Calculabilité & Complexité

Calculabilité (1/5) : introduction

Nicolas Ollinger (LIFO, Université d'Orléans)

M1 info, Université d'Orléans — S2 2025/2026

Organisation

Cours	$9 \times$ 1h30 + 30min	N. Ollinger	
TD	$5 \times$ 2h	M. Delacourt	← *calculabilité*
TD	$5 \times$ 2h	M. Liedloff	← *complexité*

Support de cours :
`https://celene.univ-orleans.fr/course/view.php?id=2106`

Évaluation : Note Finale $= \frac{1}{4}$CC $+ \frac{3}{4}$CT

CT : examen de 2h sur **Cours** + **TD**

CC : Contrôles rapides sur le temps du TD.

Méthode de travail

Le contenu de ce cours est passionnant mais **très formel**. Ne restez pas passifs, **prenez des notes**, relisez-les, posez des questions !

Lisez aussi **avant de venir en cours** :

- en calculabilité, le polycopié disponible sur Celene ;
- en complexité, les chapitres du livre de S. Périfel.

Résolvez et comprenez les exercices de TD !

Des références bibliographiques supplémentaires sont disponibles sur Celene, si besoin.

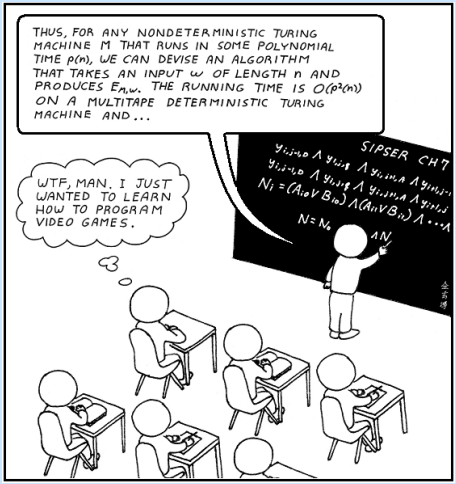
Objet du cours

Partie I. Calculabilité

« Que peut-on calculer avec un ordinateur ? »

Partie II. Complexité

*« Que peut-on calculer **efficacement** avec un ordinateur ? »*



1. Calculabilité

Calculabilité

Nous présentons quelques éléments de **calculabilité** autour de **modèles de calcul classiques**.

Un bagage classique de M1 informatique...

Calculabilité

Nous présentons quelques éléments de **calculabilité** autour de **modèles de calcul classiques**.

Un bagage classique de M1 informatique. . . bientôt en lycée ?

Contenus	Capacités attendues	Commentaires
Notion de programme en tant que donnée. Calculabilité, décidabilité.	Comprendre que tout programme est aussi une donnée. Comprendre que la calculabilité ne dépend pas du langage de programmation utilisé. Montrer, sans formalisme théorique, que le problème de l'arrêt est indécidable.	L'utilisation d'un interpréteur ou d'un compilateur, le téléchargement de logiciel, le fonctionnement des systèmes d'exploitation permettent de comprendre un programme comme donnée d'un autre programme.

extrait du programme de NSI

Calculabilité

Nous présentons quelques éléments de **calculabilité** autour de **modèles de calcul classiques**.

Un bagage classique de M1 informatique... bientôt en lycée ?

Contenus	Capacités attendues	Commentaires
Notion de programme en tant que donnée. Calculabilité, décidabilité.	Comprendre que tout programme est aussi une donnée. Comprendre que la calculabilité ne dépend pas du langage de programmation utilisé. Montrer, sans formalisme théorique, que le problème de l'arrêt est indécidable.	L'utilisation d'un interpréteur ou d'un compilateur, le téléchargement de logiciel, le fonctionnement des systèmes d'exploitation permettent de comprendre un programme comme donnée d'un autre programme.

extrait du programme de NSI

Ici ce sera **avec** formalisme théorique !

