Complexité CM6

Antonio E. Porreca
aeporreca.org

Diapositives basées en partie sur le livre Complexité algorithmique par Sylvain Perifel

Machines de Turing non déterministes (MTND)

Non-déterminisme

- Comme dans les automates finis déterministes
  vs non déterministes

- Au lieu d'avoir une fonction de transition déterministe…

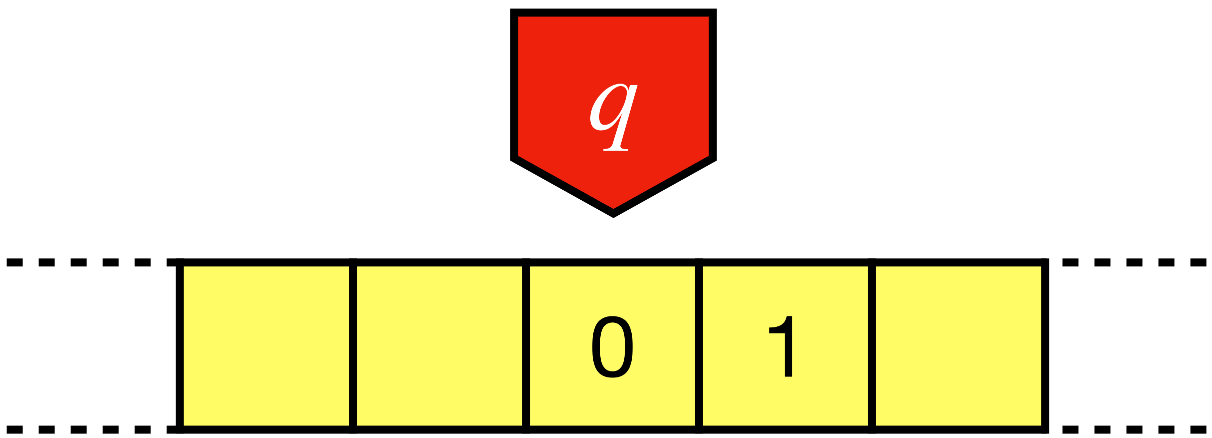
$$\delta\colon (Q \backslash \{q_{\mathrm{oui}}, q_{\mathrm{no}}\}) \times \Gamma \to Q \times \Gamma \times \{-1,0,+1\}$$

- …on admet plusieurs configurations suivantes

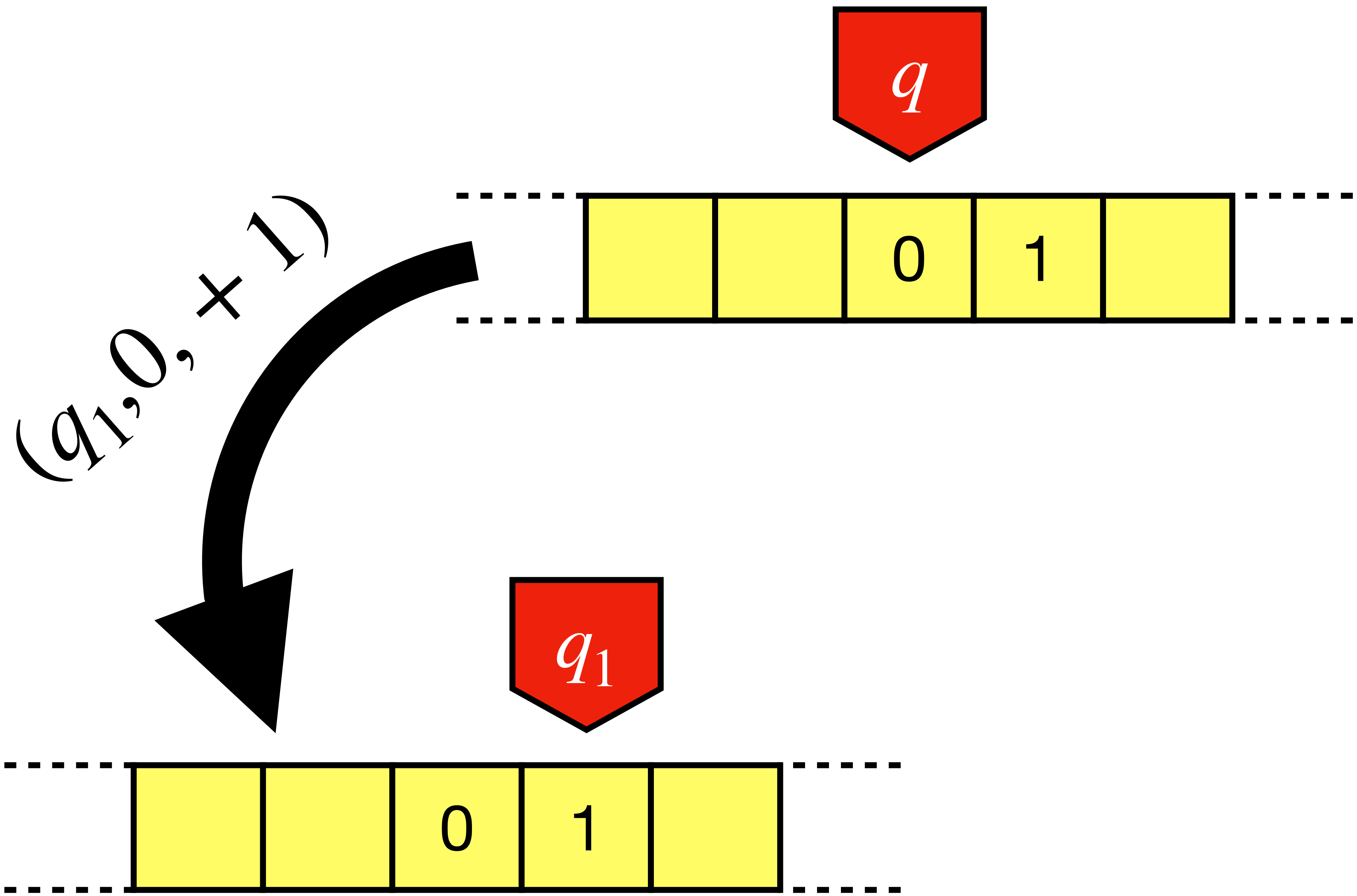
$$\delta\colon (Q \backslash \{q_{\mathrm{oui}}, q_{\mathrm{non}}\}) \times \Gamma \to \mathscr{P}(Q \times \Gamma \times \{-1,0,+1\})$$

- Il y a un maximum de $|Q| \times |\Gamma| \times 3$ transitions possibles,
  qui ne dépend pas de la taille n de l'entrée

