Méthodes formelles pour XML : automates d'arbres

Pierre-Alain Reynier
http://www.lif.univ-mrs.fr/~preynier/XML/

Plan du cours

1 - Motivations

2 - Automates

2.1 sur mots
2.2 sur arbres d'arité fixe
2.3 sur arbres d'arité variable

1 - Motivations

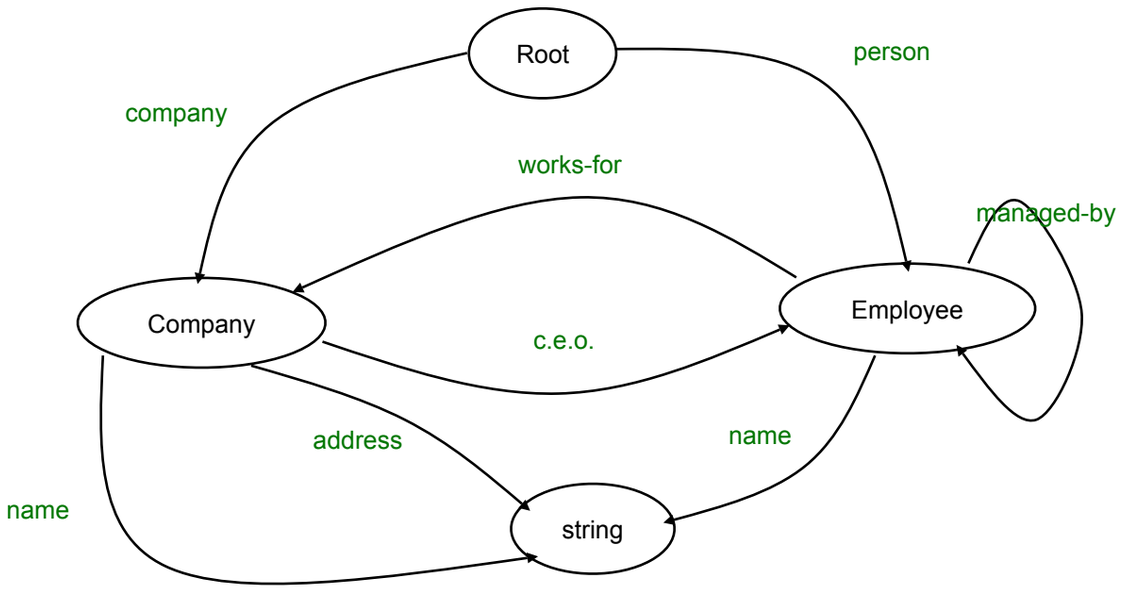
Objectifs

- Les manipulations de documents XML doivent être sûres, et efficaces
- Besoin de méthodes permettant d'étudier
  - les types des documents XML produits,
  - les requêtes XPath,
  - les transformations XSLT, XQuery...
- Comprendre la puissance expressive et la complexité de ces opérations
- Identifier des fragments présentant un compromis intéressant

XML Typing

- Type : DTD, Schéma XML, Relax NG...
- Simplifie l'écriture de programmes XML (améliore l'interopérabilité entre programmes)
- Améliore le stockage et les performances
- Facilite les requêtes
- Simplifie la protection des données (rejeter les mises à jour non conformes)

Amélioration du stockage



Lower-bound schema

Company

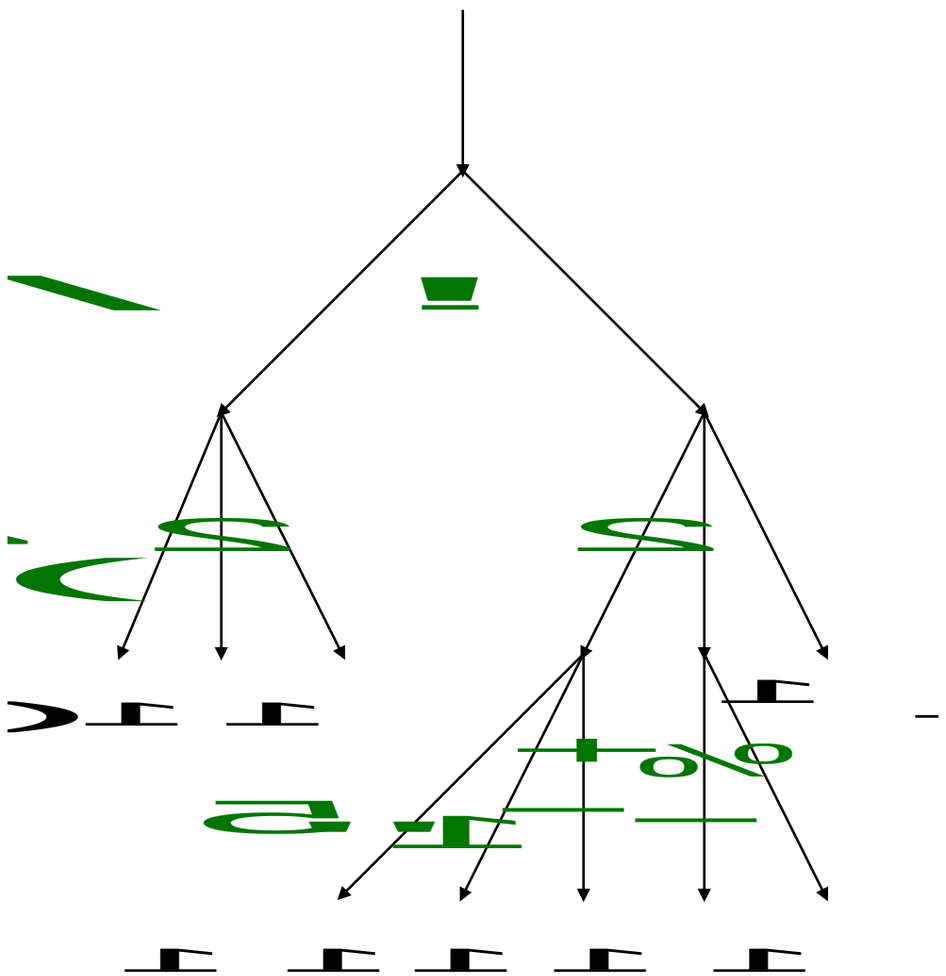
oid	name	address	c.e.o.
…	…	…	…
…	…	…	…

Employee

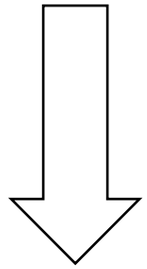
oid	name	managed-by	works-for
…	…	…	…
…	…	…	…

Store rest in overflow graph

Amélioration des performances



```
select X.title
from   Bib._ X
where X.*.zip = “12345”
```



```
select X.title
from   Bib.book X
where X.address.zip = “12345”
```

Manipulations de base

- Validation
  vérifier qu'un document XML est conforme à un type donné (DTD, Schéma XML...)
- Navigation/Extraction
  Sélectionner un ensemble de noeuds dans un document XML (XPath)
- Transformation
  Construire à partir d'un doc. XML un nouveau document XML (XSLT, XQuery..)

