Langages formels (11X003) - Automne 2025

3. Non déterminisme

Enseignant: Arnaud Casteigts

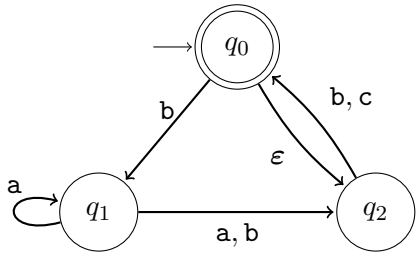
Assistants: A.-Q. Berger & M. Marseloo
Moniteurs: N. Beghdadi & E. Bussod

3.1 Automates finis non déterministes

Les automates vus précédemment (AFD) ont une fonction de transition complètement déterministe : à partir d'un état et d'un symbole, on ne peut se déplacer que sur un état. Il est possible de généraliser ce comportement en autorisant l'automate à se déplacer sur plusieurs états simultanément, c'est ce que l'on appelle le **non déterminisme**.

Dans un tel automate, l'ensemble d'arrivée de la fonction de transition δ n'est donc plus Q, mais l'ensemble des *parties* de Q, noté $\mathcal{P}(Q)$ (ou parfois 2^Q). Autrement dit, pour un état de départ et un symbole donnés, on se déplace sur un *sous-ensemble* de Q. Une autre différence est qu'il est possible d'emprunter des transitions spéciales, appelées **ε-transitions**, sans lire le moindre symbole.

Voici un exemple d'automate fini non déterministe (AFN), qui a une ε-transition de q_0 vers q_2 et *deux* transitions sortantes de q_1 pour un même symbole `a`.



Le non déterminisme peut être interprété de plusieurs manières. L'une est d'imaginer que l'exécution "décide" parmis les choix possibles, chacun engendrant un univers parallèle dans lequel on continue à être sur un seul état. Une autre est d'imaginer que l'on se dédouble en plusieurs copies en restant dans le même univers (plus intuitif). Observons ce qu'il se passe en lisant le mot `baba`. Initialement, nous sommes en q_0 et nous nous dédoublons instantanément sur q_2, nous sommes donc sur les états $\{q_0, q_2\}$. En lisant le symbole `b`, la copie qui était sur q_0 part en q_1 et la copie sur q_2 part en q_0. Cette dernière se dédouble à nouveau pour retourner sur q_2 via l'ε-transition, nous sommes donc sur les états $\{q_0, q_1, q_2\}$. En lisant `a`, les copies sur q_0 et q_2 disparaissent, car il n'y a aucune transition pour `a` sur ces états, mais la copie sur q_1 se dédouble à nouveau sur q_1 (via la boucle) et sur q_2 (via l'autre transition),

nous sommes donc sur $\{q_1, q_2\}$. En continuant ainsi, nous nous retrouvons sur $\{q_0, q_2\}$ après avoir lu `b`, puis la lecture de `a` fait disparaitre toutes nos copies (état $\emptyset$). Le mot `baba` est finalement **rejeté**. Un mot est **accepté** (ou **reconnu**) par un tel automate si au moins une copie (c.à.d. une exécution possible) se trouve sur un état final à la fin de la lecture. Par exemple, le mot `bab` est accepté.

3.1.1 Définition formelle

Un **automate fini non déterministe** (AFN) est un 5-tuple $(Q, \Sigma, \delta, q_0, F)$ où

- Q est un ensemble fini d'états,
- Σ est un alphabet,
- $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ est la fonction de transition (où $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$)
- q_0 est l'état initial,
- F est l'ensemble des états finaux avec $F \subseteq Q$,

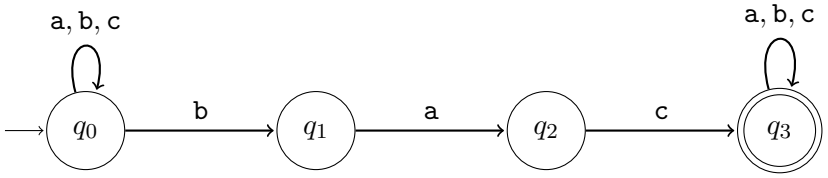
La seule différence avec les AFDs vient donc de la fonction de transition δ. En fait, les AFDs que nous connaissions sont des cas particuliers d'AFNs, qui n'ont pas ε-transitions et où δ renvoit toujours un singleton. Comme pour les AFDs, on peut ici étendre δ en une fonction $\delta^* : Q \times \Sigma^* \to \mathcal{P}(Q)$ qui indique comment l'état évolue en lisant *plusieurs* symboles d'affilée. Par exemple, ici, $\delta^*(q_0, \texttt{ba}) = \{q_1, q_2\}$, $\delta^*(q_0, \texttt{baba}) = \emptyset$, ou encore $\delta^*(q_1, \texttt{aa}) = \{q_1, q_2\}$.

