Algorithme de Hopcroft : correction et terminaison

On se donne un automate déterministe, complet et émondé $\mathcal{A} = (Q, A, E, I, F)$. La preuve qui suit est tirée de [1], à part pour l'invariant de boucle.

Terminaison. À chaque itération du `while`, soit $\mathcal{P}$ est raffinée, soit S diminué d'un élément : ces opérations ne peuvent arriver qu'un nombre fini de fois.

Définition (*congruence*). Une **congruence** est une relation d'équivalence $\sim$ sur Q qui vérifie pour $q, q' \in Q$ et $a \in A$:

$$\begin{aligned} q \sim q' &\implies (q \in F \Leftrightarrow q' \in F), \\ q \sim q' &\implies q \cdot a \sim q' \cdot a \end{aligned}$$

Définition (*stabilité*). Soit $a \in A$, et $B, C \subset Q$. Alors B est stable pour (C, a) ssi $B \cdot a \subset C$ ou $B \cdot a \cap C = \phi$. Sinon (C, a) coupe B en $B_1 = \{q \in B \mid q \cdot a \in C\}$ et $B_2 = \{q \in B \mid q \cdot a \notin C\}$. Une partition de Q sera dite stable pour (C, a) si chacune de ses parts l'est.

Remarque. $\{P_1, \ldots, P_k\}$ partition de Q compatible avec F est une congruence ssi elle est stable pour chaque $(P_i, a), i \in \{1, \ldots, k\}, a \in A$.

Lemme 1. Supposons $B \subset Q$, et $C = C_1 \sqcup C_2$. Alors B est stable pour (C_1, a) et (C, a) implique B est stable pour (C_2, a).

Lemme 2. Si B est stable sous (C, a), alors tout $P \subset B$ aussi, donc en particulier ses fils s'il y a une coupure.

Correction. Remarquons tout d'abord que la congruence de Nérode raffine toujours $\mathcal{P}$: en effet, toutes les coupures sont nécessaires. Reste à montrer qu'on en a suffisamment, c'est-à-dire qu'à la fin $\mathcal{P}$ est une congruence, et celle de Nérode étant la plus grossière on pourra conclure.

On remarque qu'à chaque itération de la boucle `while`, une partie de $\mathcal{P}$ est laissée identique ou coupée en deux, donc $\mathcal{P}$ est l'ensemble des feuilles d'un arbre binaire de racine Q, initialement avec juste deux fils F et $Q \setminus F$. Notons R l'ensemble des éléments qui ont été retirés de S. On appellera T l'arbre binaire des parties générées successivement par l'algorithme, qui contient en particulier $\mathcal{P}$ et les parties apparaissant dans $S \cup R$.

Remarquons que $S \cup R$ ne fait que croître au cours de l'algorithme. De plus, toute partie de $\mathcal{P}$ est stable sous R, un élément étant dans R si on l'a déja utilisé pour couper les éléments de $\mathcal{P}$ (la remarque vaut pour les nouveaux fils d'après le lemme 2).

Montrons l'invariant I de la boucle `while` suivant par induction sur le nombre d'étapes : « Si $P \subset Q$ est stable sous $S \cup R$, alors P est stable sous T » (au sens pour tout (C, a), $C \in T$ et $a \in A$).

Remarquons tout d'abord que si I est vérifié, à la fin S est vide donc comme $\mathcal{P}$ est stable sous R, $\mathcal{P}$ est stable sous T donc en particulier sous les $(C, a), C \in \mathcal{P}, a \in A$ donc $\mathcal{P}$ est la congruence de Nérode.

Étape 0. Si $P \subset Q$ est stable sous (F, a) (resp. $(Q \setminus F, a)$), alors comme P est stable sous (Q, a) (évident), d'après le lemme 1, P est stable sous $(Q \setminus F, a)$ (resp. (F, a)).

Induction. On suppose I à une certaine étape. Montrons I à la fin du `while`. À cette fin on fait une autre induction sur le nombre de coupures effectuées par la première boucle `for`. Si aucune coupure n'est effectuée, T (donc $\mathcal{P}$ aussi) et $S \cup R$ ne changent pas, donc l'invariant est toujours vérifié par hypothèse d'induction.

Sinon, on a une coupure de B en B_1 et B_2 pour un $a \in A$ et on va montrer que I est conservé. Si P est stable sous $S \cup R$ il est stable sous $T \setminus \{B_1, B_2\}$ (pour tout $b \in A$) par hypothèse d'induction. Pour $b \in A$, si $(B, b) \in S$, alors ses deux fils sont aussi dans S, donc P stable sous T par hypothèse. Sinon par exemple (B_1, b) est dans S, et on peut appliquer le lemme 1 à P pour (B, b) et (B_1, b) : P est donc stable sous (B_2, b) donc sous T entier, ce qui achève la preuve.

Références

[1] O.Carton, *Langages formels, Calculabilité et Complexité.*