Analyse dynamique/statique

- Contexte :
  - Un serveur reçoit un ou plusieurs docs XML respectant des contraintes de types
  - Il applique une transformation à ces données et produit un nouveau doc XML
  - Le document produit est renvoyé à un autre service et doit respecter un nouveau type
- Deux approches :
  - Vérification dynamique : après que les données soit produites
  - Vérification statique : vérification du programme produisant les données

Problèmes d'analyse statique

- Satisfaisabilité :
  Etant donné une requête q, existe-t-il un doc. XML t tel que q(t) est non vide ?

- Inclusion :
  Etant donné deux requêtes q1 et q2, est-ce que pour tout doc XML t, on a q1(t) ⊂ q2(t) ?

- Equivalence :
  Etant donné deux requêtes q1 et q2, est-ce que pour tout doc XML t, on a q1(t)=q2(t) ?

Problèmes d'analyse statique

- Inclusion de type :
  Etant donné deux types d1, d2, est-ce que tout doc. XML de type d1 est de type d2 ?

- Vérification de type :
  Etant donné deux types d1, d2, et une transformation T, est-ce que pour tout doc. XML t de type d1, T(t) est de type d2 ?

- Inférence de type :
  Etant donné un type d et une transformation T, calculer le type de {T(t) | t de type d}

Vérification de types

- Vérifié par :
  • Editeurs XML : vérifie que les données sont conformes à leur type
  • Echangeurs de XML (Services Web)
    - serveurs en envoyant les données
    - clients en les recevant
- Mieux : de façon statique
  assurer que l'application produit un document du bon type

Vérification statique

- Entrée : type T et code d'une fonction f
  (f appartient à XQuery, XPATH, XSLT...)
- Vérification de T' :
  Est-ce que $\forall d \models T, f(d) \models T'$?
- Inférence de types :
  Calculer le plus petit T' tel que $\forall d \models T, f(d) \models T'$
- Rapidement indécidable à cause des jointures (comparaison d'éléments)

Exemple

```
for $p in doc("parts.xml“)//part[color=“red"]
return <part>
                <name>$p/name</name>
                <desc>$p/desc</desc>
        </part>
```

Type résultant :

(part (name (string) desc (any))*

Si le type de parts.xml//part/desc est string :

(part (name (string) desc (string))*

Difficulté

for $X in Input, $Y in Input do { print (<b/> }

Input: <a/> <a/> <a/>

Result: <b/> <b/> <b/> <b/> <b/> <b/> <b/> <b/> <b/>

Problème: { $b^i \mid i=n^2$ for $n \geq 0$ } ne peut pas être décrit en XML Schema

Il n'y a pas de "meilleure approximation" :

- b^*
- $\varepsilon + b^2 b^*$
- $\varepsilon + b^2 + b^4 b^*$
- $\varepsilon + b^2 + b^4 + b^9 b^*$
- …

Modèle de données

- Un choix naturel : les arbres
  - reflètent la structure des doc XML
  - modèle de données sous-jacent à XML basé sur les arbres
- Limitations :
  - ne modélisent pas ID, IDREF
  - parfois des extensions sont nécessaires
- Dans la suite, nous considérerons cette vision arborescente des doc XML

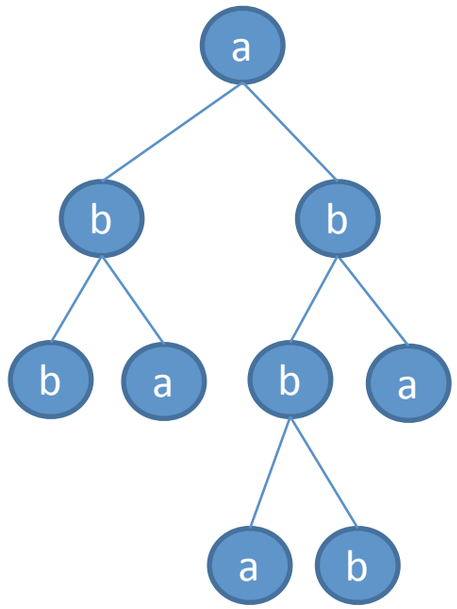
Aperçu sur les aut. d'arbres

- Théorie riche pour les mots : Automates finis
- Etendue en une théorie riche pour les arbres d'arité fixe : Automates d'arbres
  - Théorèmes fondamentaux bien connus
  - Algorithmes et complexité des problèmes de décision connus
- Extension aux arbres d'arité variable ?

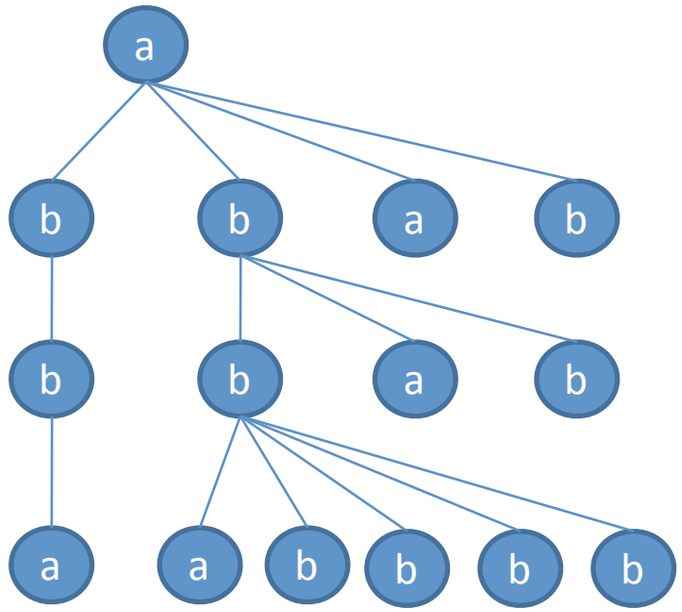
  Oui!

