HMIN103
Données du Web

Federico Ulliana
UM, LIRMM, INRIA GraphIK

Slides collected from J. Cheney, S. Abiteboul, I. Manolescu, P. Senellart, P. Genevès, D. Florescu and the W3C

Organisation des TD/TP

- Début des TD/TP (6.07,6.08 dans la limite des places +6.10)

  Groupe 1- AIGLE 6.07 Groupe 2 – DECOL 6.08

- Vendredi + Jeudi (mais, à partir du 10 octobre ☺)
- Jeudi après-midi (avant la toussaints)
  - Pour les inscrits à HMIN121M => 2 séances au vendredi
- Jeudi matin (après la toussaints)
  - Pour les inscrits à HMIN110 => 2 séances au vendredi
- Pas de TD/TP ni CM le 3 (jeudi) et 4 (vendredi) octobre, ni le 26 (jeudi) sept.
- CM 8h à partir de la semaine prochaine (27 sept.)

Rappels du dernier cours

- XML
  - Motivations pour l'introduction du standard
  - Données arborescentes
  - Éléments et attributs
- DTDs
  - Déclarations (ex. HTML : https://www.w3.org/TR/html401/sgml/dtd.html)
  - Intégrité des données : ID / IDRef

Une DTD validant le document ?

```
<D>
  <A/>
  <B>hello</B>
  <C/>
</D>
```

Une DTD validant le document ?

```
<D>
  <A/>
  <B>hello</B>
  <C/>
</D>

<!DOCTYPE D [

 <!ELEMENT D (A,B,C)>

 <!ELEMENT A (#PCDATA)>
 <!ELEMENT B (#PCDATA)>
 <!ELEMENT C (#PCDATA)>   ]>
```

Une DTD validant le document ?

```
 <D>
   <A/>
   <B>hello</B>
   <C/>
 </D>

<!DOCTYPE D [

 <!ELEMENT D (A*,B,C,D?)>

 <!ELEMENT A EMPTY>
 <!ELEMENT B (#PCDATA)>
 <!ELEMENT C ANY>   ]>
```

Une DTD validant le document ?

```
<C>
    <C id=”p4” friend=”p4”>Alice</C>
    <C id=“p1” friend=”p2”>Bob</C>
    <C id=“p2” friend=”p0”>Alice</C>
</C>
```

Une DTD validant le document ?

```
<C>

    <C id=”p4” friend=”p4”>Alice</C>

    <C id=“p1” friend=”p2”>Bob</C>

    <C id=“p2” friend=”p0”>Alice</C>

</C>
```

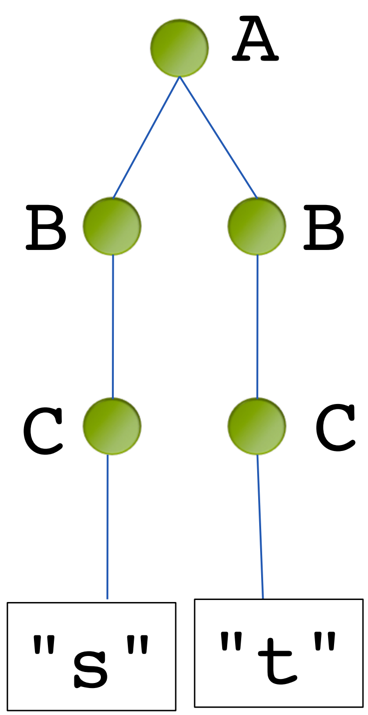
```
               <!DOCTYPE C [    <!ELEMENT C (C*)>

               <!ATTLIST id ID CDATA  IMPLIED>

               <!ATTLIST C friend IDREF CDATA>   ]>
```

Un document valide pour la DTD ?

```
<! DOCTYPE A [
   <! ELEMENT A (B,B) >
   <! ELEMENT B (C) >
   <! ELEMENT C (#PCDATA) >
  ]>
```

Un document valide pour la DTD ?

```
<! DOCTYPE A [
   <! ELEMENT A (B,B) >
   <! ELEMENT B (C) >
   <! ELEMENT C (#PCDATA) >
  ]>
```



Un document valide pour la DTD ?

```
<! DOCTYPE B [
 <! ELEMENT B (A,C)
 <! ELEMENT C (D)
 <! ELEMENT A (D)
  <! ELEMENT C (#PCDATA) >
  <! ELEMENT D (#PCDATA) >
 ]>
```

Un document valide pour la DTD ?

```
<! DOCTYPE A [

 <! ELEMENT A (B) >
 <! ELEMENT B (A) >

]>
```

Un document valide pour la DTD ?

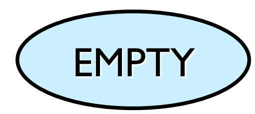
```
<! DOCTYPE C [

  (! ELEMENT C (C*) )

 ]>
```

Un document valide pour la DTD ?