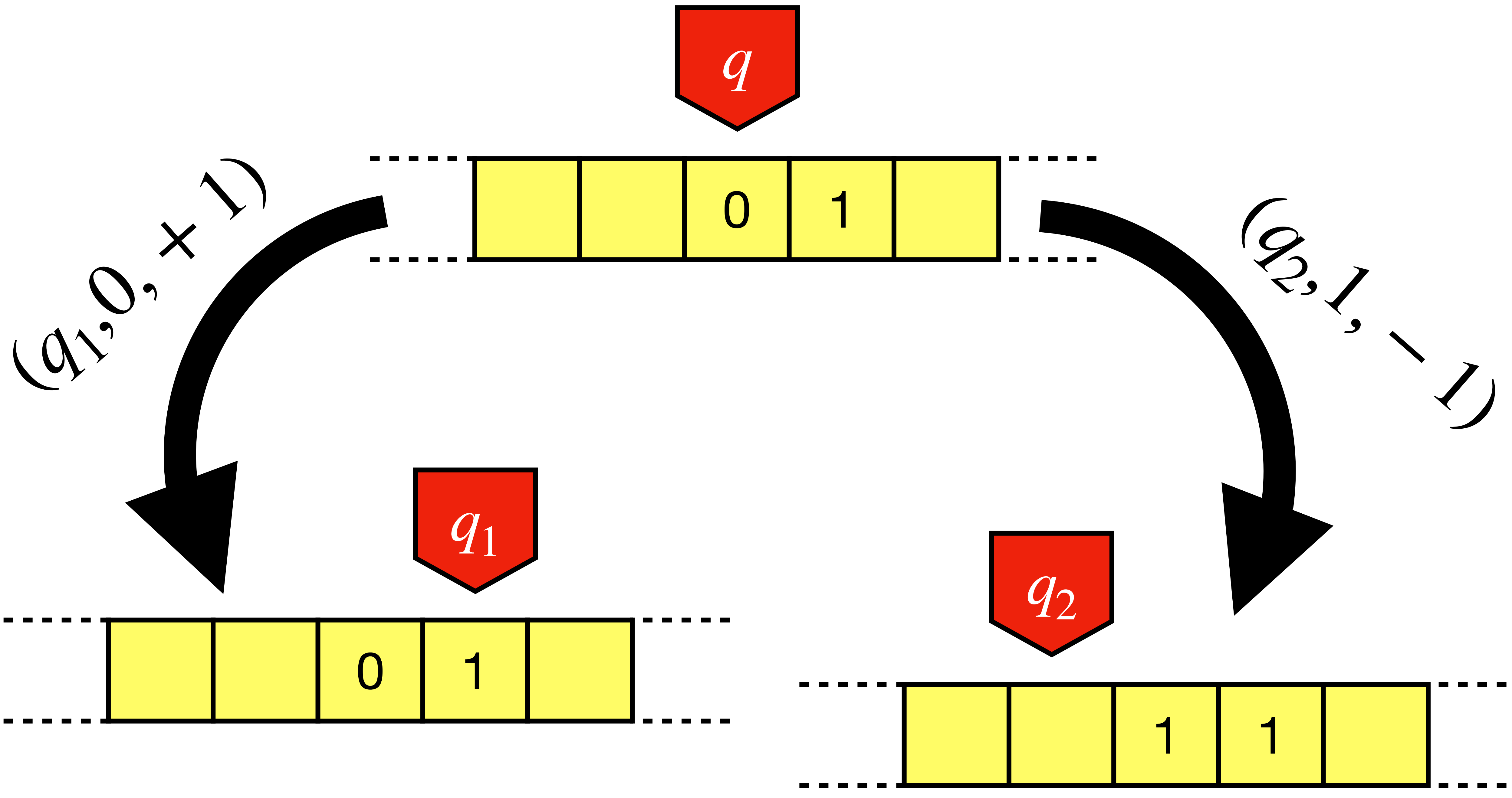
Transition $\delta(q,0) = \left\{ \begin{matrix} (q_1, 0, +1) \\ (q_2, 1, -1) \end{matrix} \right\}$



Transition $\delta(q, 0) = \left\{ \begin{array}{l} (q_1, 0, +1) \\ (q_2, 1, -1) \end{array} \right\}$

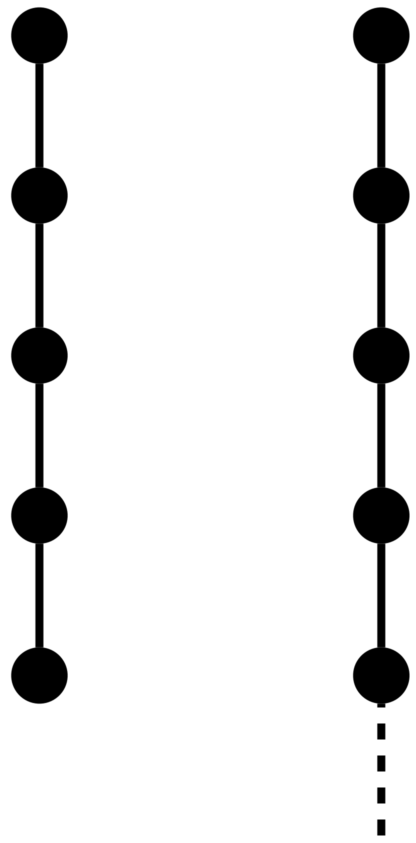


Transition $\delta(q,0) = \left\{ \begin{array}{l} (q_1, 0, +1) \\ (q_2, 1, -1) \end{array} \right\}$