Des mots aux arbres

Mots : Automates finis



Arbres binaires : Automates d'arbres d'arité 2



Arbres quelconques : Automates d'arbres d'arité variable

II - Automates

- sur mots -

Automates finis sur mots

$$(\Sigma, Q, q_0, F, \delta)$$

Alphabet

Etats

Etat initial

$q_0 \in Q$

Etats finaux/acceptants

$F \subseteq Q$

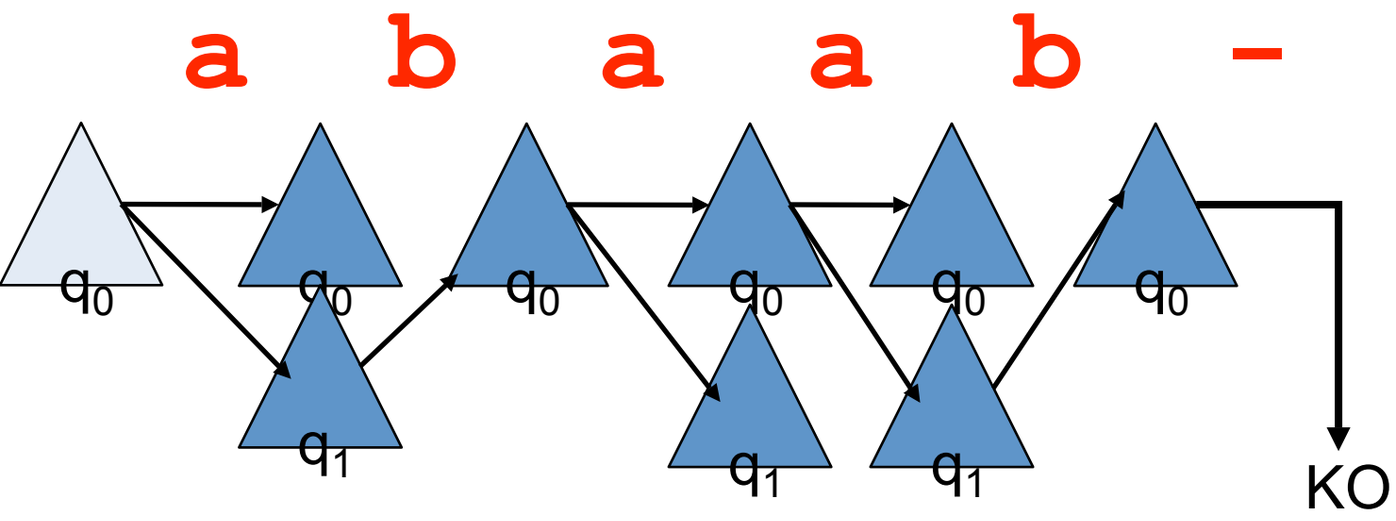
Transitions

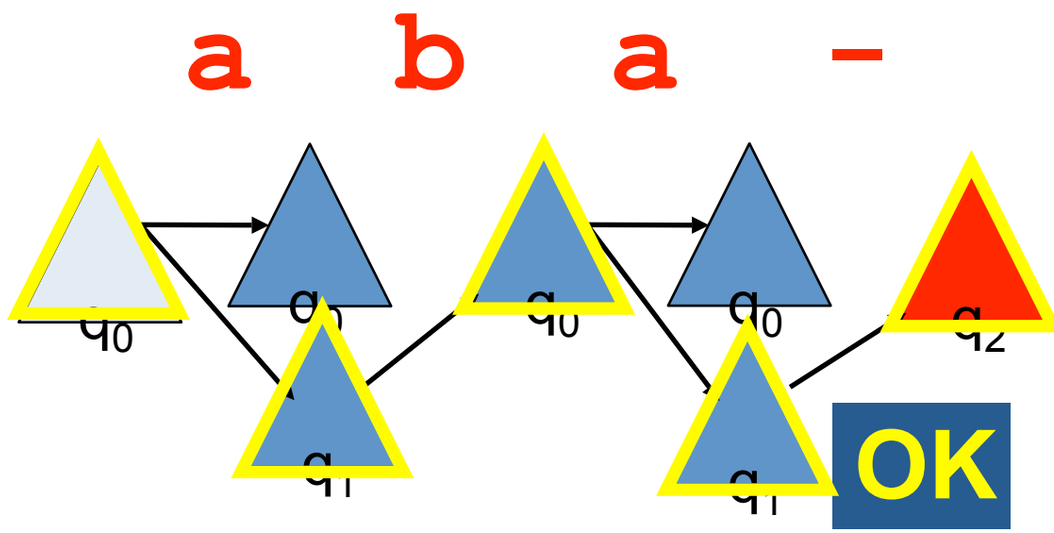
$\delta : \Sigma \times Q \rightarrow P(Q)$

Automates non déterministes

$$\Sigma = \{a, b\}$$
$$Q = \{q_0, q_1, q_2, q_3\}$$
$$F = \{q_2\}$$

$$\delta(a, q_0) = \{q_0, q_1\}$$
$$\delta(b, q_1) = \{q_0\}$$
$$\delta(-, q_1) = \{q_2\}$$
$$\delta(-, q_2) = \{q_3\}$$
$$\delta(-, q_3) = \{q_3\}$$





Rappels

- Déterministe :
  - pas de transition ε
  - pas de définition à choix comme

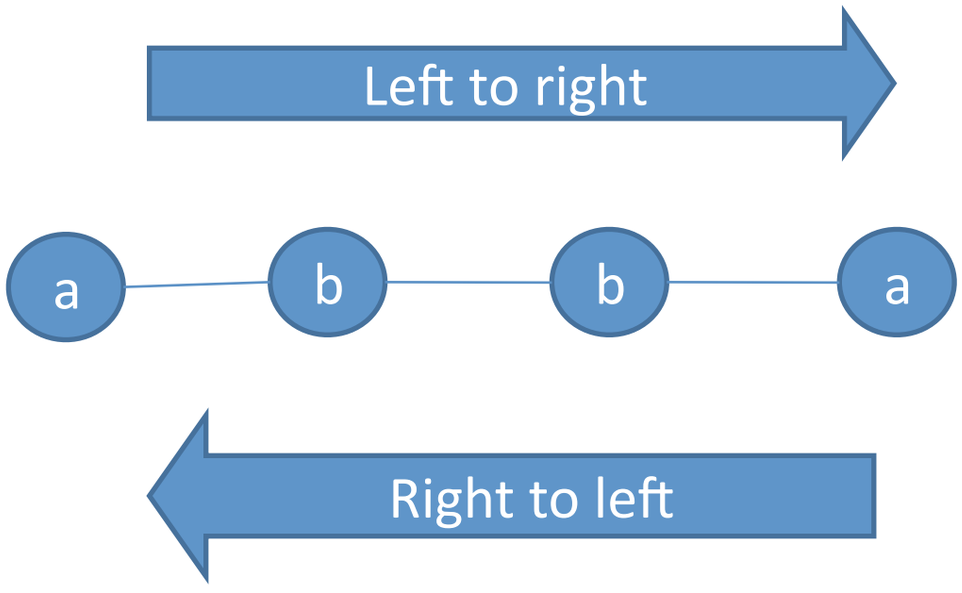
$$\delta(-, q) = \{q_0\}$$
$$\delta(a, q_0) = \{q_0, q_1\}$$