Thèse de Church, Turing, Kleene, Post *et al.*

Théorème (Gandy 1980) Toute fonction **discrète** calculable par un dispositif **mécanique déterministe**, régit par les règles de la physique classique et qui satisfait les **hypothèses** suivantes, est Turing-calculable : **(...)**

Thèse de Church, Turing, Kleene, Post *et al.*

Théorème (Gandy 1980) Toute fonction **discrète** calculable par un dispositif **mécanique déterministe**, régit par les règles de la physique classique et qui satisfait les **hypothèses** suivantes, est Turing-calculable : **(...)**

Thèse de Church-Turing Toute fonction calculable par une méthode effective est Turing-calculable.

Thèse de Church, Turing, Kleene, Post *et al.*

Théorème (Gandy 1980) Toute fonction **discrète** calculable par un dispositif **mécanique déterministe**, régit par les règles de la physique classique et qui satisfait les **hypothèses** suivantes, est Turing-calculable : **(...)**

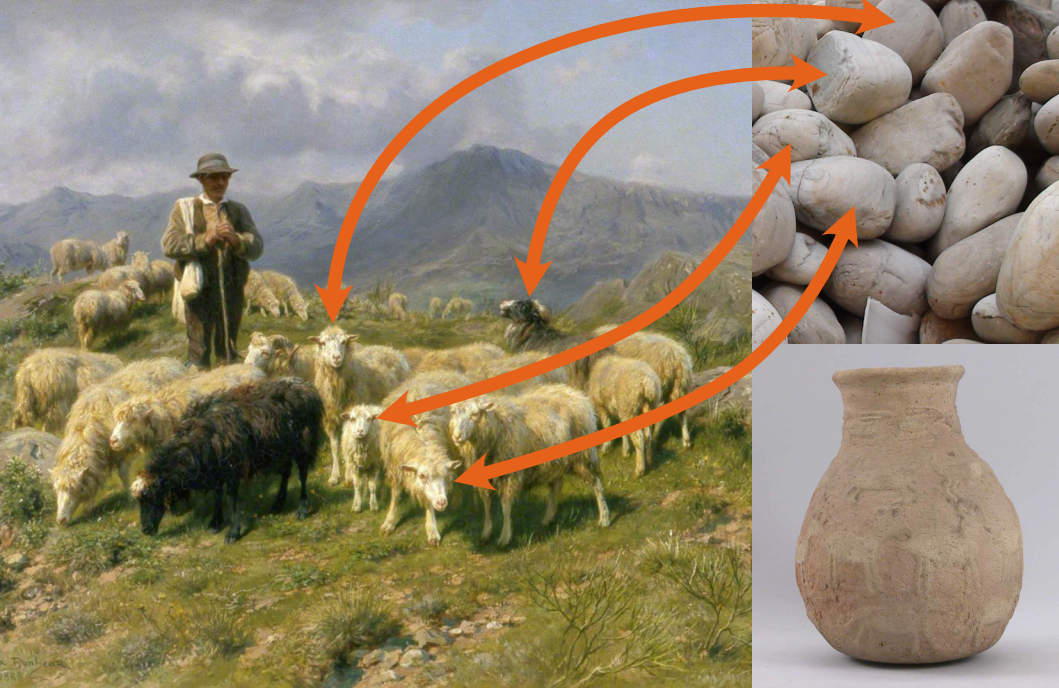
Thèse de Church-Turing Toute problème **décidable** par une méthode effective est Turing-**décidable**.

