Langages formels (11X003) - Automne 2023

11. Machines de Turing (divers)

Enseignant: Arnaud Casteigts *Assistant: Alexandre-Quentin Berger*

Dans ce cours, nous allons voir différentes notions liées aux machines de Turing, sans lien particulier. Nous présentons d'abord (rapidement) deux types de grammaires plus générales que les grammaires hors-contexte. Puis nous donnons un exemple de machine de Turing non-déterministe. Nous donnons ensuite plusieurs exemples de machines de Turing qui ne font pas que reconnaître des mots, elles calculent aussi et produisent une sortie ! Enfin, nous discutons brièvement de la thèse de Church-Turing, qui lie ces machines à nos ordinateurs.

11.1 Grammaires

11.1.1 Grammaire contextuelle

Une grammaire $G = (V, \Sigma, \mathcal{P}, S)$ est **contextuelle** si toutes les règles de production de $\mathcal{P}$ sont de la forme $S \to \varepsilon$ (on peut produire le mot vide depuis la variable de départ) ou de la forme $\alpha X \beta \to \alpha\gamma\beta$, où α, β et γ sont des mots quelconques de $(V \cup \Sigma)^*$, X est une variable de V et $\gamma \neq \varepsilon$ (d'où la nécessité de permettre aussi la règle $S \to \varepsilon$). La nouveauté principale ici (en comparaison des grammaires hors-contexte), est que l'activation d'une règle pour remplacer une variable X peut dépendre de ce qui se trouve autour : son *contexte*. Le contexte lui-même n'a pas le droit de changer, mais cela donne déjà plus de puissance. Par exemple, le langage $L = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 1\}$ peut être engendré par la grammaire suivante :

1. $S \to \mathtt{a}BC$
2. $S \to \mathtt{a}SBC$
3. $CB \to CZ$
4. $CZ \to WZ$
5. $WZ \to WC$
6. $WC \to BC$
7. $\mathtt{a}B \to \mathtt{ab}$
8. $\mathtt{b}B \to \mathtt{bb}$
9. $\mathtt{b}C \to \mathtt{bc}$
10. $\mathtt{c}C \to \mathtt{cc}$

Les deux premières règles permettent de générer $\mathtt{a}^n(BC)^n$. Notez qu'à ce stade, les variables B et C se retrouvent entrelaçées. Les règles 3 à 6 permettent d'échanger successivement chaque motif CB en BC (ce qui ne peut pas être fait en une seule règle dans le format imposé) ; Enfin, les règles 7 à 10 permettent de remplacer les variables B et C par les symboles terminaux correspondant (`b` et `c`), à condition qu'ils soient à la bonne place (grâce au contexte !).

Voyons comment produire le mot `aabbcc` (les numéros des règles utilisées sont inscrits à côté de chaque flèche) :

```
S
→2  aSBC
→1  aaBCBC
→3  aaBCZC
→4  aaBWZC
→5  aaBWCC
→6  aaBBCC
→7  aabBCC
→8  aabbCC
→8  aabbcC
→9  aabbcC
→10 aabbcc
```

Ces grammaires ont été montrées équivalentes en puissance aux **machines de Turing non-déterministes linéairement bornée**, à savoir, dont la taille de la bande peut être utilisée jusqu'à une distance proportionnelle à la longueur du mot d'entrée (le facteur de proportionalité dépend du langage).

11.1.2 Grammaire générale

Même chose qu'une grammaire contextuelle, mais cette fois-ci le contexte lui-même peut aussi être modifié par les règles. Ces grammaires sont alors parfaitement équivalentes aux **machines de Turing** (sans restriction).

11.1.3 Hiérarchie de Chomsky

La hiérarchie de Chomsky regroupe les équivalences que nous avons vu entre différentes familles de langages (et grammaires correspondantes) et différents modèles de calculs. Les correspondances sont exactes. Les voici :

Grammaire	Langage	Machine
Régulière	Régulier	Automate fini
Hors-contexte	Hors-contexte	Automate à pile (non-déterministe)
Contextuelle	Contextuel	MT non-déterministe linéairement bornée
Générale	Turing-reconnaissable	Machine de Turing

Note : Pour des raisons historiques, les langages Turing-reconnaissables sont aussi appelés langages **récursivement énumérable**.

11.2 Machines de Turing non-déterministes

L'expressivité des machines de Turing non-déterministes n'est pas supérieure à celle des machines de Turing déterministes, car ces dernières peuvent les **simuler**. Notez que cette équivalence ne fonctionne que si la mémoire n'est pas bornée (d'où l'importance de préciser dans le tableau ci-dessus, pour les langages contextuels). Le non-déterminisme permet cependant d'exprimer les choses plus simplement. Par ailleurs, les machines non-déterministes (à supposer qu'elles existent physiquement, ce qui n'est pas le cas) sont plus rapides lorsqu'on s'intéresse à la complexité.