- Déterminisation :
  - possible de construire un équivalent déterministe
  - les états sont des sous-ensembles de Q
  - potentielle explosion exponentielle
- Minimisation
- Limitations : $\{a^n b^n, n \in \mathbb{N}\}$
  (plus faible que les langages hors-contexte)
- Outil très important (analyse lexicale)

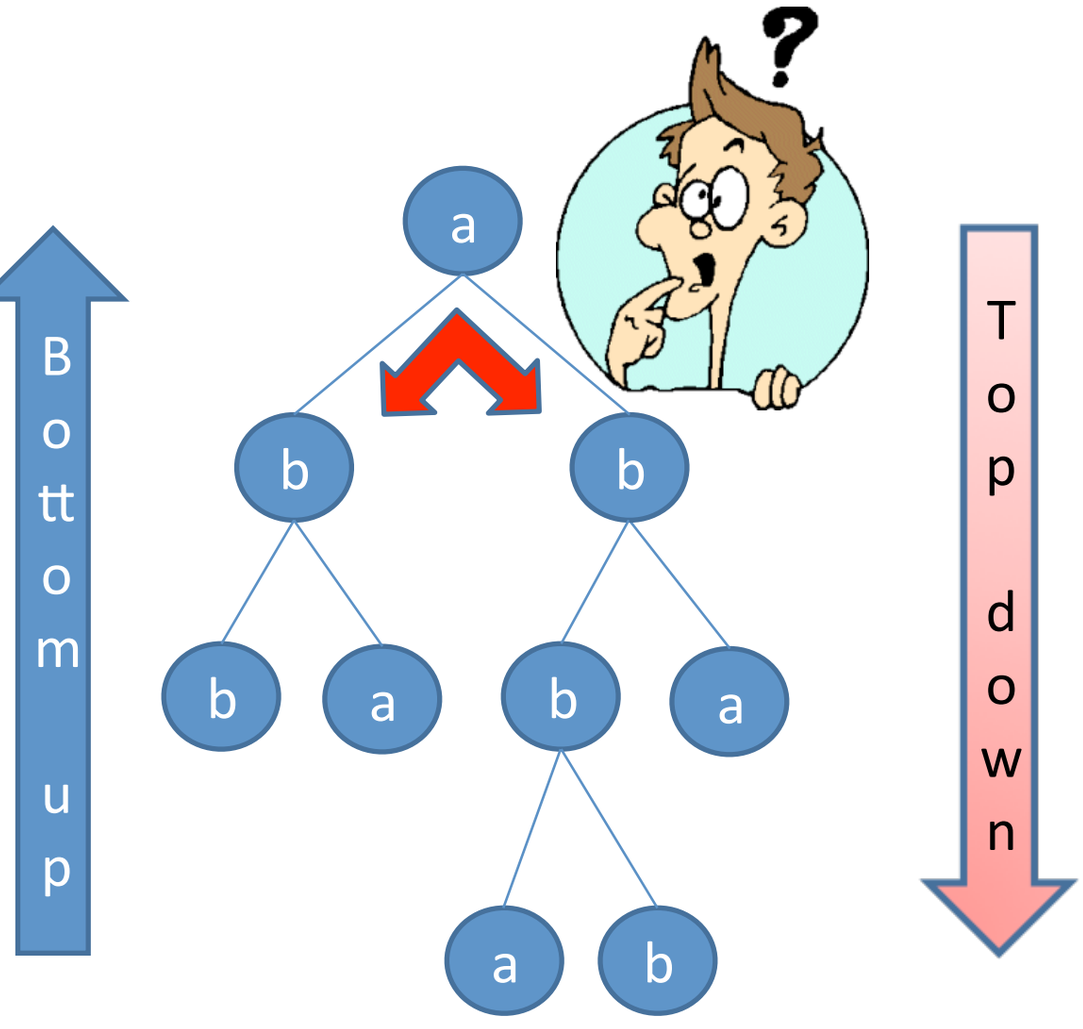
Rappels (suite)

- L(A) = langage accepté par l'automate A
- Langages réguliers
- Peuvent être décrits par des expression régulières : a(b+c)*d
- Fermés par complément : $\Sigma^* - L(A)$
- Fermés par union, intersection : $L(A) \cap L(B)$ $L(A) \cup L(B)$
  produit d'automates avec états (s,s')
  où s (resp. s') est un état de A (resp. A')

Automates sur mots/ sur arbres



Pas de différence



Différences...