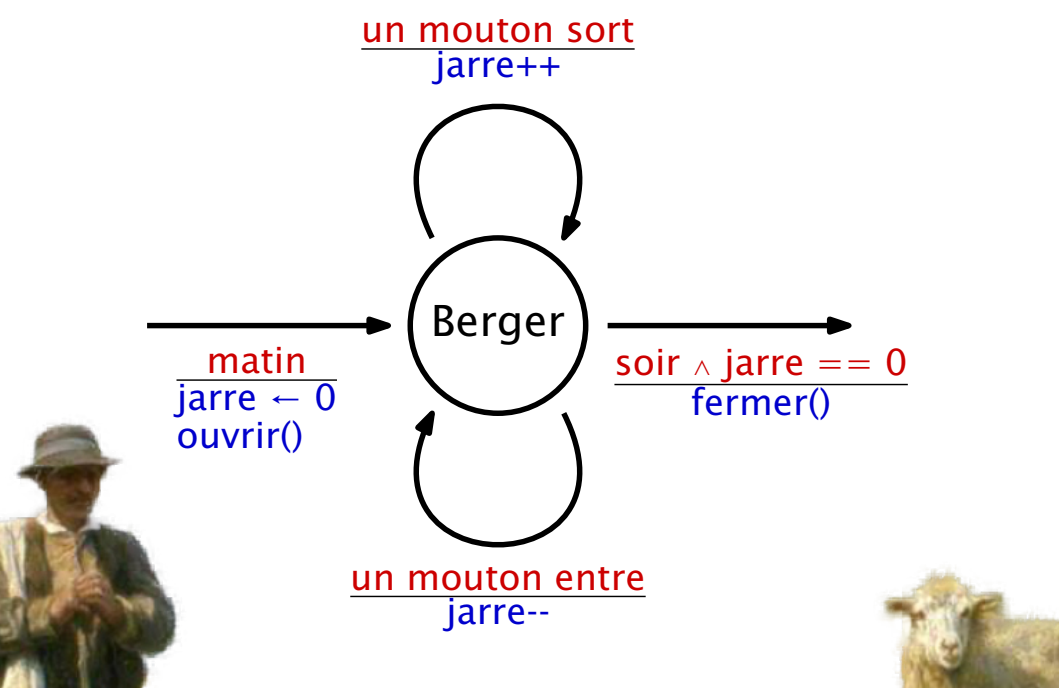
Thèse de Church, Turing, Kleene, Post *et al.*

Théorème (Gandy 1980) Toute fonction **discrète** calculable par un dispositif **mécanique déterministe**, régit par les règles de la physique classique et qui satisfait les **hypothèses** suivantes, est Turing-calculable : **(...)**

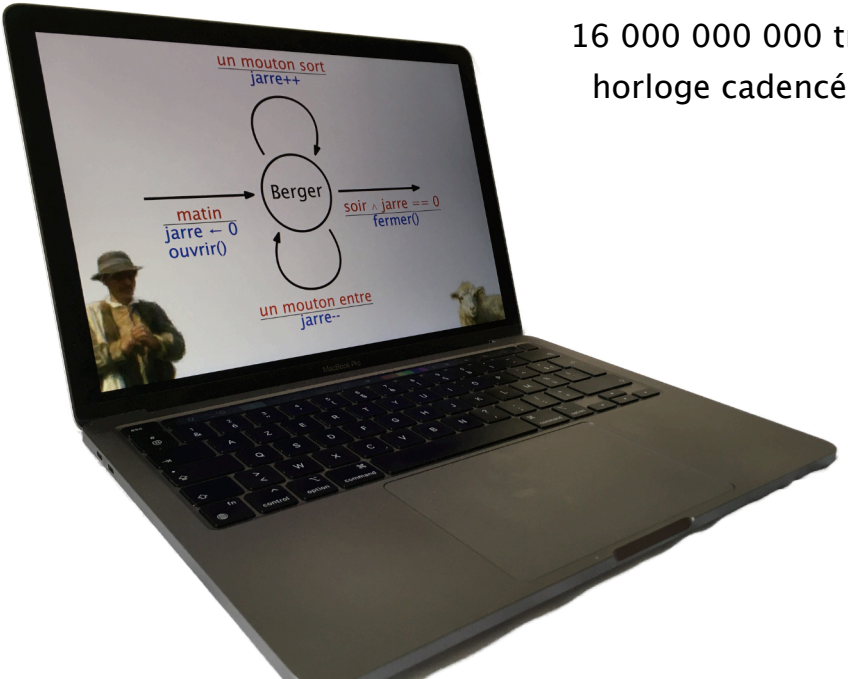
Thèse de Church-Turing Toute problème **décidable** par une méthode effective est Turing-**décidable**.