Un mot $w \in \Sigma^*$ est **accepté** si et seulement si $\delta^*(q_0, w) \cap F \neq \emptyset$. Le **langage reconnu** par un AFN A est l'ensemble des mots acceptés $L(A) = \{w \in \Sigma^* | \delta^*(q_0, w) \cap F \neq \emptyset\}$.

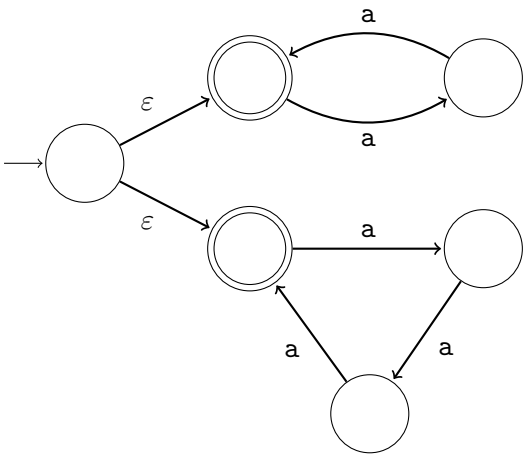
Remarque 3.1. Pour simplifier le raisonnement, il est souvent utile de considérer une version encore plus générale de δ, dont le domaine lui-même est aussi $\mathcal{P}(Q)$, à savoir $\delta : \mathcal{P}(Q) \times \Sigma_\varepsilon \to \mathcal{P}(Q)$. Cette version de δ nous envoit directement d'un sous-ensemble vers un autre sous-ensemble (via un symbole). La même remarque tient pour δ^*. Nous utiliserons l'une ou l'autre de manière implicite, sans que cela n'apporte de confusion.

3.1.2 Exemples

Les AFNs sont souvent plus pratiques que les AFDs, car ils permettent d'exprimer plus simplement le traitement à réaliser. Voici un exemple d'AFN sur l'alphabet $\{\texttt{a}, \texttt{b}, \texttt{c}\}$ qui reconnaît tous les mots contenant le facteur `bac` :



On peut voir que le facteur recherché apparaît directement dans la construction. Grâce au non déterminisme, c'est un peu comme si l'automate pouvait **deviner**, en voyant un `b`, s'il s'agit du début du facteur `bac`. Voici un autre exemple, qui reconnaît les mots dont la longueur est un multiple de deux ou un multiple de trois sur un alphabet unitaire $\Sigma = \{a\}$ (un alphabet est dit unitaire s'il n'a qu'un symbole).



Dans cet exemple, les deux cas (multiple de deux ou multiple de trois) peuvent être traités indépendamment, ce qui est très pratique. Un AFD reconnaissant le même langage serait plus compliqué à construire. D'ailleurs, peut-on toujours trouver un AFD équivalent ?

3.2 Déterminisation d'un AFN

Oui, on peut. Dans cette section, nous allons donner un algorithme qui transforme n'importe quel AFN $A = (Q, \Sigma, \delta, q_0, F)$ en un AFD $A' = (Q', \Sigma, \delta', q'_0, F')$ tel que A et A' sont **équivalent**, c'est à dire qu'ils reconnaissent exactement le même langage.

L'idée principale est de créer un état de A' pour chaque sous-ensemble d'états de A qui est effectivement atteignable lors de l'exécution, puis de relier ces nouveaux états de A' par les transitions correspondantes. Nous présentons cet algorithme en l'illustrant sur l'exemple du premier automate de ce chapitre, répété ici par commodité.