Comme pour les automates finis ou les automates à pile, le non-déterminisme peut être vu comme le fait de choisir plusieurs actions simultanément, à savoir un sous-ensemble des actions possibles, d'où la fonction de transition qui aura pour signature :

$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$

Exemple

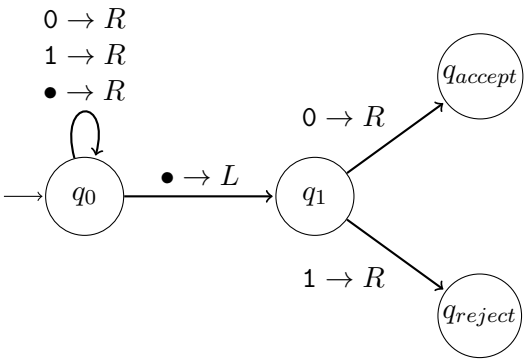
Voyons un exemple d'une telle machine : on reçoit en entrée une liste de mots binaires séparés par des • et on veut déterminer si au moins l'un d'entre eux est pair (termine par `0`). Par simplicité, on supposera qu'un point supplémentaire est présent après le dernier mot, par exemple, voici une entrée possible :

`01101 • 0110 • 1010101 • 100100 •`

Une machine déterministe pour ce problème examinerait chaque mot séquentiellement. Avec le non-déterminisme, on peut créer différentes branches de calcul qui examinent chacune un mot en parallèle. Plus précisément, lorsque le premier point est atteint, une branche non-déterministe va examiner ce mot, tandis que la branche d'origine va se diriger directement vers le point suivant. La machine acceptera à condition qu'*au moins une* de ces branches atteigne l'état q_{accept}. À l'intérieur de chaque branche, toutes les transitions qui ne sont pas

spécifiées explicitement sont supposées mener à l'état q_{reject} (un peu comme l'état puits dans les automates finis).

Voici la machine correspondante :

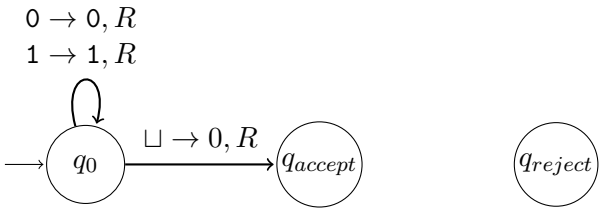


11.3 Calculer avec une machine de Turing

Contrairement aux automates finis ou aux automates à pile, les machines de Turing sont capables de faire d'autres choses que de répondre OUI ou NON. Elles peuvent calculer et produire des choses en sortie, comme nos ordinateurs.

Exemple 1 : multiplier un nombre (binaire) par deux

Description textuelle : aller au bout du mot, rajouter un 0, puis terminer.



Remarque : L'état q_{reject} n'est pas utilisé ici, mais on pourrait lui donner un sens, par exemple similaire à celui des *exceptions* en Java.

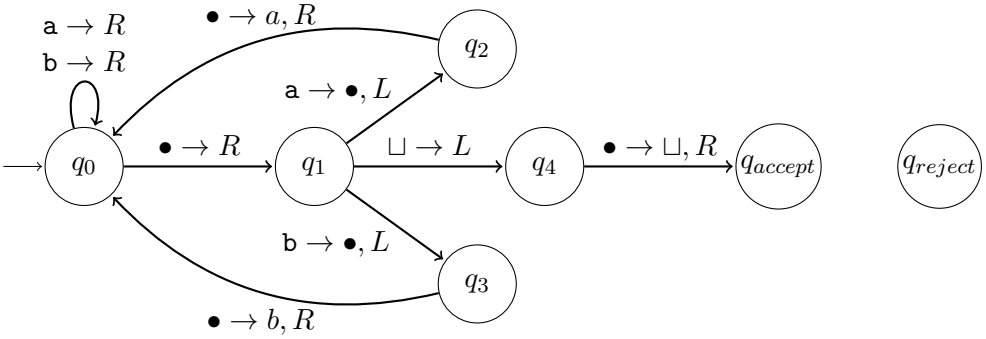
Exemple 2 : concaténer deux mots

On souhaite concaténer deux mots en entrée, disons sur l'alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \bullet\}$, où $\bullet$ est le symbol séparateur entre les deux mots.

Par exemple : `abba` • `bab` doit produire `abbabab`.

Idée générale : décaler le deuxième mot d'un cran vers la gauche.

Description textuelle : on commence par aller vers la droite jusqu'au séparateur. Ensuite on répète : passer le séparateur, remplacer le symbole courant par un deuxième séparateur en reculant d'un cran à gauche, réécrire ce symbole à la place du premier séparateur en avançant d'un cran à droite. Si un espace est rencontré, effacer le dernier point et terminer.



Exemple 3 : additionner deux nombres binaires

→ En séance d'exercices. La principale difficulté sera de gérer la retenue.

Bande de sortie

Contrairement à ces exemples, il est souvent d'usage d'utiliser une bande de sortie lorsque la machine ne produit des sorties. Pour le premier exemple, une telle machine recopierait l'entrée sur la bande de sortie à mesure qu'elle la lit, puis y ajouterait le `0`, sans modifier la bande d'entrée. Pour le second exemple, le comportement serait en fait plus simple, puisqu'il n'y aurait plus besoin de décaler le deuxième mot, juste de recopier les deux mots en s'abstenant de recopier le séparateur. Rappelons-nous de la fonction de transition si l'on utilise plusieurs bandes, ici deux :

$\delta : Q \times \Gamma \times \Gamma \to Q \times \Gamma \times \Gamma \times \{L, R\} \times \{L, R\}$

Pour représenter cela graphiquement, chaque transition doit indiquer ce qu'elle doit lire et/ou écrire sur les deux bandes (et comment s'y déplacer). Plusieurs notations existent. Par exemple, on pourrait écrire :

$\mathtt{a}, \mathtt{b} \to \mathtt{c}, \mathtt{d}[R, L]$

sur une transition, pour signifier qu'on doit lire `a` sur la première bande et `b` sur la seconde afin d'effectuer cette transition, puis on écrit `c` sur la première et `d` sur la seconde,

en se deplaçant ensuite vers la droite sur la première bande et vers la gauche sur la seconde. Les crochets ne servent ici qu'à séparer les déplacement du reste, pour plus de lisibilité.

11.4 Thèse de Church-Turing

Voir le cours n°12.