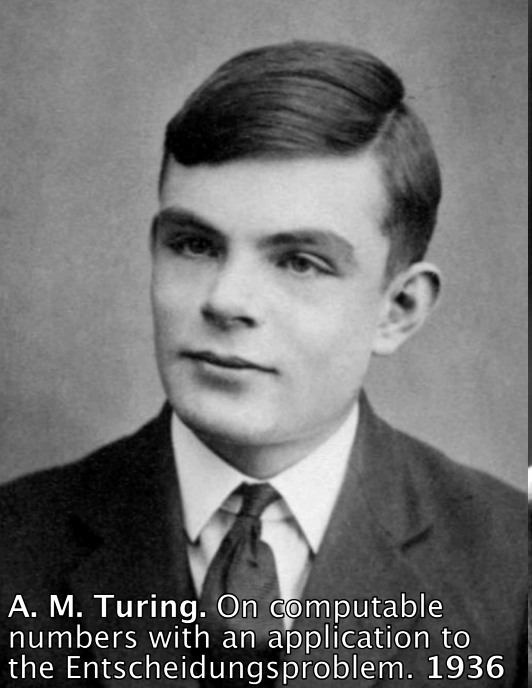
Corollaire Tout problème Turing-**indécidable** est **indécidable** par toute méthode effective.



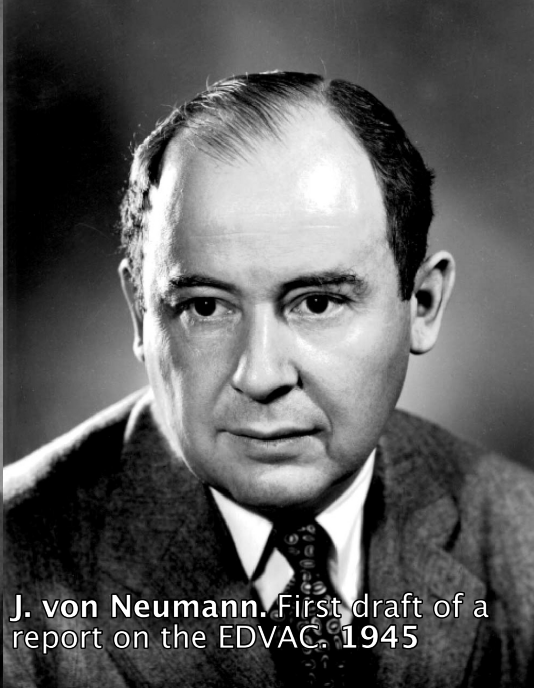

un mouton sort
jarre++
Berger
matin
jarre ← 0
ouvrir()
soir ∧ jarre == 0
fermer()
un mouton entre
jarre--