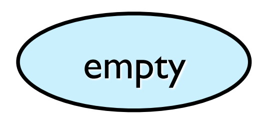
```
<! DOCTYPE EMPTY [

 <! ELEMENT EMPTY EMPTY >

]>
```

Un document valide pour la DTD ?

```
<! DOCTYPE EMPTY [
  <! ELEMENT EMPTY EMPTY >
]>
```




```
<! DOCTYPE empty [
  <! ELEMENT empty EMPTY >
]>
```

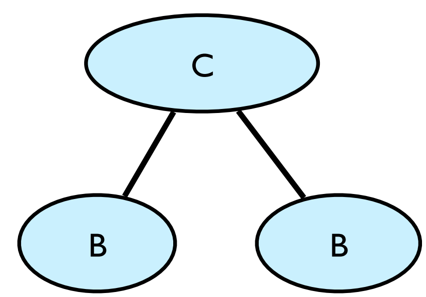


Un document valide pour la DTD ?

```
<! DOCTYPE C [

 <! ELEMENT C (B* | (C, C, C, C)) >

]>
```

Un document valide pour la DTD ?

```
<! DOCTYPE C [
  <! ELEMENT C (B* | (C, C, C, C)) >
]>
```

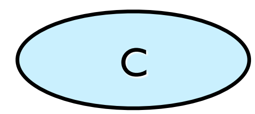


Un document valide pour la DTD ?

```
<! DOCTYPE C [
 <! ELEMENT C (C,EMPTY)? >
]>
```

Un document valide pour la DTD ?

```
<! DOCTYPE C [

 <! ELEMENT C (C,EMPTY)? >

]>
```

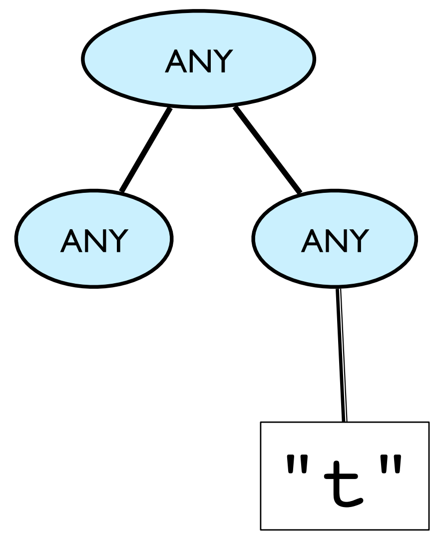


Un document valide pour la DTD ?

```
<!DOCTYPE ANY [

<!ELEMENT ANY ANY>

]>
```

Un document valide pour la DTD ?

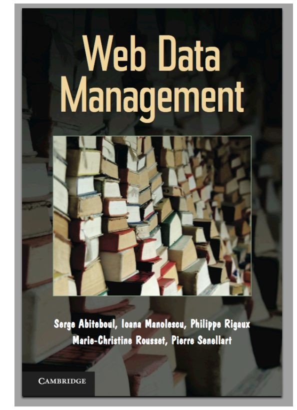
```
<!DOCTYPE ANY [

<!ELEMENT ANY ANY>

]>
```



DTDs and Regular Tree Grammars

(fun with regular expressions)

Readings



- Web Data Management - Abiteboul & al.
- [WDM-XML] Chapter : Data-model
  - http://webdam.inria.fr//Jorge/files/wdm-datamodel.pdf
- [WDM-DTD] Chapter : Schemas (only section 3)
  - http://webdam.inria.fr//Jorge/files/wdm-typing.pdf

