n'utilisant que des ensembles finis (de mots ou lettres) que vous combinerez avec les opérateurs $\cup$, $\star$, et la concaténation. Les autres opérateurs ($\cap$, $\overline{\ }$, *Suff*, *Pref*, *Fac*...) ne sont pas autorisés.

Par exemple $Pref(\{a\}\{b\}^\star)$ peut se réécrire $\{\varepsilon\} \cup \{a\} \cup \{a\}\{b\}^\star$. On se débarrasse ainsi de l'opérateur *Pref*.

$$
\begin{aligned}
L_1 &= Suff(\{a\}\{b\}^\star) \\
L_2 &= Fac(\{a\}\{b\}^\star) \\
L_3 &= (\{a\}\{b\}^\star\{a\}^\star) \cap (\{a\}^\star\{b\}^\star\{a\}) \\
L_4 &= \overline{\{a\}^\star} \\
L_5 &= \overline{\{a\}\{b\}^\star} \cap \{a\}^\star
\end{aligned}
$$

Exercice 2 – Langage préfixe

On dit que L est un *langage préfixe* si $\forall (u, v) \in L^2, u \neq v \implies u \notin Pref(v)$. Autrement dit si aucun des mots de L n'est préfixe d'un autre mot de L.

Pour deux langages préfixes L_1 et L_2, démontrez (rigoureusement, cela va de soi) que :

1. $\varepsilon \in L_1 \implies L_1 = \{\varepsilon\}$
2. $L_1 \cap L_2$ est préfixe
3. $L_1 L_2$ est préfixe
4. $L_1 \cup L_2$ n'est pas forcément préfixe

Exercice 3 – Digicode pour matheux

Imaginez un digicode équipé de 11 touches : les dix chiffres « `0` »...« `9` » plus une touche « `E` » (comme « Entrée », pour valider la saisie).

Vous devez programmer ce digicode de façon à ce que la porte ne s'ouvre que lorsque les deux conditions suivantes sont remplies :

— la touche « `E` » vient d'être enfoncée

— le nombre formé de *tous* les chiffres tapés avant « `E` » (c'est-à-dire ceux tapés depuis le précédent « `E` » ou depuis que le digicode a été mis en route) est divisible par 7.

Le nombre saisi peut être aussi long que l'on veut, sans limite de taille. Par exemple « `123456789123456789E` » ouvrira la porte, car 123456789123456789 est un multiple de 7. En revanche « `123456789E` » ne doit pas ouvrir la porte. Il n'y a donc pas un seul code qui ouvre la porte, mais une infinité.

À la fin de la semaine, vous serez capable de dire à table une phrase du genre « l'ensemble (infini) de tous les codes acceptés par ce digicode est un langage rationnel, donc il peut être reconnu par une machine avec une mémoire de taille constante » [1]. Bien sûr, il existe des codes acceptés qui sont aussi grands que l'on souhaite, plus grands que la RAM de la machine, par exemple. Il doit donc exister des algorithmes qui n'ont pas besoin de voir tous les chiffres en même temps.

Notre cas est en fait plus contraint : les touches sont frappées une à une, et vous ne connaissez pas à l'avance la taille du nombre qui sera entré. Comme le digicode est implémenté sur un micro-contrôleur avec très peu de RAM, n'imaginez même pas stocker toute la chaîne de caractères et attendre le « `E` » pour la parcourir à nouveau. Il vous faut traiter les chiffres au fur et à mesure qu'ils arrivent, et trouver [2] un moyen de n'utiliser que quelques octets (pas plus de deux ou trois variables) pour retenir l'état du digicode. Le peu de RAM exclut aussi les appels récursifs [3].

Complétez le squelette suivant, représentant le programme exécuté par le digicode. Il est donné en C, mais vous avez le droit d'utiliser un autre langage tant que vous en conservez le principe.

```
/* variables globales (à compléter si besoin) */
...


for (;;) /* boucle infinie */
{
   /* get_key() attend une touche puis renvoie une valeur
      entre 0 et 9, ou -1 pour "entrée" */
   int key = get_key();

   if (key == -1)
     {
        if (/* condition à remplir */)
          {
            open_door();
          }
        /* dans tous les cas, les prochains
           chiffres font partie d'un nouveau nombre */

        ...
     }
   else /* 0 <= key <= 9 */
     {
        /* assimiler le nouveau chiffre */

        ...
     }
}
```

1. Ou l'inverse. De toute façon, vos convives vous auront déjà pris pour un fou.
2. Ça implique de commencer par chercher, mais les Epitéens sont forts pour cela.
3. Ce serait une façon cachée d'utiliser de la mémoire dynamique, donc non bornée.