16 000 000 000 transistors
horloge cadencée à 3 GHz

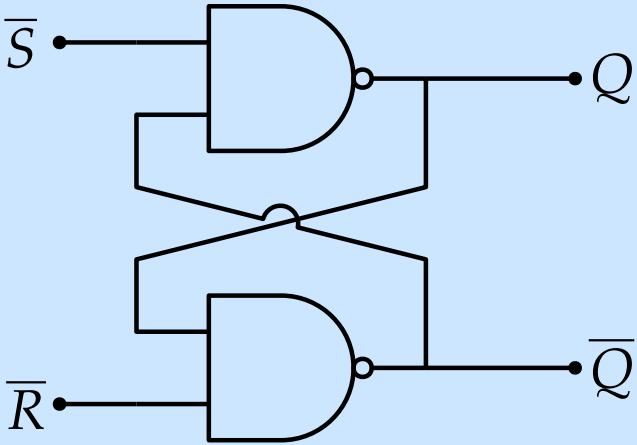




A. M. Turing. On computable numbers with an application to the Entscheidungsproblem. **1936**



J. von Neumann. First draft of a report on the EDVAC. **1945**



2. Machines finies et circuits booléens

Calcul de fonction

$$f : A \to B$$

Calcul de fonction

$$f : \{0,1\}^m \to \{0,1\}^n$$

Calcul de fonction

$$f : \{0,1\}^m \to \{0,1\}^n$$

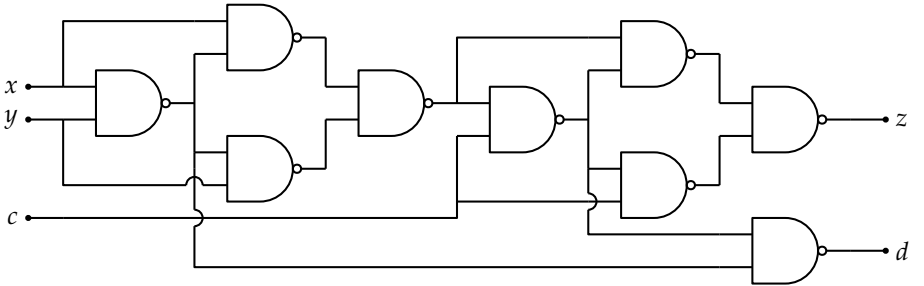
Théorème Toute fonction **binaire** est calculée par un **circuit combinatoire** comportant des portes **ET**, **OU** et **NON**.

Calcul de fonction

$$f : \{0,1\}^m \to \{0,1\}^n$$

Théorème Toute fonction **binaire** est calculée par un **circuit combinatoire** comportant des portes **ET**, **OU** et **NON**.

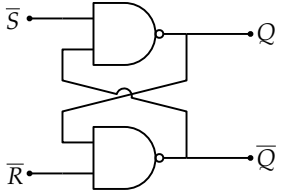
Théorème Toute fonction **binaire** est calculée par un **circuit combinatoire** comportant uniquement des portes **NON-ET**.



Addition de trois bits avec neuf portes NON-ET

Circuits séquentiels

Que se passe-t-il lorsque le **graphe du circuit** comporte des **cycles** ?

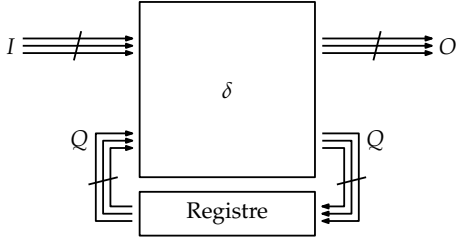


On obtient des **circuits séquentiels** dans lesquels les sorties dépendent des **états passés** :

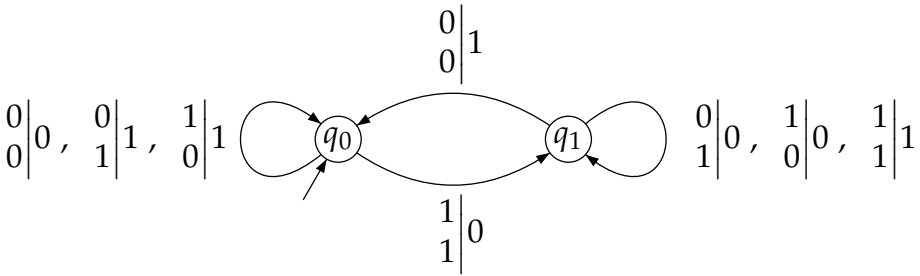
$$y(t+1) = F(x(t), y(t))$$

Automates de Mealy

Définition Un **automate de Mealy** est un tuple (Q, I, O, q_0, δ) où Q est l'ensemble fini des **états**, I et O sont les **alphabets** finis d'entrée et de sortie, $q_0 \in Q$ est l'**état initial** et $\delta : Q \times I \to Q \times O$ est la **fonction de transition** de l'automate.