Schemas for XML Data

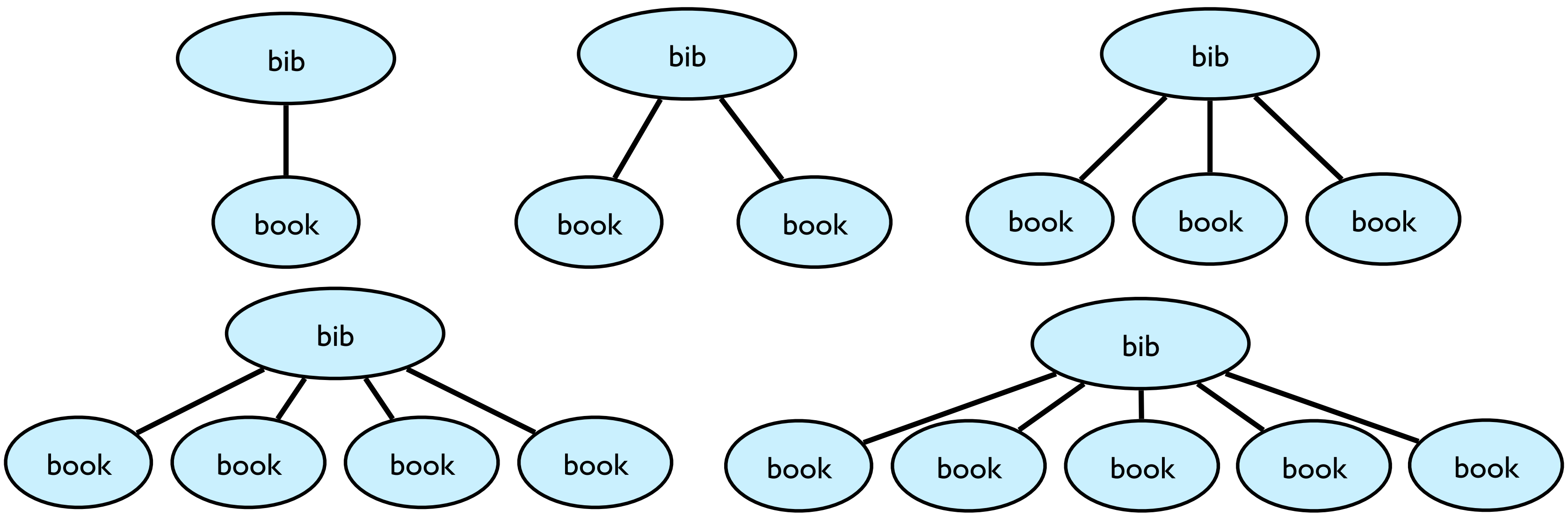
- Many schema languages/formalisms have been proposeed
  - DTD (XML 1.0)
  - XML Schema (W3C)
  - Relax/NG (OASIS), DSD, Schematron, ...
  - Regular expression types (XDuce, XQuery)

- Every XML schema language is based on regular expressions and grammars.
  - This illustrates an important use of theory in real applications.

Regular Tree Grammars

A DTD defines a (possibly infinite) set of **regular** XML trees.

```
<!ELEMENT bib book+>        <!ELEMENT book EMPTY>
```



Plan

- Grammars
- Validation
- Determinism

Grammars (context-free)

Sets of rules used to specify a formal language (of words).

```
S -> PQ          P -> aP|a          Q -> b
```

Grammars (context-free)

Sets of rules used to specify a formal language (of words).

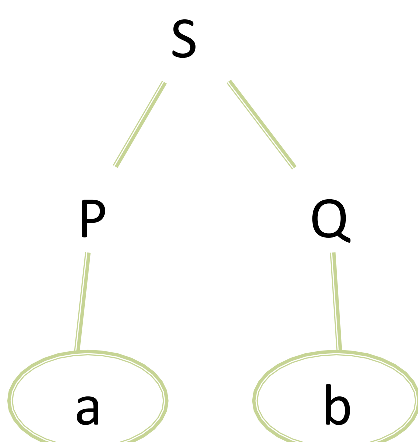
```
S -> PQ          P -> aP|a          Q -> b
```

```
ab

aab

aaab

aaaab
```

Grammars (context-free)

Sets of rules used to specify a formal language (of words).

`S -> PQ`   `P -> aP|a`   `Q -> b`

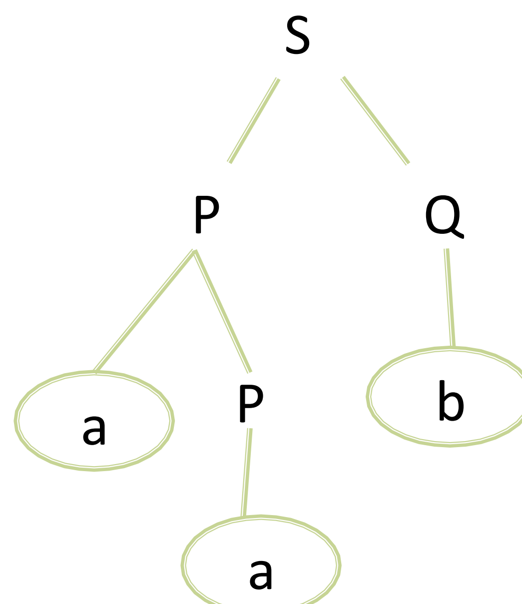


`ab`

`aab`

`aaab`

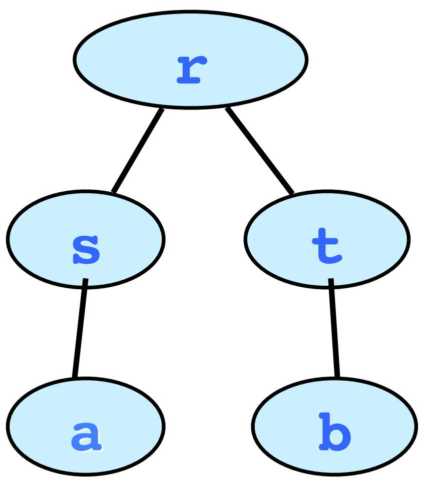
`aaaab`



Tree Grammars

Sets of rules used to specify a formal language of trees.