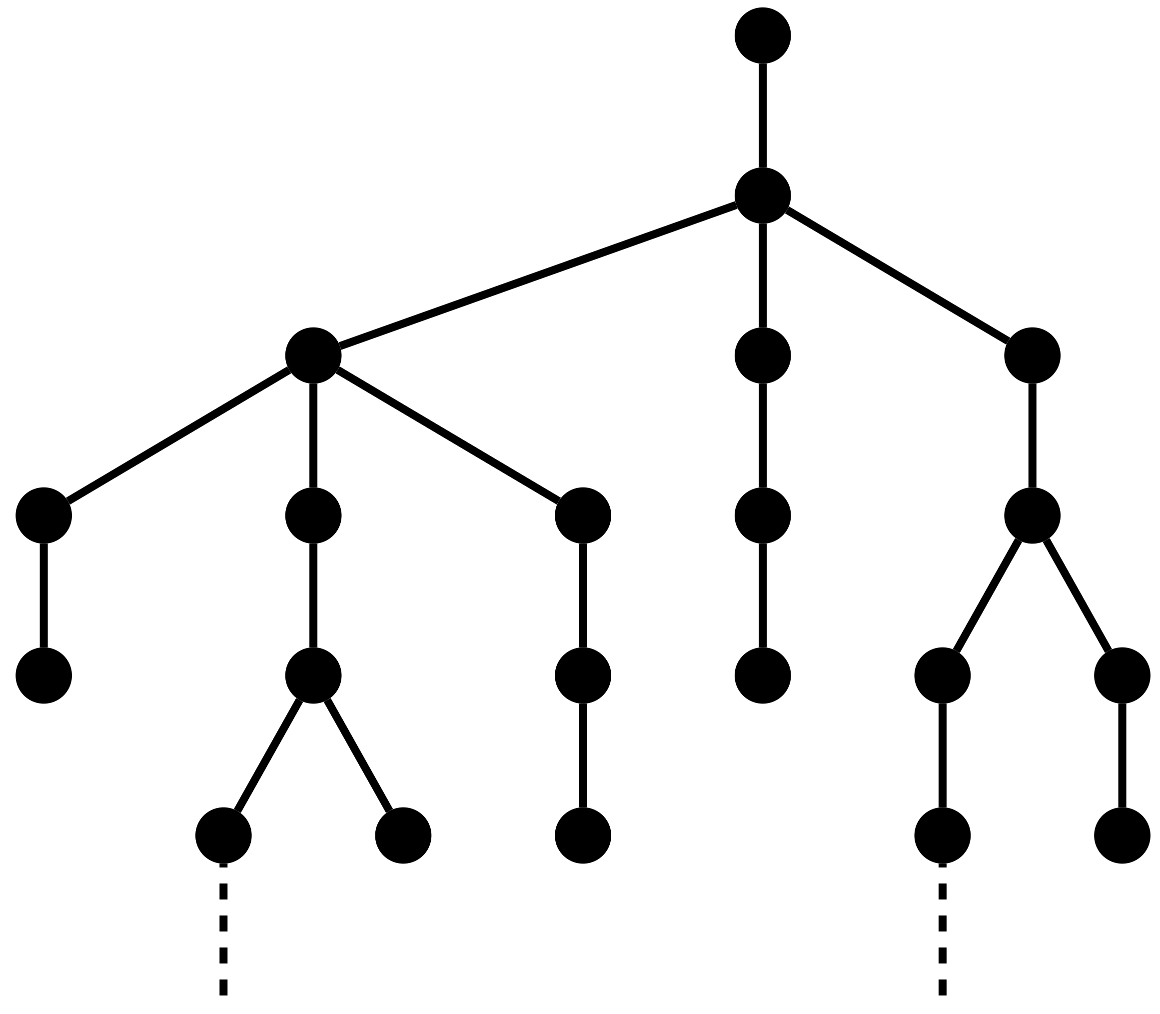


Arbres de calcul

Machines déterministes



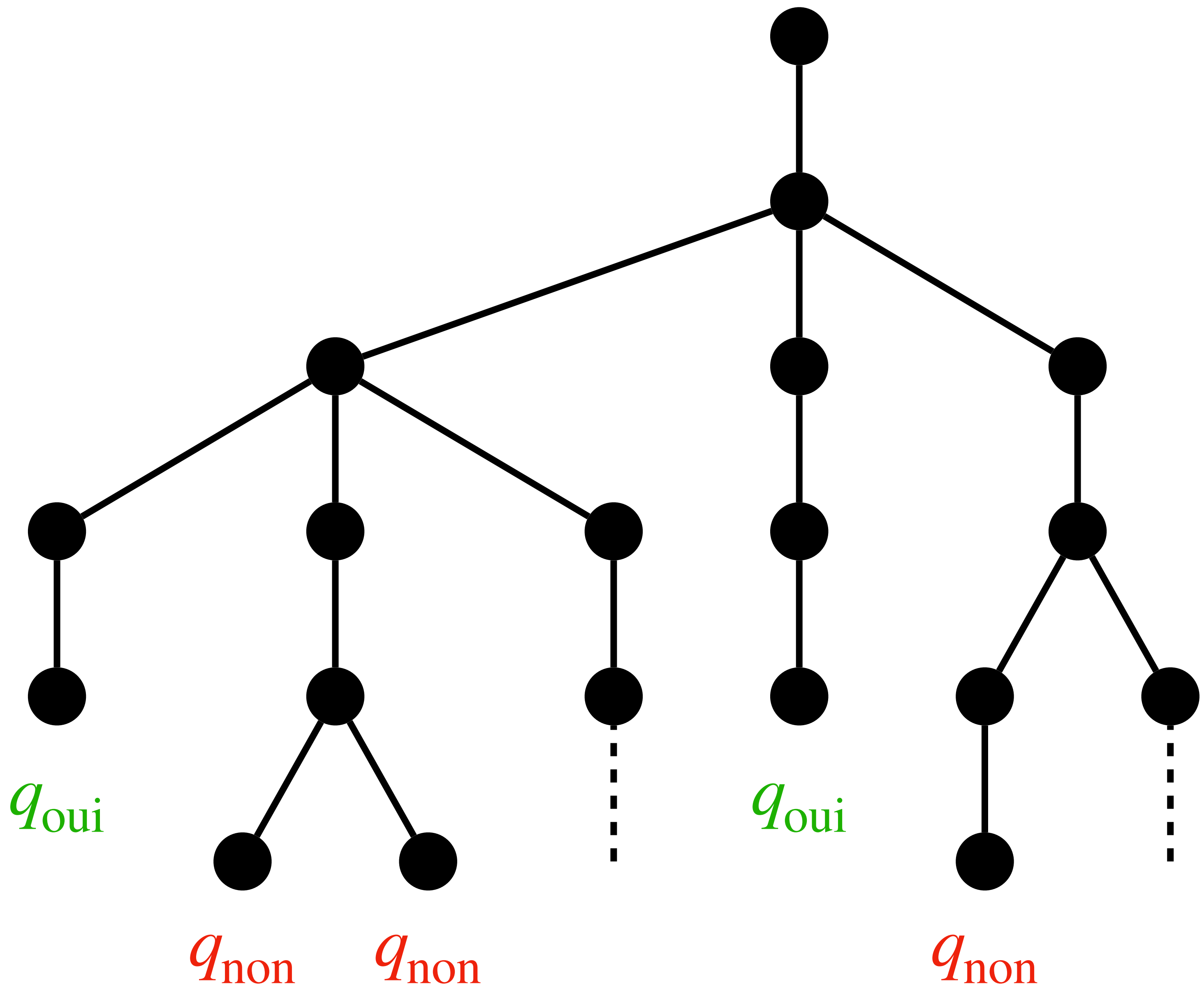
Machine non déterministe



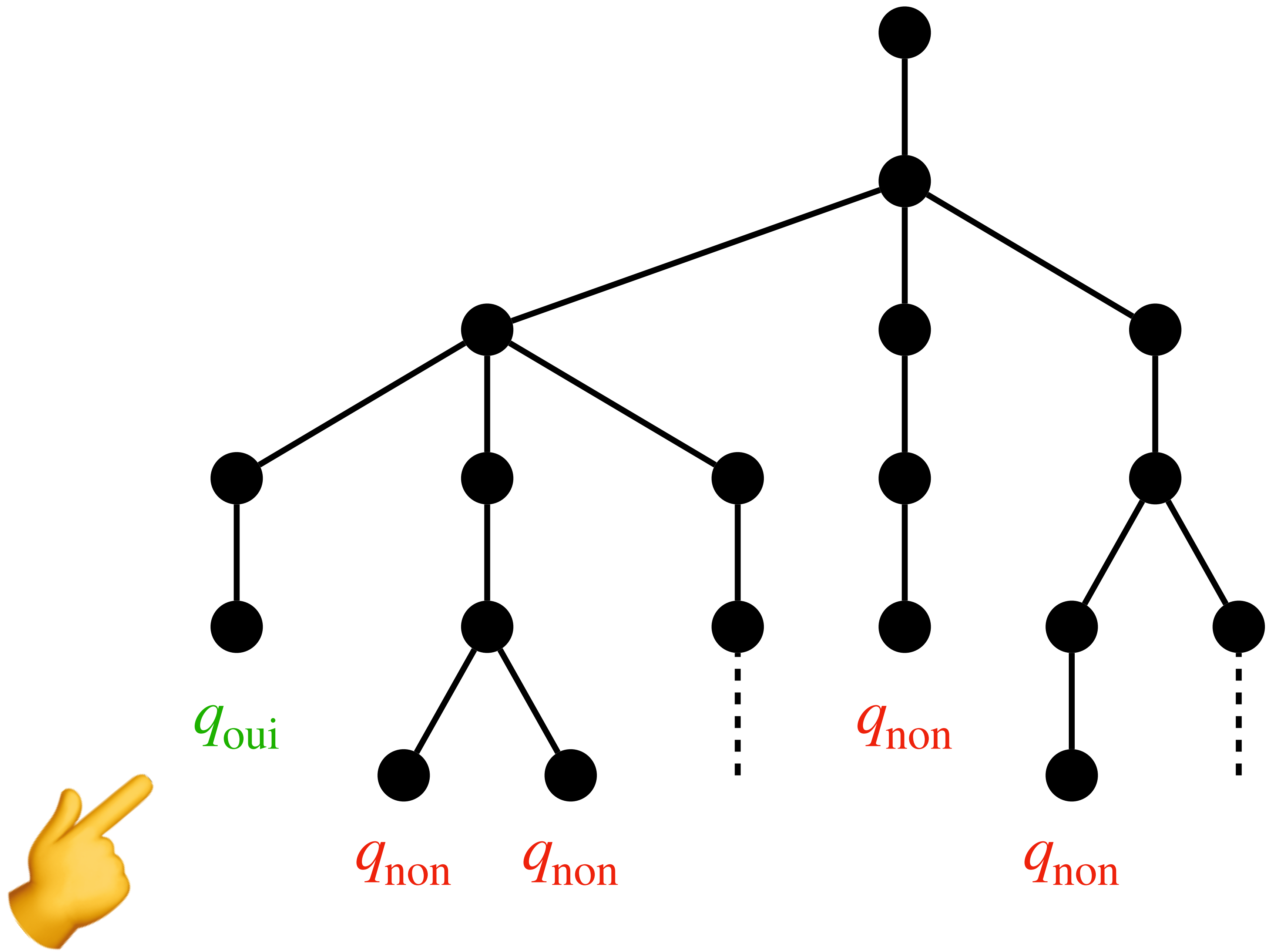
Langage reconnu par une machine de Turing non déterministe N

- $L(N) = \text{Accepte}(N) = \left\{ \begin{array}{c} x \in \Sigma^\star : \text{il existe un calcul de } N \\ \text{sur } x \text{ qui se termine par } q_{\text{oui}} \end{array} \right\}$
- $\text{Rejet}(N) = \left\{ \begin{array}{c} x \in \Sigma^\star : \text{au moins un calcul de } N \text{ sur } x \\ \text{rejette et aucun calcul n'accepte} \end{array} \right\}$
- $\text{Boucle}(N) = \left\{ \begin{array}{c} x \in \Sigma^\star : \text{aucun calcul} \\ \text{de } N \text{ sur } x \text{ ne s'arrête} \end{array} \right\}$