Théorème Tout **automate de Mealy** peut être mis en œuvre par un **circuit séquentiel synchrone** composé de portes NON-ET et de registres mémoire.



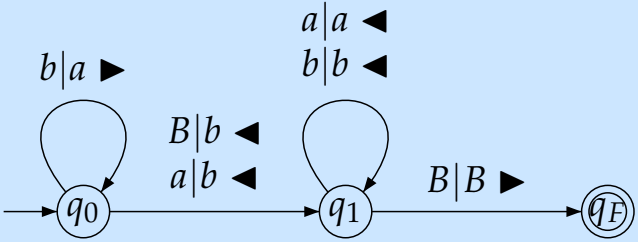
Addition de deux nombres binaires de longueur non bornée

Limites du calcul sans mémoire

Proposition Il n'est pas possible de **multiplier** deux entiers avec le même codage.

Il suffit de considérer la famille de produits $2^n \times 2^n = 2^{2n}$ et d'utiliser un argument de **pompage**.

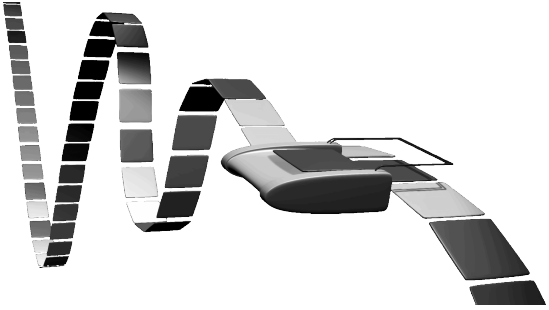
Il faut se laisser le **temps** et l'**espace** pour calculer.



3. Machines à mémoire non bornée

Machines de Turing

La **machine de Turing** classique : un **contrôle fini** couplé à un **ruban** biinfini muni d'une **tête** d'E/S mobile pointant sur une cellule.



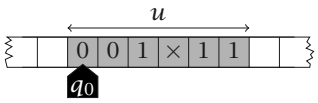
Formellement

Définition Une **MT** est un tuple $(Q, \Gamma, \Sigma, \delta, q_0, B, q_F)$ où

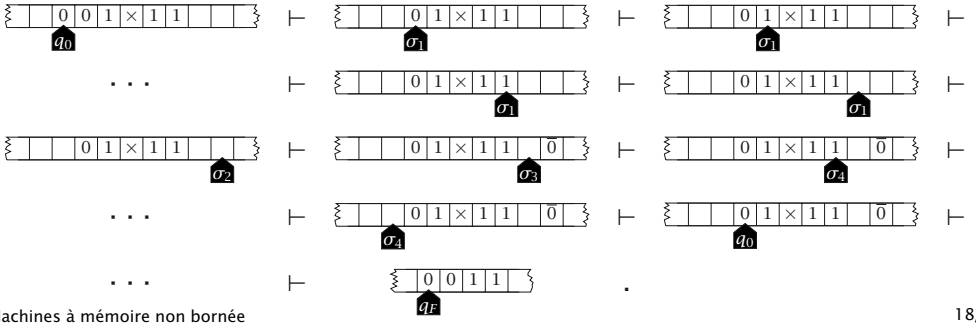
- Q est l'ensemble fini des **états** ;
- Γ est l'**alphabet** fini de **travail** ;
- $\Sigma \subseteq \Gamma$ est l'**alphabet** fini **d'entrée** ;
- $\delta : Q \times \Gamma \rightharpoonup Q \times \Gamma \times \{\blacktriangleleft, \blacktriangledown, \blacktriangleright\}$ est la **fonction de transition**, partielle ;
- $q_0 \in Q$ est l'**état initial** ;
- $B \in \Gamma \setminus \Sigma$ est le **symbole blanc** ;
- $q_F \in Q$ est l'**état d'acceptation** de la machine.