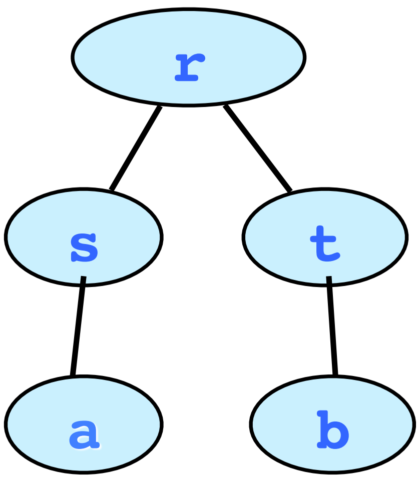
```
S -> r[PQ]          P -> s[a]          Q -> t[b]
```

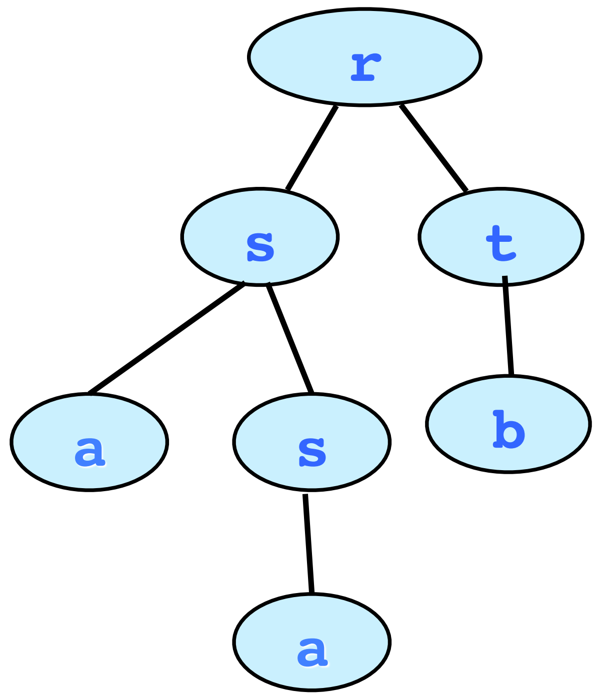


Tree Grammars

Sets of rules used to specify a formal language of trees.

```
S -> r[PQ]          P -> s[aP|a]        Q -> t[b]
```



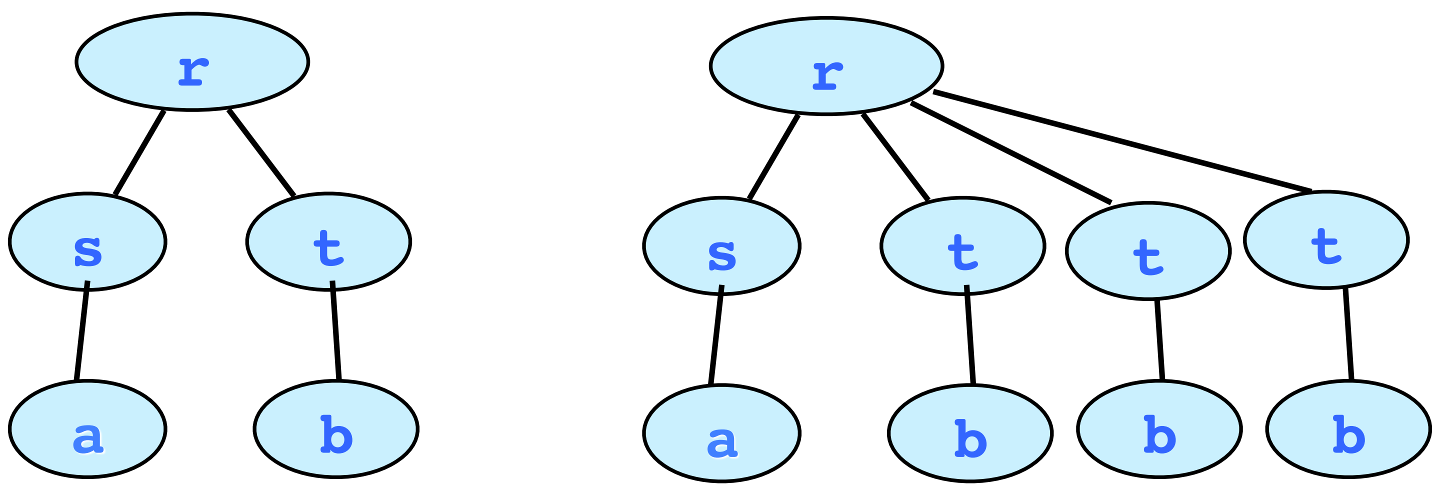


Vertical recursion

Tree Grammars

Sets of rules used to specify a formal language of trees.