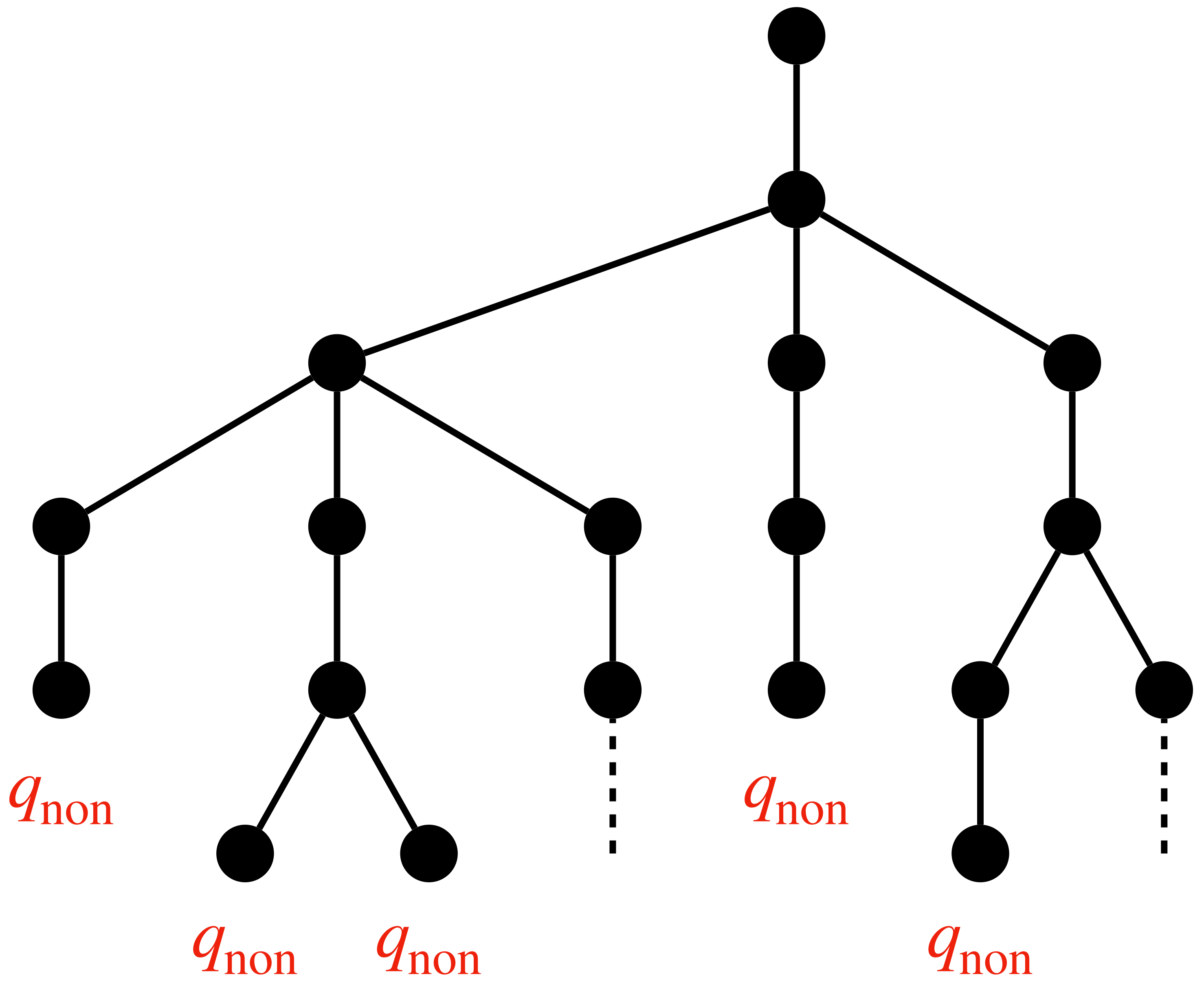
N accepte x



N accepte x



N n'accepte pas x



Résolution de problèmes par une machine de Turing non déterministe

On dit qu'un programme N pour MTND résout un problème de décision π sous un système de codage S si, pour chaque entrée $m \in \Sigma^\star$, tous les calculs de N sur m s'arrêtent et si N reconnait le langage associé au problème, c-à-d si $L(N) = L(\pi, S)$

Puissance des machines de Turing non déterministes

- Les machines non déterministes sont plus générales que les déterministes

- Chaque machine déterministe est un cas particulier de machine non déterministe, avec $|\delta(q, a)| = 1$ pour chaque (q, a)

- Les machines non déterministes ont l'air plus puissant, mais en réalité…

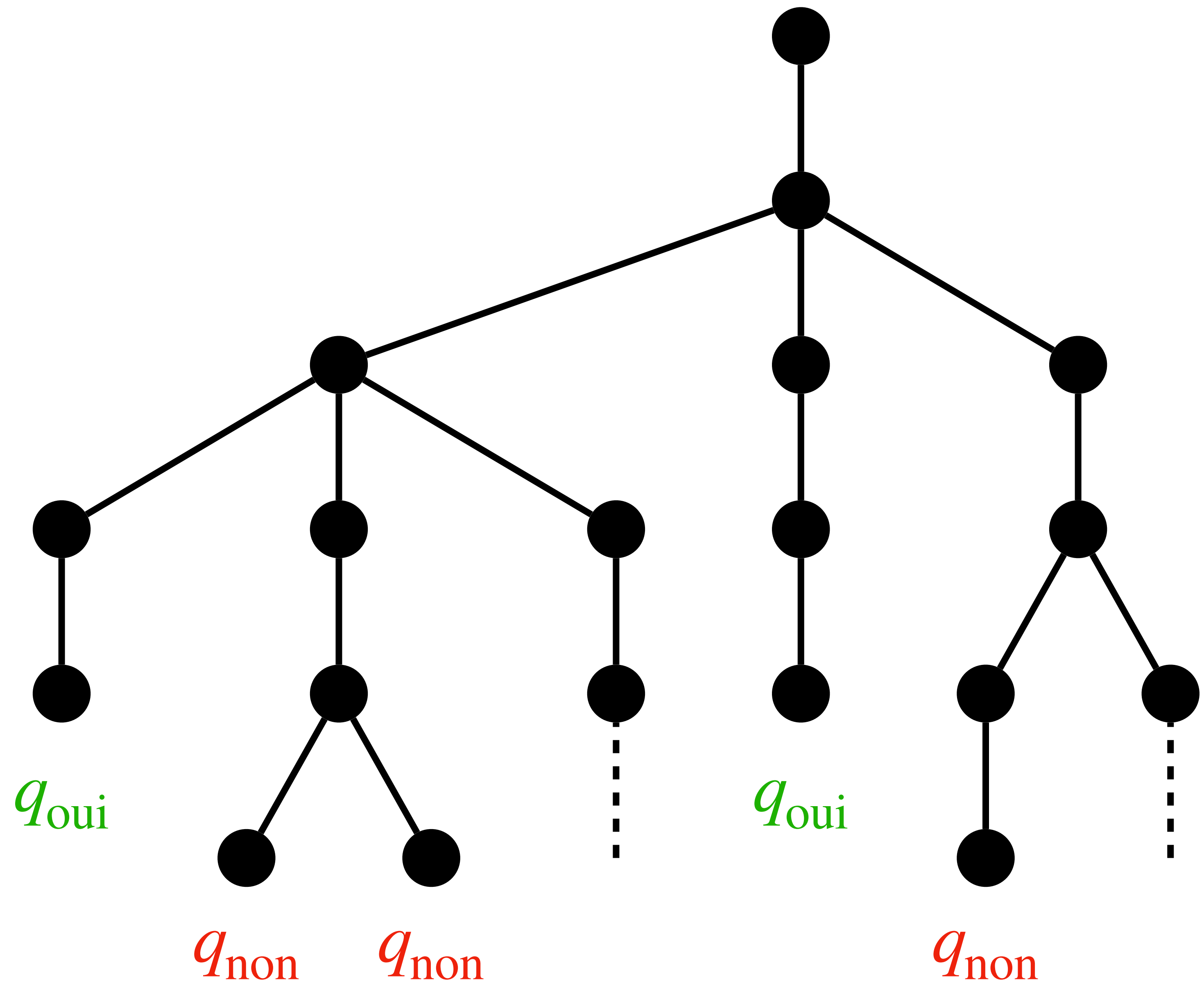
Equivalence des machines déterministes et non déterministes