Une transition $\delta(q, a) = (q', b, \Delta)$ signifie :
« Dans l'état q, lorsque je lis le symbole a,
le remplacer par b, entrer dans l'état q' et se déplacer de Δ »

Configurations et étapes de calcul



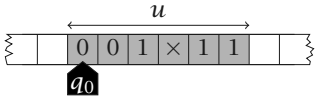
Partant d'une **configuration initiale** où la tête est placée dans l'état q_0 au début du **mot d'entrée** u, la MT calcule en enchaînant les **transitions**.



Configuration initiale

Remarque Même si formellement une configuration est un élément de $Q \times \Gamma^{\mathbb{Z}} \times \mathbb{Z}$, on préfère manipuler des **descriptions instantannées**.

Définition Une **configuration** est un triple $uqv \in \Gamma^* \times Q \times \Gamma^*$. L'espace des configurations est quotienté par l'ajout de symboles B à gauche de u ou à droite de v.



Définition La **configuration initiale** sur l'entrée $u \in \Gamma^*$ est la configuration $c_0(u) = q_0 u$.

Transitions

Définition Un **pas de calcul** $c \vdash c'$ transforme une configuration c en une configuration c' selon les règles suivantes :

$$
\begin{aligned}
ua'qav &\vdash uq'a'bv && \text{si } \delta(q,a) = (q', b, \blacktriangleleft) \\
uqav &\vdash ubq'v && \text{si } \delta(q,a) = (q', b, \blacktriangleright) \\
uqav &\vdash uq'bv && \text{si } \delta(q,a) = (q', b, \blacktriangledown)
\end{aligned}
$$

pour tous $a, a', b \in \Gamma$, $u, v \in \Gamma^*$, $q, q' \in Q$.

On note $\vdash^k$ l'itération, $\vdash^+$ la clôture transitive et $\vdash^*$ la clôture réflexo-transitive de la relation $\vdash$ sur les configurations.

Configuration terminale

Définition Une **configuration terminale** est une configuration pour laquelle aucun pas de calcul n'est possible.

Sur une entrée u, le calcul de la machine partant de $c_0(u)$ peut :

- être **fini** si $c_0(u) \vdash^+ c'$ avec c' une configuration terminale, on dira dans ce cas que **le calcul s'arrête** ;
- être **infini** sinon, on dira alors que **le calcul diverge**.

Lorsque le calcul s'arrête, on observe l'état de la machine dans la configuration terminale :

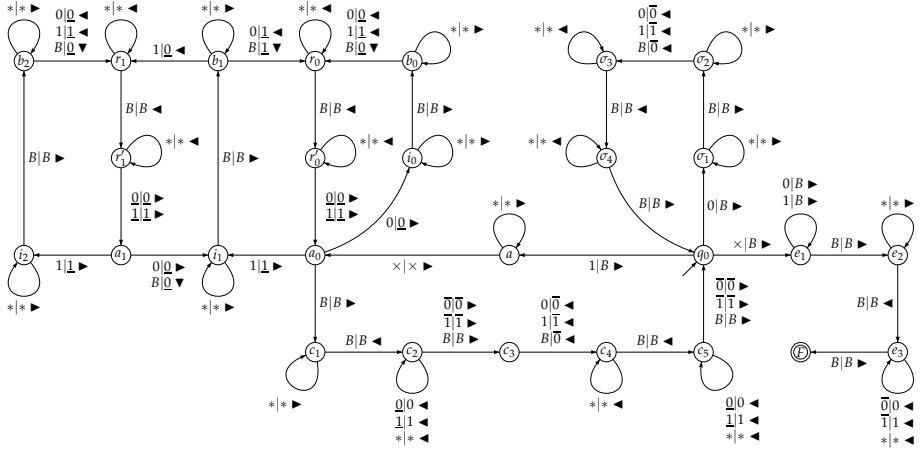
- **la machine accepte** l'entrée u si elle s'arrête dans l'état q_F ;
- **la machine rejette** l'entrée u si elle s'arrête dans un autre état.