II - Automates

- sur arbres d'arité fixe -

Arbres binaires

- Evaluation parallèle

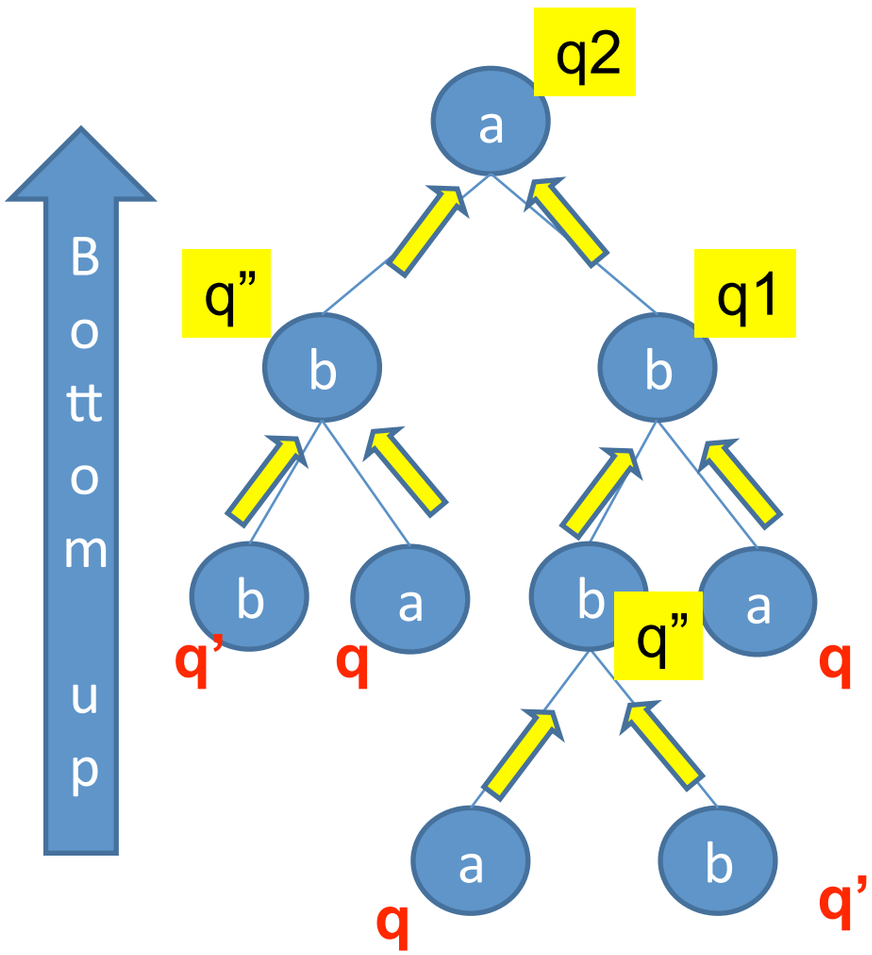
$$(\Sigma, Q, F, \delta)$$

- Pour les feuilles :

$$\delta : \Sigma \rightarrow P(Q)$$

- Pour les autres noeuds :

$$\delta : \Sigma \times Q \times Q \rightarrow P(Q)$$



Automates d'arbres ascendants

$$\delta(a, q, q') \rightarrow \{r, r'\}$$

- Ascendant : si un noeud, étiqueté par a, a ses deux fils dans les états q et q', alors ce noeud passe de façon non déterministe à l'état r ou r'
- Arbre accepté si sa racine est dans un état de F
- Non déterministe si transitions ε ou choix :

$$\delta(a, q, q') \rightarrow \{r, r'\} \qquad \delta(\varepsilon, r) \rightarrow \{r'\}$$

Exemple : circuits booléens

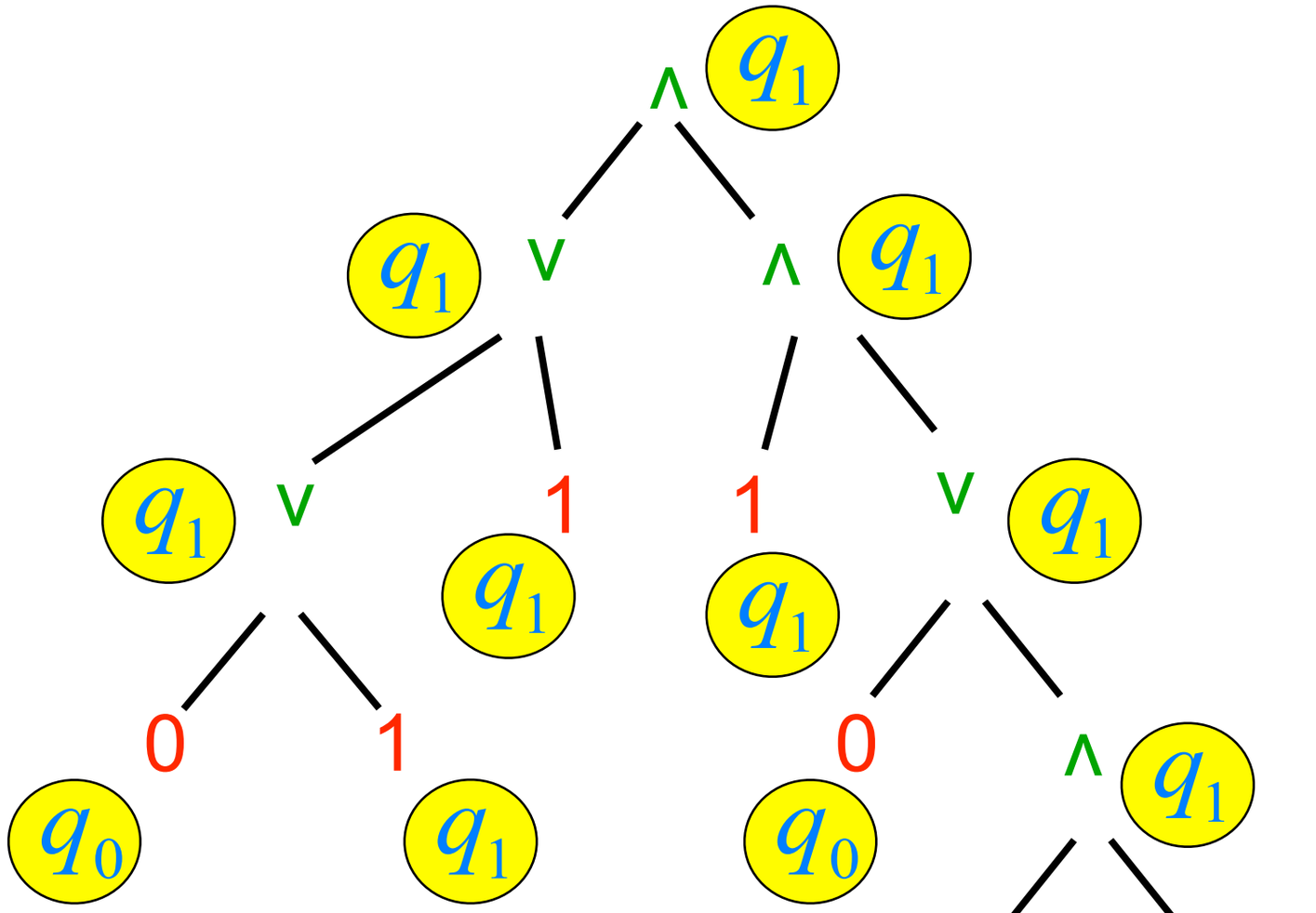
$$\Sigma = \{0,1,\wedge,\vee\}$$
$$Q = \{q_0, q_1\}$$
$$F = \{q_1\}$$

$$\delta_1(0) \rightarrow \{q_0\}$$
$$\delta_1(1) \rightarrow \{q_1\}$$

$$\delta_2(\wedge, q_1, q_1) \rightarrow \{q_1\}$$
$$\delta_2(\wedge, q_0, q_0), \delta_2(\wedge, q_1, q_0), \delta_2(\wedge, q_0, q_1) \rightarrow \{q_0\}$$
$$\delta_2(\vee, q_0, q_0) \rightarrow \{q_0\}$$
$$\delta_2(\vee, q_1, q_1), \delta_2(\vee, q_1, q_0), \delta_2(\vee, q_0, q_1) \rightarrow \{q_1\}$$