- Un langage L est reconnu par une machine non déterministe ssi il est reconnu par une machine déterministe

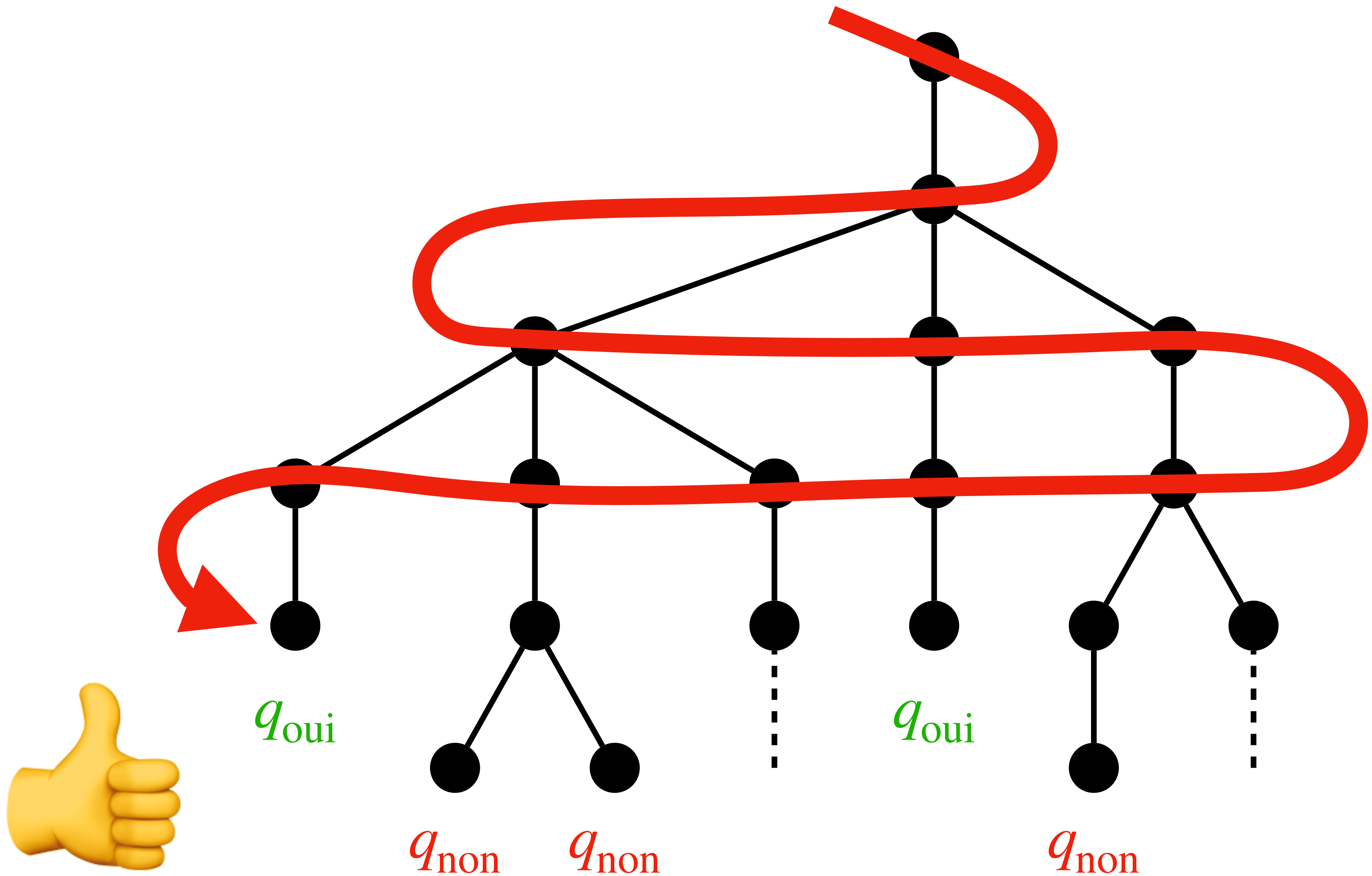
- On a vu que chaque machine déterministe est un type de machine non déterministe

- Vice-versa, on peu simuler de façon déterministe une machine non déterministe en parcourant en largeur son arbre de calcul

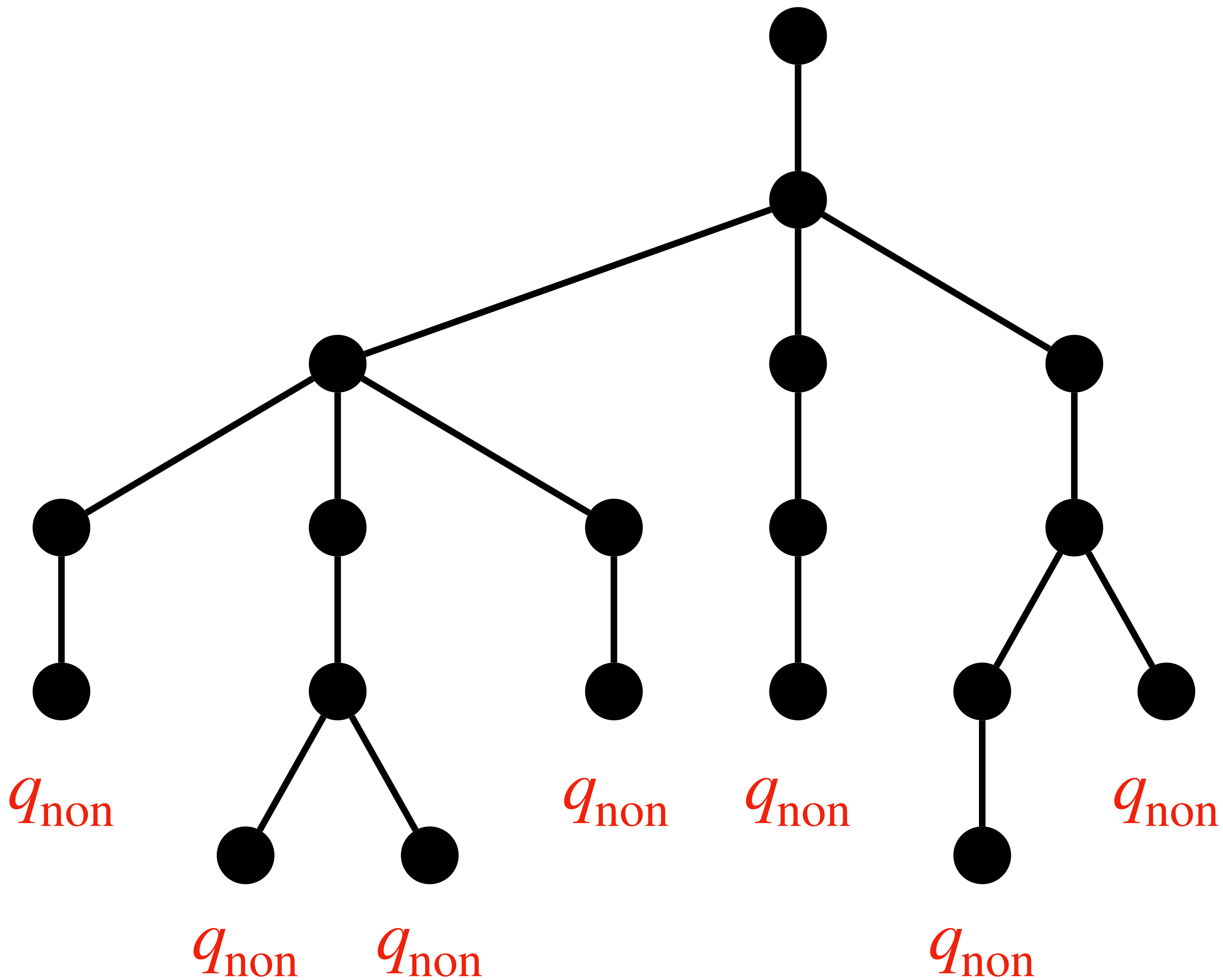
On s'arrête en acceptant si on trouve un calcul qui se termine par q_{oui}