Fonctions Turing-calculables

Lorsque la machine **accepte** une entrée, en s'arrêtant dans l'état acceptant q_F, la sortie est lue à droite de la tête, jusqu'au premier symbole blanc exclus.

Définition La **fonction partielle** $f_{\mathcal{M}} : \Sigma^* \rightharpoonup \Gamma^*$ calculée par une MT $\mathcal{M}$ est la fonction qui à une entrée associe sa sortie pour $\mathcal{M}$, lorsqu'elle est définie.

Définition Une **fonction partielle** $f : \Sigma^* \rightharpoonup \Gamma^*$ est **Turing-calculable** si elle est calculée par une machine de Turing.



Multiplication de deux nombres binaires de longueur non bornée codés bit de poids faible en tête

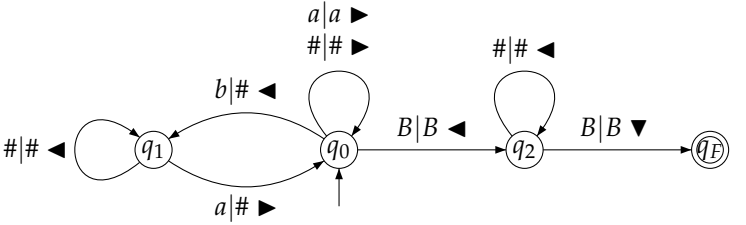
Langages récursifs

Pour raisonner sur le **calculable**, il est souvent pratique de se restreindre aux **prédicats calculables**, c'est-à-dire à une notion de **langage reconnaissable**.

Définition Le **langage $L_{\mathcal{M}}$ associé** à une MT $\mathcal{M}$ est l'ensemble des entrées pour lesquelles $\mathcal{M}$ termine dans l'**état acceptant**.

Définition Un langage est

- **récursivement énumérable** s'il est reconnu par une MT ;
- **co-récursivement énumérable** si son complémentaire l'est ;
- **récursif** s'il est reconnu par une MT **totale**, *i.e.* qui s'arrête sur toute entrée.



MT totale reconnaissant les mots bien parenthésés sur $\{a, b\}$

Normalisation

Lorsqu'on choisit une MT pour reconnaître un langage récursif ou récursivement énumérable, il est toujours possible de choisir une MT **normalisée**.

Définition Une MT **normalisée** est une MT munie d'un état de rejet q_R et telle que :

1. tous les calculs finis terminent dans l'état q_F ou dans l'état q_R ;
2. toutes les transitions de la MT sont définies à l'exception de celles issues de q_F et q_R.

Proposition La famille des **languages récursifs** est close par passage au **complémentaire**.

Programmation concurrente

Étant données deux MT $\mathcal{M}_1$ et $\mathcal{M}_2$, il est possible d'en construire une troisième $\mathcal{M}$ qui **simule** les comportements de $\mathcal{M}_1$ et $\mathcal{M}_2$ en effectuant alternativement des pas de calcul de chacunes des machines sur deux pistes séparées codées sur son ruban.

Proposition Un langage est **récursif** si et seulement s'il est **récursivement énumérable** et **co-récursivement énumérable**.

Proposition La familles des langages **récursivement énumérables** est close par **union** et **intersection**.

Corollaire La familles des langages **récursifs** est close par **union** et **intersection**.

Un modèle robuste

Le modèle des MT présenté semble très **ad hoc**.

Cependant, la classe des fonctions définies est **robuste** à tout un tas de **variations** :

- ruban mono-infini ;
- plusieurs rubans ;
- mémoire comme une file ;
- mémoire comme deux piles ;
- mémoire 2D ;
- mémoire collection de compteurs unaires ;
- . . .