`S -> r[PQ]`   `P -> s[a]`   `Q -> t[b],Q |t[b]`

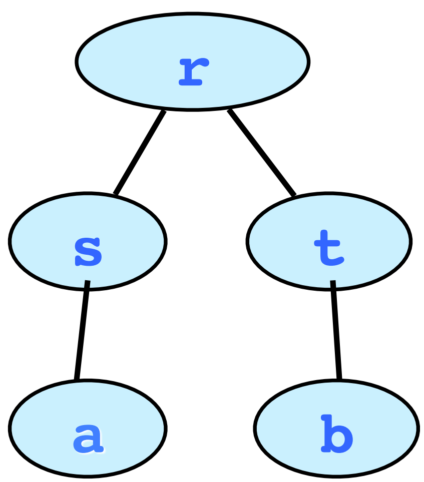


Horizontal recursion

Regular Tree Grammars

Sets of rules used to specify a formal regular language of trees.

```
S -> r[P,Q]        P -> s[aP|a]        Q -> t[b],Q |t[b]
```



Forbid certains uses of horizontal recursion

```
Q -> t[b],Q        OK

Q -> Q,t[b],Q      NO
```

(analogous to the definition of regular word grammars)

Regular Tree Grammars

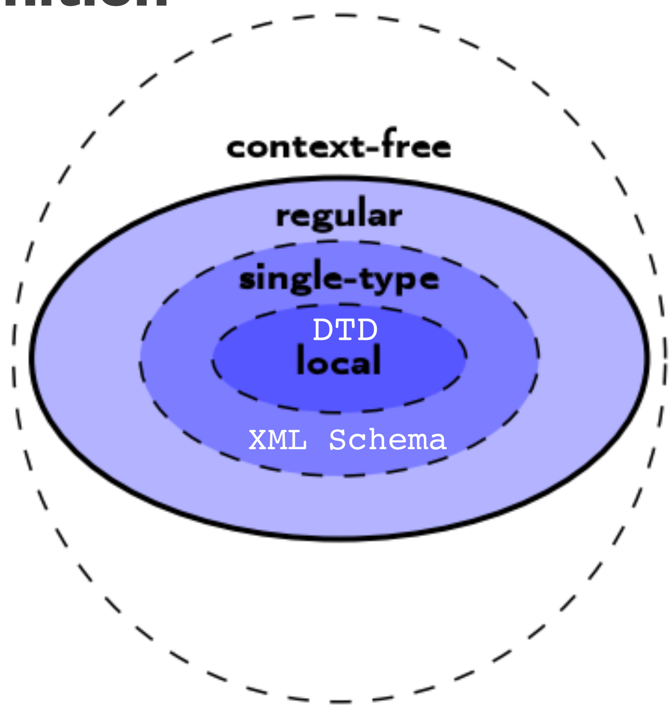
DTD are a subclass of regular tree-grammars called “**local**” :

any element has at most one definition

<!ELEMENT root child*>

<!ELEMENT child (#PCDATA)>

~~<!ELEMENT child EMPTY>~~



Why Regular Tree Grammars ?

- Regular Tree Grammars are expressive enough, and computationally more easy to handle than context-free
  - To illustrate, the following problems for context-free **tree** grammars <u>cannot be algorithmically solved</u> :
    - determine wether a context-free grammar is actually a regular grammar
    - determine wether a context-free grammar G1 is more general than (or, "includes") a context-free grammar G2

XML VALIDATION

Document Validation

- Problem : is an XML document valid with respect to a given DTD ?

Validation Algorithm

- Traverse XML tree in pre-order (document order) & check:

1. that each node is valid
2. that each attribute (of a node) is valid
3. the id-unicity and idref-references

Validation Algorithm

- Traverse XML tree in pre-order (document order) & check:

1. that each node is valid
2. that each attribute (of a node) is valid
3. the id-unicity and idref-references