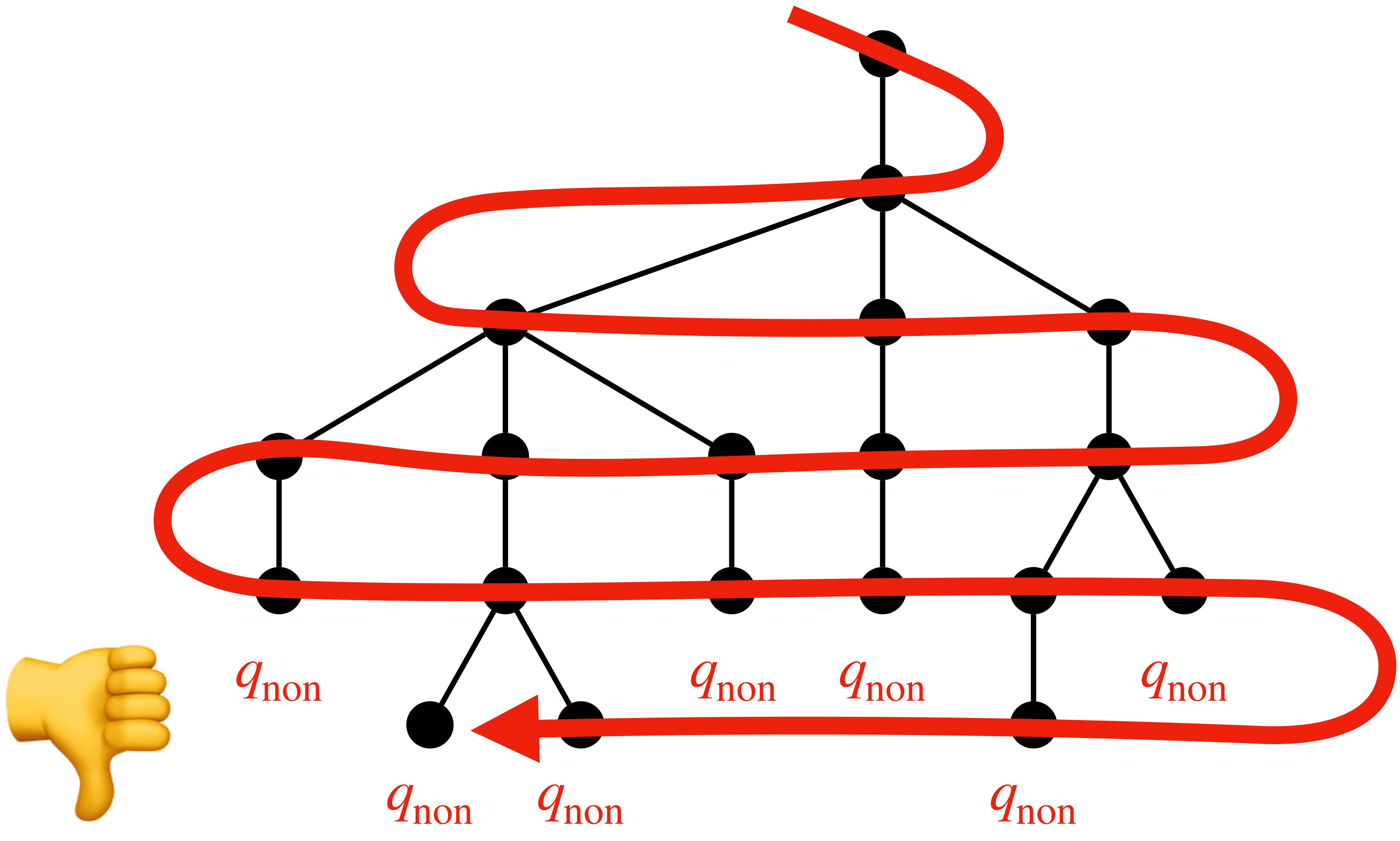
On s'arrête en acceptant si on trouve un calcul qui se termine par q_{oui}



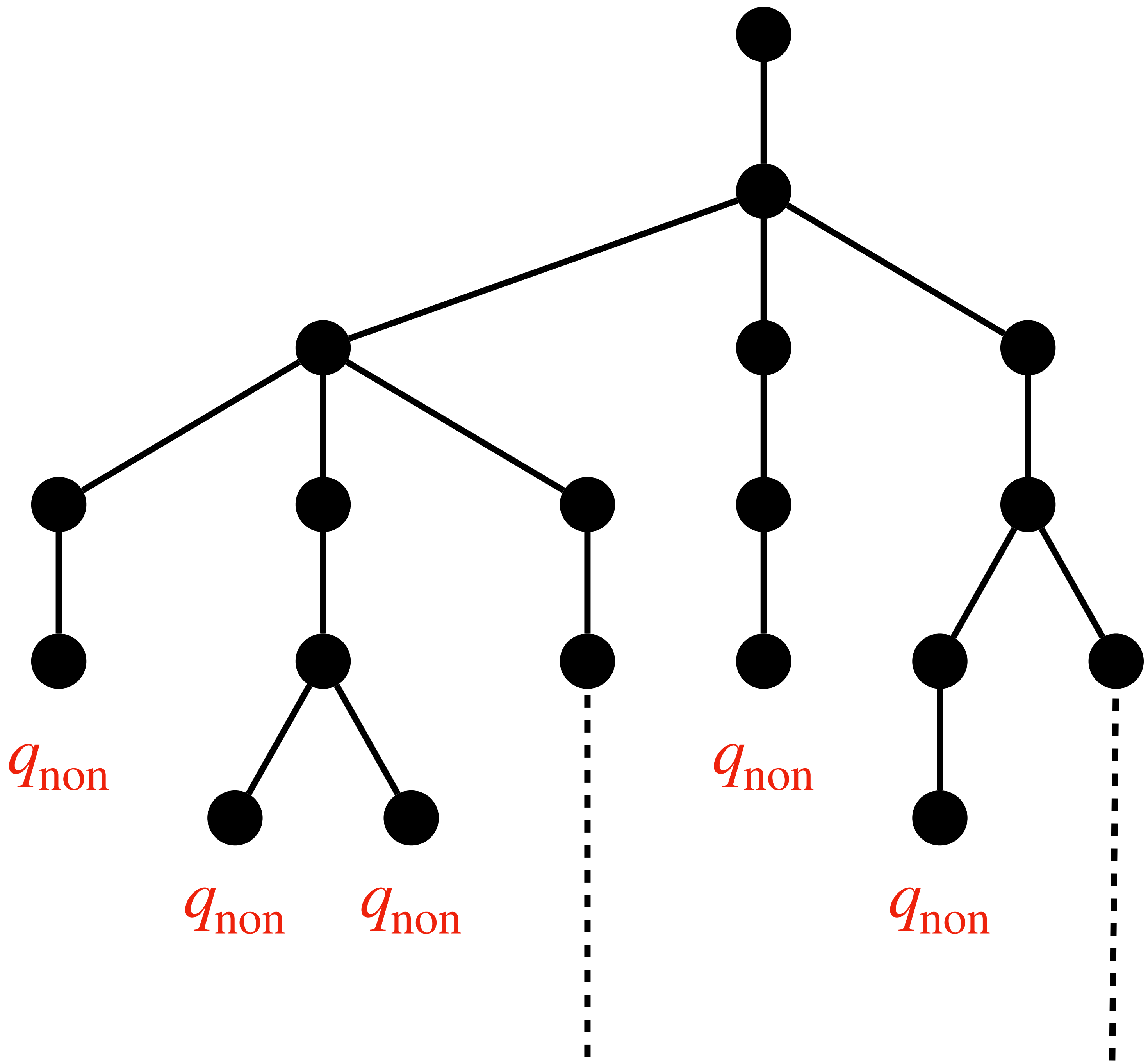
On s’arrête en rejettant si tous les calculs se terminent par q_{no}