Evaluation de circuits booléens

Bottom up



OK

$$\delta_1(0) \rightarrow \{q_0\}$$
$$\delta_1(1) \rightarrow \{q_1\}$$

$$\delta_2(\wedge, q_1, q_1) = \{q_1\}$$
$$\delta_2(\wedge, q_0, q_0) = \{q_0\}$$
$$\delta_2(\wedge, q_1, q_0) \rightarrow \{q_0\}$$
$$\delta_2(\wedge, q_0, q_1) \rightarrow \{q_0\}$$
$$\delta_2(\vee, q_0, q_0) \rightarrow \{q_0\}$$
$$\delta_2(\vee, q_1, q_1) \rightarrow \{q_1\}$$
$$\delta_2(\vee, q_1, q_0) \rightarrow \{q_1\}$$
$$\delta_2(\vee, q_0, q_1) \rightarrow \{q_1\}$$

Langages réguliers

Définition :

Un langage d'arbres est dit *régulier* ssi il est accepté par un automate d'arbre ascendant.

Langages d'arbres réguliers

- Les propositions suivantes sont équivalentes :
  - L est un langage d'arbres régulier
  - L est accepté par un automate d'arbre ascendant non déterministe
  - L est accepté par un automate d'arbre ascendant déterministe
  - L est accepté par un automate d'arbre descendant non déterministe
- Les automates descendants déterministes sont strictement moins expressifs

Automates descendants

$$\delta(a, q'') \rightarrow \{(q, q')\}$$

• Si un noeud étiqueté a est dans un état q”, alors son fils gauche va dans l’état q, et son fils droit dans l’état q’

• On part de la racine depuis un état initial q0.

• L’arbre est accepté si toutes les feuilles sont dans un état final. Pour les feuilles : chgt d’état local.

• Non déterministe si :

$$\delta(a, q'') \rightarrow \{(q, q'), (r, r')\}$$

Pourquoi déterminisme + descendant est plus faible ?

- Considérons le langage L = {f(a,b), f(b,a)}

- L peut être accepté par un automate ascendant A :
  L = L(A) (exercice)

- Supposons que L est accepté par un automate déterministe descendant B : L = L(B)
  Exercice : montrer que B accepte f(a,a)

Fait : aucun automate descendant déterministe n'accepte L

Propriétés arbres d'arité fixe

- Comme les mots, avec une complexité plus élevée
- Déterminisation
- Minimisation
- Fermés par :
  - complément
  - union
  - intersection

Résumé Arbres d'arité fixe égale à d

Automates ascendants

Forme : (Σ, Q, ∂, F)

Transitions :

Feuilles : $\partial : \Sigma \rightarrow P(Q)$

Noeuds internes :

$\partial : \Sigma \times Q^d \rightarrow P(Q)$

Arbre *accepté* ssi il existe un calcul t.q. *la racine est dans F*

Automates descendants

Forme : (Σ, Q, ∂, I)

La racine part de I

Transitions :

Noeuds internes :

$\partial : \Sigma \times Q \rightarrow P(Q^d)$

Feuilles : $\partial : \Sigma \times Q \rightarrow P(Q)$

Arbre *accepté* ssi il existe un *calcul terminant*

Mais...

- Les documents XML sont d'arité variable
- Le type d'arbres que l'on souhaite manipuler :

book (intro,section*,conclusion)

II - Automates

- sur arbres d'arité variable -

Automates sur arbres d'arité variable : cas ascendant

$\delta_2(\wedge, t) \to \{t\}$ $\delta_2(\wedge, t, t) \to \{t\}$ $\delta_2(\wedge, t, t, t) \to \{t\}$...
$\delta_2(\wedge, f) \to \{f\}$ $\delta_2(\wedge, t, f) \to \{f\}$ $\delta_2(\wedge, f, t) \to \{f\}$...
$\delta_2(\vee, t) \to \{t\}$ $\delta_2(\vee, t, f) \to \{t\}$ $\delta_2(\vee, f, t) \to \{t\}$...
$\delta_2(\vee, f) \to \{f\}$ $\delta_2(\vee, f, f) \to \{f\}$ $\delta_2(\vee, f, f, f) \to \{f\}$...

Problème : représenter un ensemble infini de transitions

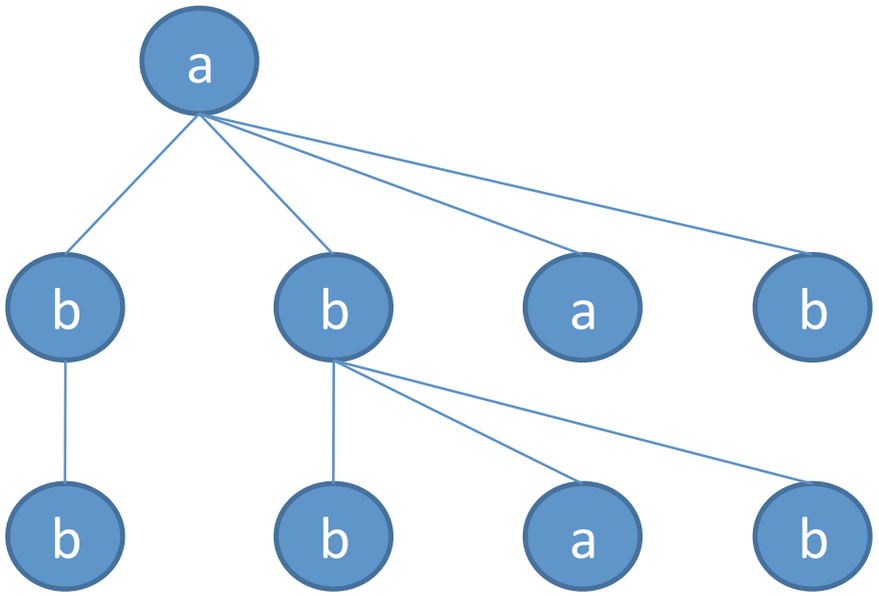
Solution : utiliser des expression régulières

