4.9 Implémentation de l'algorithmique de Moore

Cette section est inspirée de [David, 2010]. On suppose que les états sont codés par des entiers dans $\{1, \dots, n\}$. L'alphabet Σ est noté $\{a_1, \dots, a_k\}$.

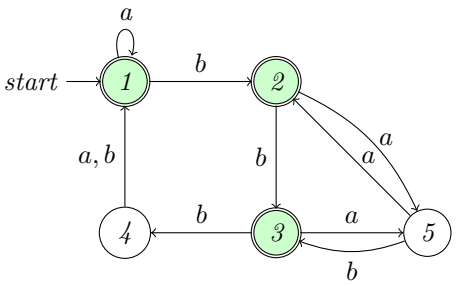
Type abstrait : relation d'équivalence

— Construire une relation d'équivalence

— Raffiner selon la propriété pour toute lettre a, $q \sim q'$ et $\delta(q, a) \sim \delta(q', a)$.

Implémentation : on implémente la relation d'équivalence par un tableau d'entiers π. On numérote les classes d'équivalence à partir de 1 et la case $\pi[q]$ contient le numéro de la classe d'équivalence de $q \in \{1, \dots, n\}$.

Example 44 *Considérons l'automate suivant :*



La partition $\{\{1, 2, 3\}, \{4, 5\}\}$ *est représentée par le tableau* π *suivant :*

2	2	2	1	1

En plus, on stocke le nombre total de classes dans $\pi.num$.

Example 45 *Ici* $\pi.num = 2$.

Voici le pseudo-code détaillé du raffinement avec cette structure de données :

Précondition : un automate $\mathcal{A}$ un automate déterministe complet où tous les états sont accessibles depuis l'état initial, un tableau π qui représente une relation $\sim_i$;
Postcondition : **return** $(\pi', change?)$ où π' représente la relation $\sim_{i+1}$ et $change?$ est un booléen vrai ssi $\sim_i \neq \sim_{i+1}$.

```
function raffiner(A, π)
    Construction de la signature
    for q = 1 à n do
        s[q] := (π[q], π[δ(q, a_1)], ..., π[δ(q, a_k)])
                                                    où Σ = {a_1, ..., a_k}
    endFor

    Tri radix
    σ := tableau des états q triés selon s[q] avec l'ordre lexicographique

    Renumérotation
    num = 1
    π'[σ(1)] := num
    for ℓ = 2 à n do
        if s[σ[ℓ]] ≠ s[σ[ℓ − 1]] then
            num := num + 1
        endIf
        π'[σ(ℓ)] := num
    endFor
    π'.num := num
    change? := (π.num ≠ num)
    return (π', change?)
endFunction
```

4.9.1 Construction de la signature

On appelle signature de q le $(1 + |\Sigma|)$-uplet $(\pi[q], \pi[\delta(q, a_1)], \dots, \pi[\delta(q, a_k)])$. La signature reflète le fait d'être dans la même classe et que les a-successeurs sont dans la même classe pour toute lettre a.

La première boucle calcule la nouvelle $s[q]$ des états q. Si δ est codé avec une matrice d'adjacence, cette première boucle est en $O(|\Sigma|n)$.

Example 46 *On a :*
$s[1] = (2, 2, 2)$
$s[2] = (2, 1, 2)$
$s[3] = (2, 1, 1)$
$s[4] = (1, 2, 2)$
$s[5] = (1, 2, 2)$

4.9.2 Tri radix

Mais maintenant, il nous faut une signature avec des nombres pour représenter π' et non pas des tuples. C'est pourquoi on trie les éléments par ordre lexicographique.

Example 47 *Sur l'exemple, le tri donne l'ordre croissant suivant : 4, 5, 3, 2, 1.*

Le tri lexicographique peut s'implémenter en

$$O(\text{longueur des mots} \times (\text{nombre d'éléments} + \text{nombres de chiffres})$$

et un espace supplémentaire de mémoire égal au nombre de chiffres (voir un cours d'algorithmique). Ici :
— la longueur des mots est $1 + |\Sigma|$ où Σ est l'alphabet ;
— Le nombre de chiffres est le nombre de classes d'équivalence. Il est majoré par n ;
— Les éléments à trier sont les états. Ainsi, le nombre d'éléments est n et le nombre maximum de composantes soit n.
Ainsi, ce tri est en $O(|\Sigma|n)$.

4.9.3 Renumérotation

Enfin, on renumérote les classes d'équivalence. Le test $s[q] \neq s[q-1]$ s'effectue en comparant les tuples $s[q]$ et $s[q-1]$ composante par composante. On suppose que la comparaison d'une composante avec une autre est en $O(1)$ car c'est un nombre entre 1 et n. Ainsi, la comparaison $s[q] \neq s[q-1]$ coûte $O(|\Sigma|)$. Donc le numérotage coûte $O(|\Sigma|n)$.

Example 48 *Sur l'exemple, on obtient un ordre, par exemple :* $4, 5, 3, 2, 1$. *C'est à dire*

$$\sigma[1] = 4,\ \sigma[2] = 5,\ \sigma[3] = 3,\ \sigma[4] = 2 \text{ et } \sigma[5] = 1.$$

Du coup, on numérote les classes :
$\pi'[4] := 1$
$\pi'[5] := 1$
$\pi'[3] := 2$
$\pi'[2] := 3$
$\pi'[1] := 4$
C'est à dire π' est le tableau :

4	*3*	*2*	*1*	*1*

4.9.4 Bilan

Proposition 24 *Un appel raffiner($\mathcal{A}$, π) coûte $O(|\Sigma|n)$.*

Proposition 25 *L'algorithme de minimisation automateMinimal($\mathcal{A}$) peut s'implémenter en $O(|\Sigma| \times n^2)$.*

Proof.
On a une boucle qui calcule à chaque étape $\sim_i$ pour i allant de 0 à jusqu'à au plus $n-2$. Ainsi, il y a au plus $O(n)$ calcul de raffinement. Chaque raffinement coûte $O(|\Sigma|n)$. D'où $O(|\Sigma| \times n^2)$. ■