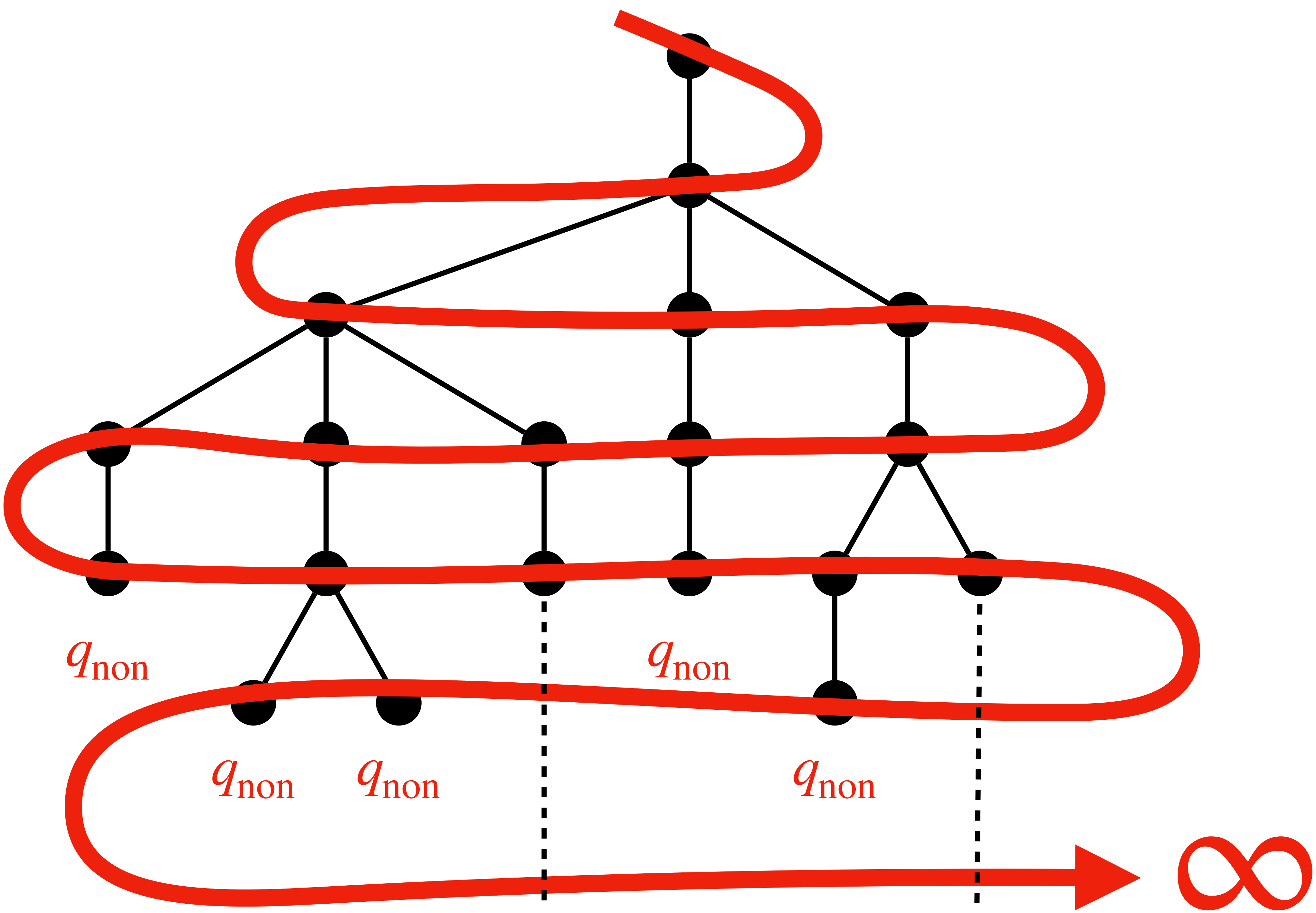
On s’arrête en rejettant si tous les calculs se terminent par q_{no}



Sinon, on ne s'arrête pas



Sinon, on ne s'arrête pas



Algorithmes non déterministes

Pseudo-code non déterministe

- On exprime un choix non déterministe en pseudo-code avec une fonction « devine », qui choisit parmi un ensemble fini de valeurs de taille constante :

$$x := \text{devine}(\text{choix}_1, \text{choix}_2, \ldots, \text{choix}_k)$$

- On peut toujours se reconduire à une divination binaire, si besoin est :

$$x := \text{devine}(0,1)$$

Divination d'un element x d'un ensemble X de taille non constante

$m := |X|$

$i := 0$

tant que $i \leq m - 2$ **et** devine$(0,1)$ $= 0$ **faire**

    $i := i + 1$

$x := X[i]$

Divination d'un element x d'un ensemble X de taille non constante

$m := |X|$
$i := 0$
tant que $i \leq m - 2$ **et** devine$(0,1) = 0$ **faire**
    $i := i + 1$
$x := X[i]$

$x :=$ devine(X)

Un exemple : clique

```
fonction clique(S, A, k)
    C := tableau(k)
    pour i := 0 à k − 1 faire
        C[i] := devine(S)
    pour chaque s ∈ S faire
        si C contient s plusieurs fois alors
            rejeter
    pour i := 0 à k − 1 faire
        pour j := i + 1 à k − 1 faire
            si {C[i], C[j]} ∉ A alors
                rejeter
    accepter
fin
```

Un exemple : clique

fonction clique(S, A, k)
    C := tableau(k)
    pour $i := 0$ **à** $k - 1$ **faire**
        $C[i]$:= devine(S)
    pour chaque $s \in S$ **faire**
        si C contient s plusieurs fois **alors**
            rejeter
    pour $i := 0$ **à** $k - 1$ **faire**
        pour $j := i + 1$ **à** $k - 1$ **faire**
            si $\{C[i], C[j]\} \notin A$ **alors**
                rejeter
    accepter
fin

C est-il un ensemble de k éléments différents ?

Un exemple : clique

fonction clique(S, A, k)
    C := tableau(k)
    pour $i := 0$ **à** $k - 1$ **faire**
        $C[i]$:= devine(S)
    pour chaque $s \in S$ **faire**
        si C contient s plusieurs fois **alors**
            rejeter
    pour $i := 0$ **à** $k - 1$ **faire**
        pour $j := i + 1$ **à** $k - 1$ **faire**
            si $\{C[i], C[j]\} \notin A$ **alors**
                rejeter
    accepter
fin

C est-il un ensemble de k éléments différents ?

les sommets dans C sont-ils tous reliés par des arêtes ?