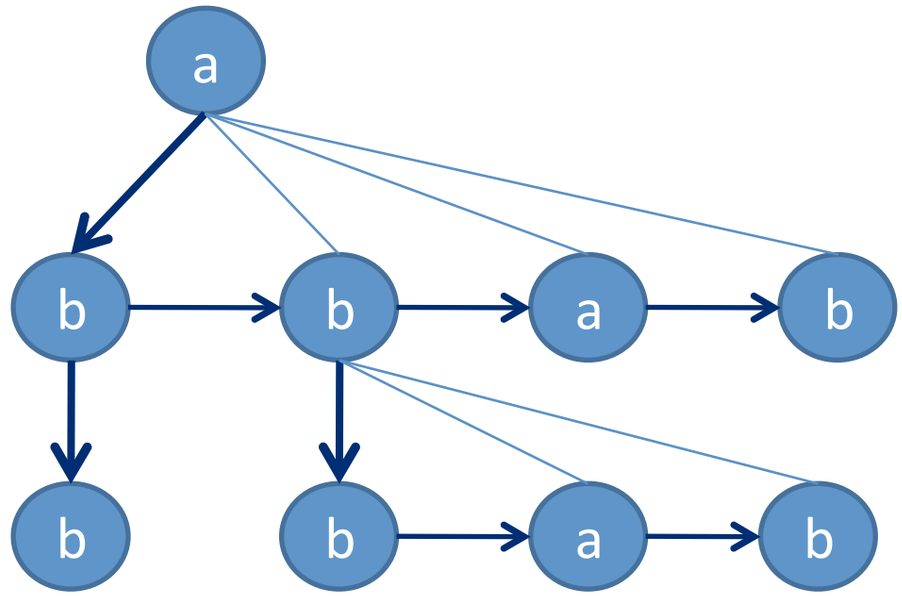
Automates de haies ascendants

- Règles : $\delta(a, L(Q)) \rightarrow \{r_1, \ldots, r_m\}$
- Feuilles : $\partial : \Sigma$->P(Q)
- Sémantique : si les enfants d'un noeud étiqueté par a forment un mot dans L(Q), alors ce noeud passe dans un état de {r1,...,rm}

$$\delta_2(\wedge, And1) \rightarrow \{t\} \quad where \quad And1 = t +$$
$$\delta_2(\wedge, And0) \rightarrow \{f\} \quad where \quad And0 = (t + f)^* f (t + f)^*$$
$$\delta_2(\vee, Or1) \rightarrow \{t\} \quad where \quad Or1 = (t + f)^* t (t + f)^*$$
$$\delta_2(\vee, Or0) \rightarrow \{f\} \quad where \quad Or0 = f +$$

Construction à partir du cas borné





Arbre de degré 2 : au plus 2 fils
Premier fils - Prochain frère

F: codage dans un arbre d'arité 2 :
- F est une bijection

F^{-1}: décodage

Construction à partir du cas borné (2)

- Pour tout automate de haies A, il existe un automate d'arbres d'arité 2 acceptant F(L(A))
- Pour tout automate d'arbre A d'arité 2, il existe un automate de haies acceptant $F^{-1}(L(A))$
- Les deux sont faciles à construire

Conséquence : les langages de haies sont clos par union, intersection, complément.

Déterminisation

- Déf. de déterministe dans le cas ascendant d'arité variable :

$$\forall \alpha \in \Sigma, \forall w \in Q^*, \text{ there exists a unique rule } \delta(\alpha, L)$$
$$\text{such that } w \in L.$$

- Tout automate d'arbre ascendant d'arité variable peut être déterminisé
- Peut-on utiliser l'encodage "Premier fils - Prochain suivant" ?

  Non ! il ne préserve pas le déterminisme

Automates descendants ?

- On part de la racine et on descend
- On lit l'étiquette du noeud, son état, et on donne un état à chaque fils, selon une expression régulière
- Au lieu de donner un tuple d'états (cas d'arité fixe), on donne une expression régulière sur les états.

Cas descendant (suite)

Forme : (∑,Q,∂,I) ; La racine part de I

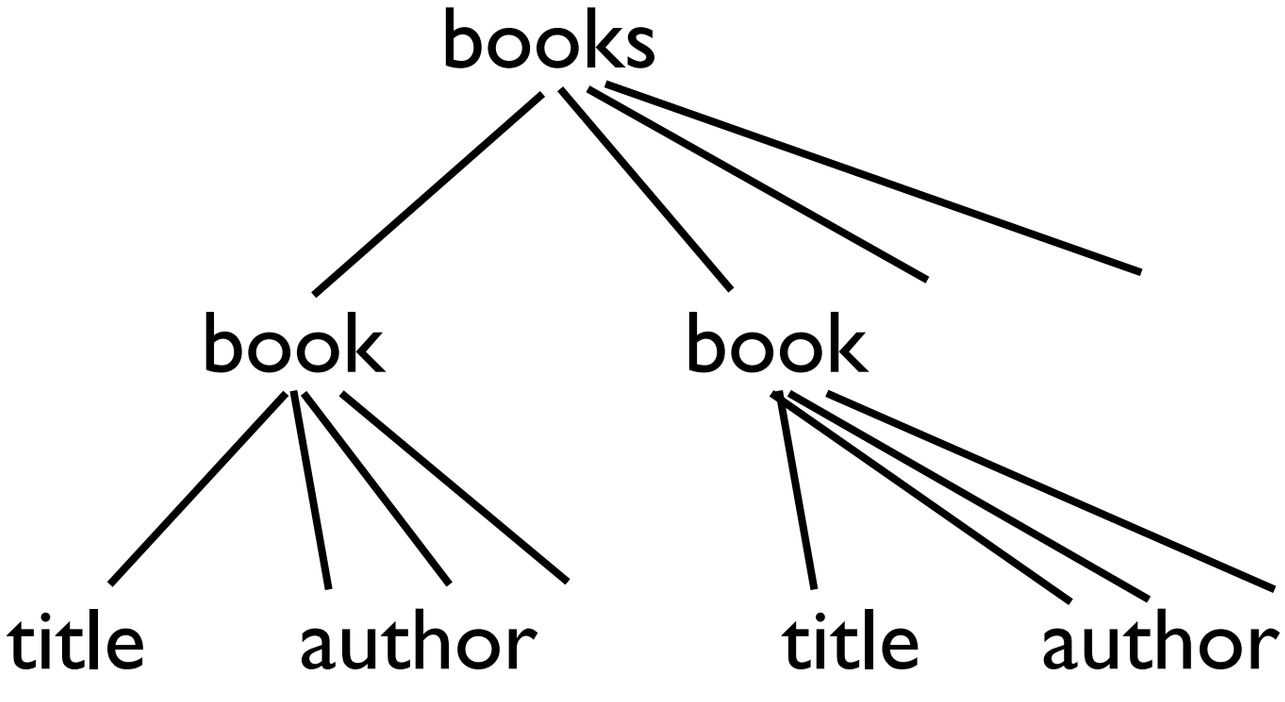
Transitions : Noeuds internes : ∂ : ∑xQ -> R
où R est dans Reg(Q)

Arbre *accepté* ssi il existe un *calcul terminant*

L'automate est *déterministe* s'il propose au plus une expression rationnelle.