Validation Algorithm

- Traverse XML tree in pre-order and check

1. that each node is valid

   more interesting

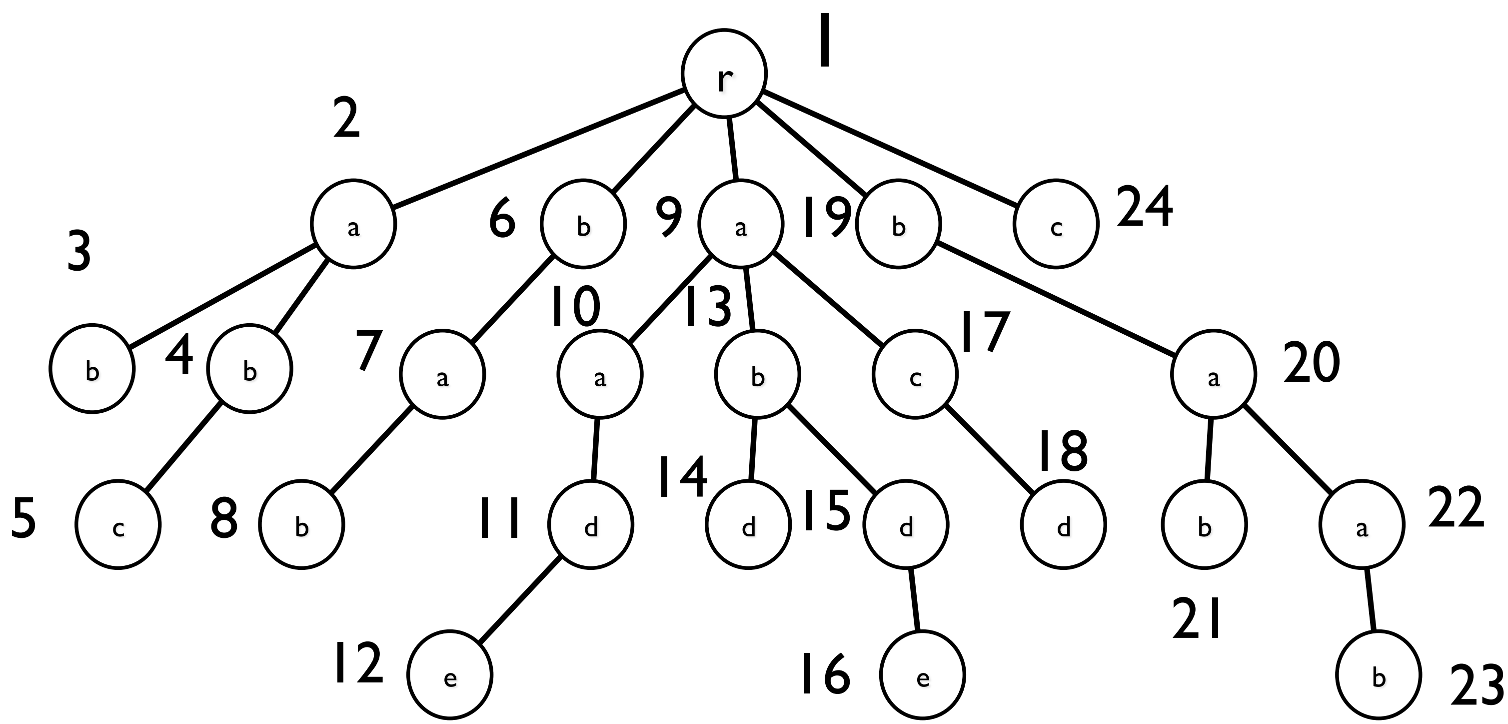
2. that each attribute (of a node) is valid

   trivial

3. the id-unicity and idref-references

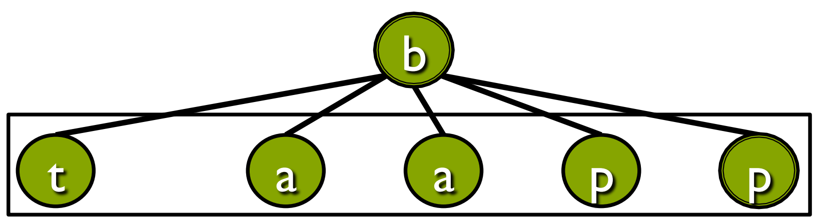
   easy

Pre-order Traversal



(Single Node) Validation

- Problem : does the sequence of children of the node match the regular expression specified by the DTD ?



Regular expressions

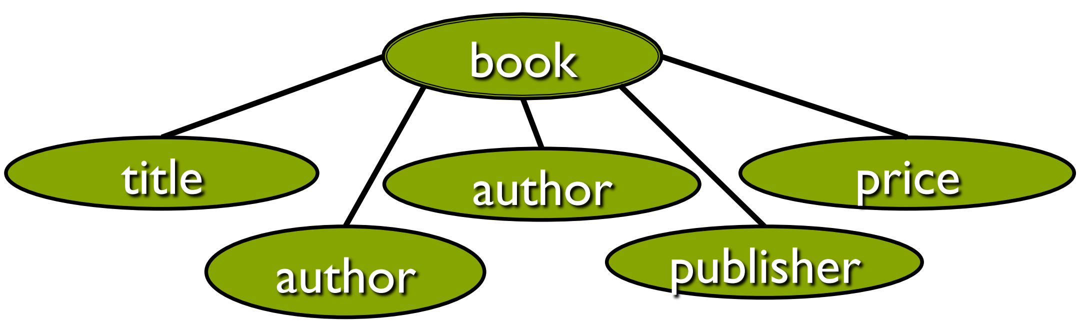
`r ::=`	ϵ	empty sequence
`|`	`a`	atomic symbol (in DTD, an element name)
`|`	`(r,s)`	sequential composition
`|`	`(r|s)`	union (alternation)
`|`	`(r+)`	repetition