Un exemple : clique

```
fonction clique(S, A, k)
    C := tableau(k)
    pour i := 0 à k − 1 faire
        C[i] := devine(S)
    pour chaque s ∈ S faire
        si C contient s plusieurs fois alors
            rejeter
    pour i := 0 à k − 1 faire
        pour j := i + 1 à k − 1 faire
            si {C[i], C[j]} ∉ A alors
                rejeter
    accepter
fin
```

$O(k \log |S|)$ **bits devinés**

C **est-il un ensemble de** k **éléments différents ?**

les sommets dans C **sont-ils tous reliés par des arêtes ?**

Simulation du non déterminisme dans le monde réel*

```python
from nondeterminism import *

@nondeterministic
def clique(vertices, edges, size):
    n = len(vertices)
    c = []
    for i in range(size):
        v = guess(vertices)
        c.append(v)
    for v in vertices:
        if c.count(v) > 1:
            reject()
    for i in range(size):
        for j in range(i+1, size):
            if {c[i], c[j]} not in edges:
                reject()
    accept()
```

* github.com/aeporreca/nondeterminism

Simulation du non déterminisme dans le monde réel*

```python
from nondeterminism import *

@nondeterministic
def clique(vertices, edges, size):
    n = len(vertices)
    c = []
    for i in ra
        v = gue
        c.appen
    for v in ve
        if c.co
            rej
    for i in ra
        for j i
            if

    accept()
```

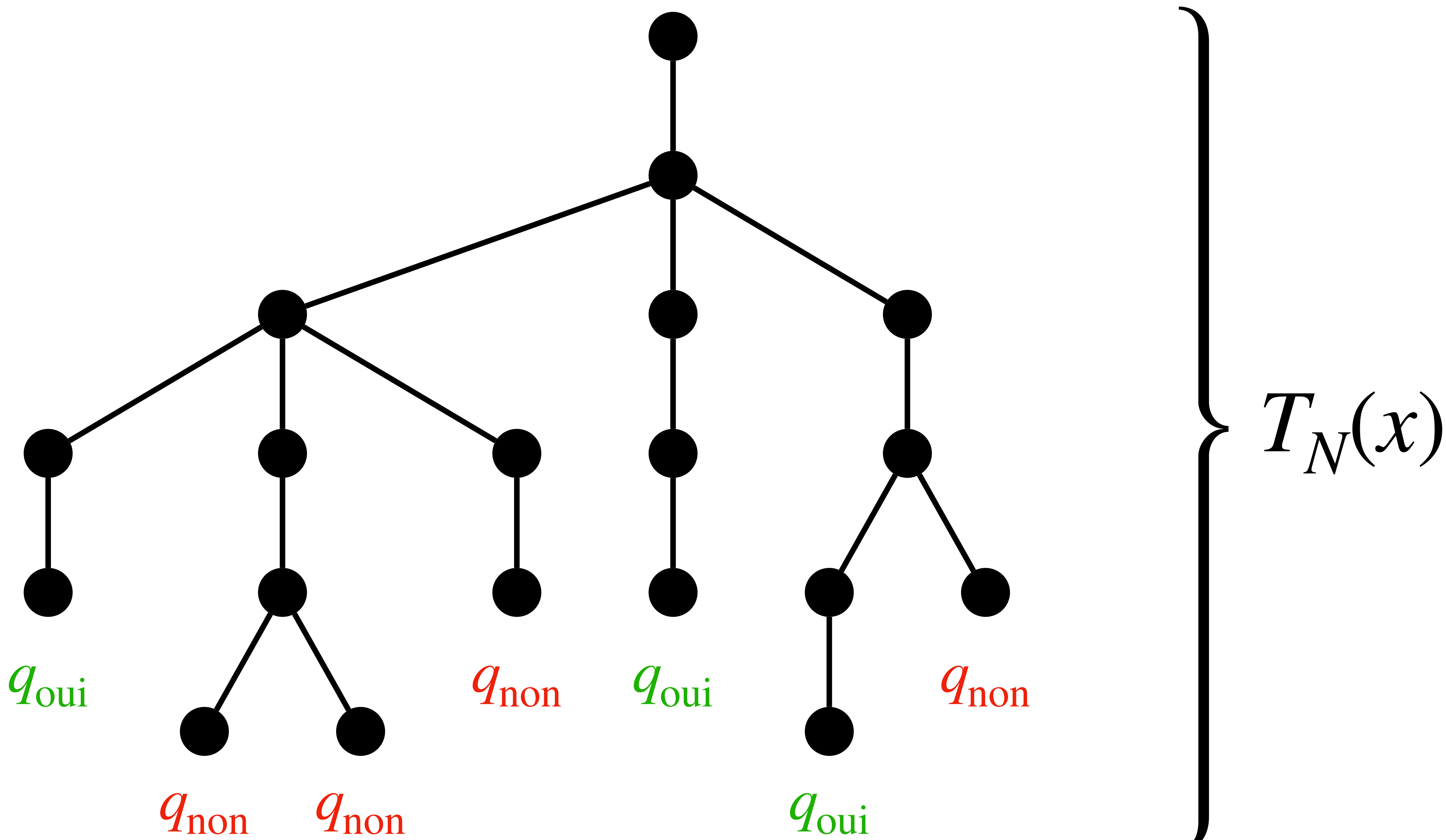
nondeterminism — 38×12 — ⌥⌘1

```
>>> vertices = [0,1,2,3,4,5]
>>> edges = [{0, 1}, {1, 3}, {3, 0}]
>>> clique(vertices, edges, 3)
True
>>> clique(vertices, edges, 4)
False
>>>
```

* github.com/aeporreca/nondeterminism

Mesure du temps de calcul non déterministe

Temps de calcul sur l'entrée x = hauteur de l'arbre de calcul



Temps de calcul d'une machine non déterministe N

- Comme dans le cas déterministe, on prend le max des temps de calcul des entrées de taille n :

$$T_N(n) = \max\{T_N(x) : x \in \Sigma^\star \text{ et } |x| = n\}$$

- Donc la longueur du chemin de calcul le plus long des arbres de calcul des entrées de taille n

- Le temps de calcul est polynomial si $T_N(n) \in O(p(n))$ pour un polynôme p

La classe de complexité $\mathbf{NP}$

- C'est la classe de langages reconnus par des machines de Turing non déterministes en temps polynomial

$$\mathbf{NP} = \left\{ \begin{array}{c} L : \text{il existe une machine de Turing} \\ \text{non déterministe } N \text{ qui fonctionne en temps} \\ \text{polynomial telle que } L = L(N) \end{array} \right\}$$

- De façon équivalente, c'est aussi la classe de problèmes π sous le codage S tels que $L(\pi, S) \in \mathbf{NP}$

$\mathbf{P}$ vs $\mathbf{NP}$

- Comme on peut voir chaque machine déterministe comme machine non déterministe, on a automatiquement $\mathbf{P} \subseteq \mathbf{NP}$

- On sait qu'on peut simuler de façon déterministe chaque machine non déterministe, mais on ne sais pas si on peut le faire efficacement (en temps polynomial)

- Donc on ne sais pas si $\mathbf{NP} \subseteq \mathbf{P}$ et donc si $\mathbf{P} = \mathbf{NP}$

- C'est l'un des Millennium Prize Problems du Clay Mathematics Institute, et il y a un prix de 1000000 $ pour celui qui trouve la réponse !

La classe de complexité $\mathbf{coNP}$

- C'est la classe de langages dont le complément appartient à $\mathbf{NP}$:

$$\mathbf{coNP} = \{L : \text{co-}L \in \mathbf{NP}\}$$

- ⚠️ $\mathbf{coNP}$ n'est pas le complement de $\mathbf{NP}$, c'est-à-dire, ce n'est pas la classe des problèmes qu'on ne peut pas résoudre en temps polynomial de façon non déterministe

- Notamment, on a $\mathbf{P} = \mathbf{coP}$, donc $\mathbf{P} \subseteq \mathbf{coNP}$, ce qui implique $\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}$: $\mathbf{NP}$ et $\mathbf{coNP}$ ne sont pas disjointes

- On ne sait pas si $\mathbf{NP} = \mathbf{coNP}$ non plus !

$\mathbf{NP} \neq \mathbf{coNP}$ vaut 1000000 $

- Si $\mathbf{NP} \neq \mathbf{coNP}$ alors $\mathbf{P} \neq \mathbf{NP}$!

- On a vu que $\mathbf{P} = \mathbf{coP}$, donc $\mathbf{P} = \mathbf{NP}$ impliquerait $\mathbf{coNP} = \mathbf{coP} = \mathbf{P}$ et donc $\mathbf{NP} = \mathbf{coNP}$

- La proposition contraposée de $\mathbf{P} = \mathbf{NP} \Rightarrow \mathbf{NP} = \mathbf{coNP}$ est $\mathbf{NP} \neq \mathbf{coNP} \Rightarrow \mathbf{P} \neq \mathbf{NP}$

- $\mathbf{NP}$ vs $\mathbf{coNP}$ ne semble pas du tout plus simple à résoudre que $\mathbf{P}$ vs $\mathbf{NP}$…