Remarque : Cette expression est toujours représentable par un automate fini déterministe.

Cas descendant : exemple



- Un élément racine books possédant des fils book
- Un élement book possède un fils title et (au moins un) fils author

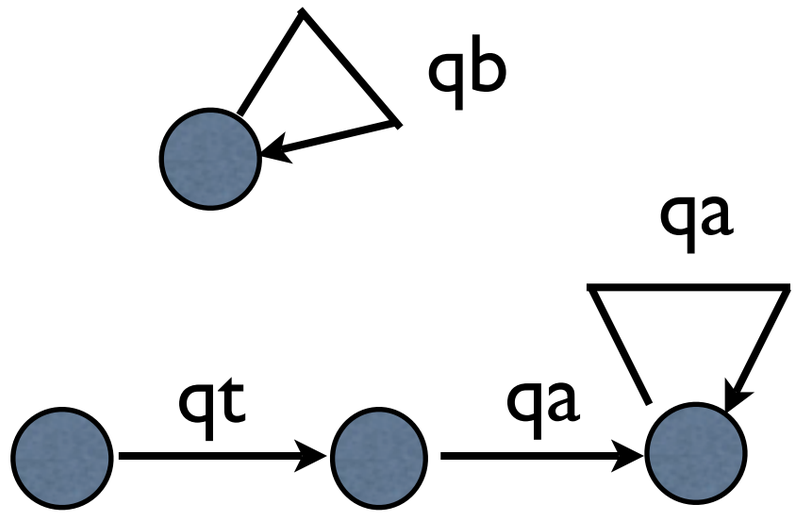
$\sum$ = {books,book,title,author}

Q = {q0,qb,qs,qt,qa,qf} I = {q0}

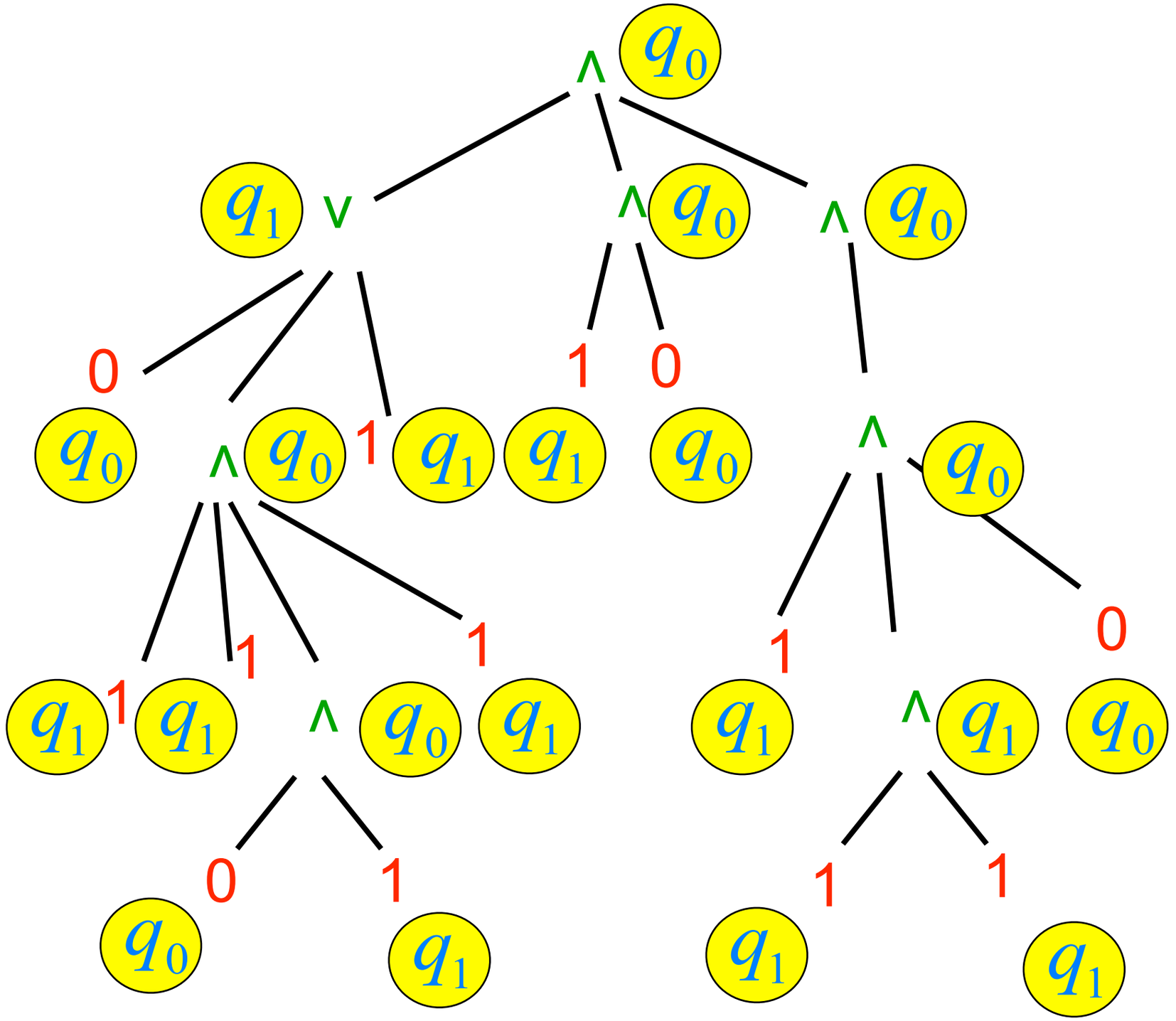
∂(books,q0) ={qb*}

∂(book,qb) ={qt.qa^+}

∂(title,qt)= ∂(author,qa)=qf



Evaluation de circuits booléens



Résumé Arbres d'arité variable

Automates ascendants

Forme : (Σ, Q, ∂, F)

Transitions :

Feuilles : $\partial : \Sigma \rightarrow P(Q)$

Noeuds internes :

$\partial : \Sigma \times Reg(Q) \rightarrow P(Q)$

Arbre *accepté* ssi il existe un calcul t.q. *la racine est dans F*

Automates descendants

Forme : (Σ, Q, ∂, I)

La racine part de I

Transitions :

Noeuds internes :

$\partial : \Sigma \times Q \rightarrow P(Reg(Q))$

Arbre *accepté* ssi il existe un *calcul terminant*