`r* = r+|`ϵ `r? = r|`ϵ

Example

The regular expression for a book node is

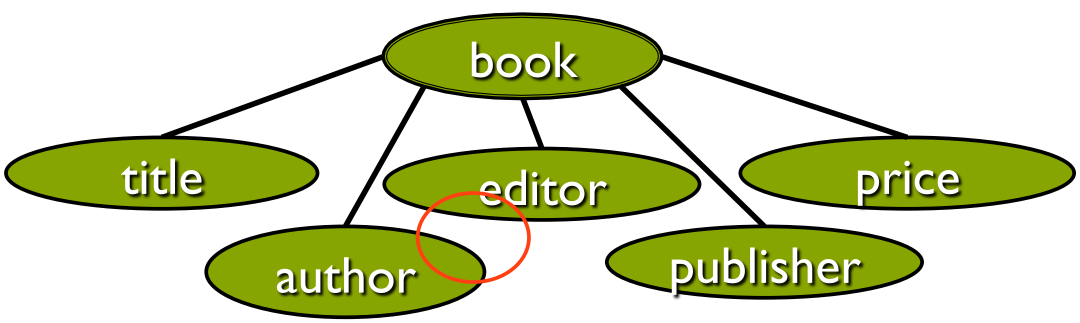
```
(title, (author+ | editor+ ), publisher, price )
```



Example

The regular expression for a book node is

```
(title, (author+ | editor+ ), publisher, price )
```



DETERMINISM

W3C Restriction

- Can we write any regular expression in a DTD ?
- NO.

- Regular expressions in DTDs must be deterministic:
  - *"there must be only one way to match any sequence of tags, no backtrack or look-ahead is required"*
- This is equivalent to say that the automata corresponding to the regular expression is <u>deterministic</u>.
- This eases the validation process, and makes it doable in <u>streaming</u>

Example of Ambiguity

`( title, author* ) | ( title , editor* )` **NO**

can't decide if `<title/>` matches first or second "`title`"

Better to write `title , ( author* | editor* )`

How to test Determinism?

- Simplified algorithm

- Ingredients : three auxiliary functions

FirstTag() LastTag() FollowsTag()

(1/3) FirstTag

*What can be the **first** tag of a sequence matching r ?*

r_1 = `(title, (author+ | editor+ ), publisher, price )`

FirstTag(r_1) ? `title`

r_2 = `(author+ | editor+ )`

FirstTag(r_2) ? `author, editor`

(2/3) LastTag

*What can be the **last** tag of a sequence matching r ?*

r_1 = `(title, (author+ | editor+ ), publisher, price )`

LastTag(r_1) ? `price`

r_2 = `(author+ | editor+ )`

LastTag(r_2) ? `author, editor`

(3/3) FollowsTag

What tag can follow x in r ?

r_1 = `(title, (author+ | editor+ ), publisher, price )`

FollowsTag(r_1 , `title`) ? `author, editor`

r_4 = `(author | editor )*`

FollowsTag(r_4 , `author`) ? `author, editor`

Disambiguation

r_5 = `(author, title)? , author`

We resolve ambiguity by enumerating the tag occurrences

$r_5^\#$ = $\texttt{(author}_1\texttt{, title)? , author}_2$

FirstTag($r_5^\#$) = $\texttt{author}_1\texttt{, author}_2$

LastTag($r_5^\#$) = $\texttt{author}_2$

FollowsTag($r_5^\#$, `title`) = $\texttt{author}_2$

Determinism Algorithm

1) Enumerate all the occurrences of a tag in r

2) Build a graph were

- there is a node x for each tag in ($r^{\#}$), plus a source-node x_0

- there is a directed edge (x_0,y) if y belongs to FirstTag($r^{\#}$)

- there is a directed edge (x,y) if y belongs to FollowsTag($r^{\#}$,x)

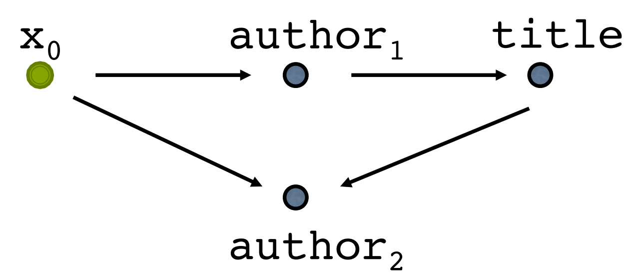
3) return **false** if there exists edges (x,y_i) and (x,y_j) with $i \neq j$