Comme discuté plus haut, dès le début de l'exécution de A, nous nous trouvons dans l'état $\{q_0, q_2\}$ car q_0 est l'état initial de A et il existe une ε-transition de q_0 vers q_2. Nous allons donc créer l'état initial q'_0 pour l'automate A', qui correspondra à l'ensemble $\{q_0, q_2\}$

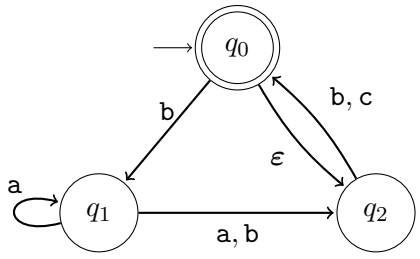


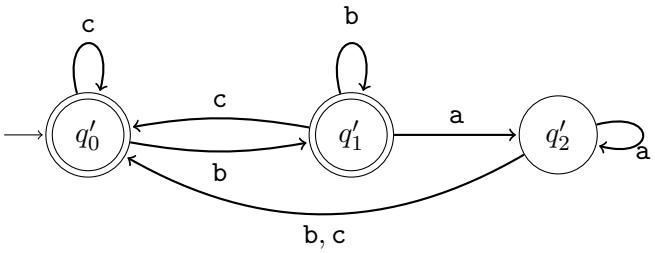
FIGURE 1 – Automate non déterministe (AFN) que nous allons transformer en AFD.

de l'automate A. Puis, nous commençons une exploration méthodique des ensembles d'états atteignable dans A. Cela se fait en notant, pour chaque symbole de Σ, l'ensemble des états où l'on se retrouve après avoir lu le symbole correspondant. Si cette combinaison est nouvelle, nous créons un nouvel état de A' qui lui correspond et qui sera ensuite explorée de la même manière, jusqu'à ce que toutes les combinaisons atteignables aient été traitées.

Cela est résumé par les transitions suivantes :

États de A	État de A'	Symbole	États de A	État de A'
$\{q_0, q_2\}$	q'_0	a	$\emptyset$	état puits (ignoré)
		b	$\{q_0, q_1, q_2\}$	q'_1 (création)
		c	$\{q_0, q_2\}$	q'_0
$\{q_0, q_1, q_2\}$	q'_1	a	$\{q_1, q_2\}$	q'_2 (création)
		b	$\{q_0, q_1, q_2\}$	q'_1
		c	$\{q_0, q_2\}$	q'_0
$\{q_1, q_2\}$	q'_2	a	$\{q_1, q_2\}$	q'_2
		b	$\{q_0, q_2\}$	q'_0
		c	$\{q_0, q_2\}$	q'_0

Ainsi, l'ensemble Q' des états de notre nouvel AFD A' consiste en q'_0, q'_1 et q'_2, qui correspondent à toutes les combinaisons d'états atteignables dans A, à savoir $\{q_0, q_2\}$, $\{q_0, q_1, q_2\}$ et $\{q_1, q_2\}$. Le fait que A' ait le même nombre d'états que A ici est un hasard, en pratique il arrive souvent qu'il ait plus (voir beaucoup plus) d'états dans l'AFD que dans l'AFN de départ. Quels doivent être les états finaux F' ? Rappelons-nous qu'un AFN accepte un mot du moment qu'*au moins une* des exécutions possibles se termine sur un état final. Les états finaux de A' doivent donc être tous les états de Q' qui "contiennent" l'état final de A (à savoir q_0), donc q'_0 et q'_1, autrement dit $F' = \{q'_0, q'_1\}$. Enfin, la fonction de transition δ' correspond directement au tableau que nous avons créé. Nous obtenons donc l'AFD suivant :



Par construction, l'automate fini déterministe A' simule exactement le comportement de l'automate fini non-déterministe A, il est donc facile de se convaincre que A' acceptera un mot si et seulement si A l'accepte aussi : les deux automates sont équivalents.

En conclusion, bien que les AFNs semblent plus puissants au premier abord, ils ont en fait la même puissance que les AFDs : ils reconnaissent, eux aussi, ni plus ni moins que les *langages réguliers*. Ils sont cependant plus pratiques à utiliser et ont aussi l'avantage d'avoir un plus petit nombre d'états (en général). Nous les utiliserons la semaine prochaine pour démontrer qu'en effet, ces automates peuvent reconnaître *tous* les langages réguliers.

Mais alors, le non-déterminisme, les univers parallèles, etc., ça ne rend pas une machine plus puissante ? Si, c'est le cas pour des modèles de machines plus élaborées, mais pas pour les automates finis.