4) return **true** otherwise

Testing Determinism

r_5 = `(author, title)? , author`

$r_5^{\#}$ = $(\texttt{author}_1, \texttt{title})? , \texttt{author}_2$



$\text{FirstTag}(r_5^{\#}) = \texttt{author}_1, \texttt{author}_2$

$\text{FollowsTag}(r_5^{\#}, \texttt{author}_1) = \texttt{title}$

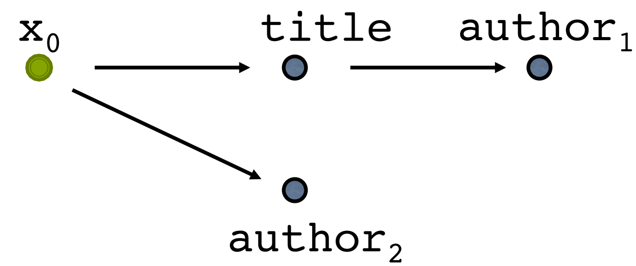
$\text{FollowsTag}(r_5^{\#}, \texttt{title}) = \texttt{author}_2$

r_5 not deterministic

Testing Determinism

r_6 = `(title, author) | author`

$r_6^{\#}$ = $(\texttt{title}, \texttt{author}_1) \mid \texttt{author}_2$



FirstTag($r_6^{\#}$) = $\texttt{title}, \texttt{author}_2$

FollowsTag($r_6^{\#}$,title) = $\texttt{author}_1$

r_6 deterministic

Determinism - Quiz

Are the following regular expressions deterministic ?

- `((e|cb),b)*((cc|b)e,d)*`
- `(a,(ab|c))|(b,(a|c))`

Why did we define **LastTag**(r) afterall ?

- It is hidden behind the definition of FollowsTag(r,x)

FirstTag()

Definition on the structure of the regular expression

- FirstTag(ϵ) = {}
- FirstTag(`a`) = { `a` }
- FirstTag(`r|s`) = firstTag(`r`) U firstTag(`s`)
- FirstTag(`r*`) = firstTag(`r`)
- FirstTag(`r,s`) = firstTag(`r`)

FirstTag()

Definition on the structure of the regular expression

- FirstTag(ϵ) = {}
- FirstTag(`a`) = { `a` }
- FirstTag(`r|s`) = ?
- FirstTag(`r*`) = ?
- FirstTag(`r,s`) = ?

FirstTag()

Definition on the structure of the regular expression

- FirstTag(ϵ) = {}
- FirstTag(`a`) = { `a` }
- FirstTag(`r`|`s`) = firstTag(`r`) U firstTag(`s`)
- FirstTag(`r`*) = firstTag(`r`)
- FirstTag(`r`,`s`) = firstTag(`r`) [U firstTag(`s`) IF `r` matches ϵ]

LastTag()

Definition on the structure of the regular expression

- LastTag(ϵ) = ?
- LastTag(a) = ?
- LastTag(r | s) = ?
- LastTag(r*) = ?
- LastTag(r , s) = ?

LastTag()

Definition on the structure of the regular expression

- LastTag(ϵ) = {}
- LastTag(a) = { a }
- LastTag(r | s) = LastTag (r) U LastTag (s)
- LastTag(r*) = LastTag(r)
- LastTag(r , s) = LastTag (s)   [U LastTag(r) IF s matches ϵ]

FollowsTag()

Definition on the structure of the regular expression

- FollowsTag(ϵ) = ?
- FollowsTag(`a`) = ?
- FollowsTag(`r` | `s`) = ?
- FollowsTag(`r*`) = ?
- FollowsTag(`r` ,`s`) = ?

FollowsTag()

Definition on the structure of the regular expression

- FollowsTag(ϵ) = {}
- FollowsTag(`a`) = { }
- FollowsTag(`r` | `s`) = FollowsTag (`r`) U FollowsTag (`s`)
- FollowsTag(`r*`) = FollowsTag (`r`, `r`)
- FollowsTag(`r` ,`s`) = FollowsTag (`r`) U FollowsTag (`s`)

  U LastTag(`r`) x FirstTag(`s`)

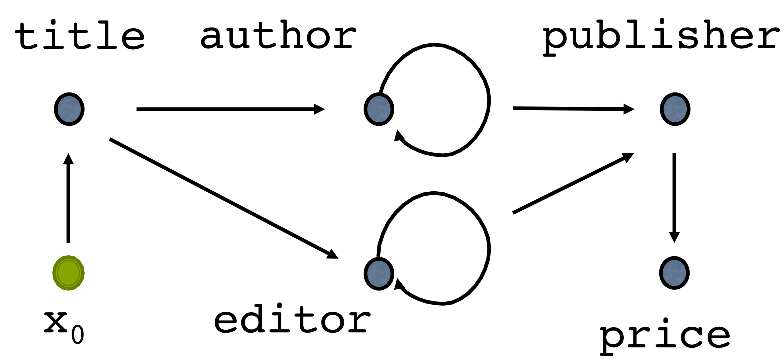
... now back to Node Validation

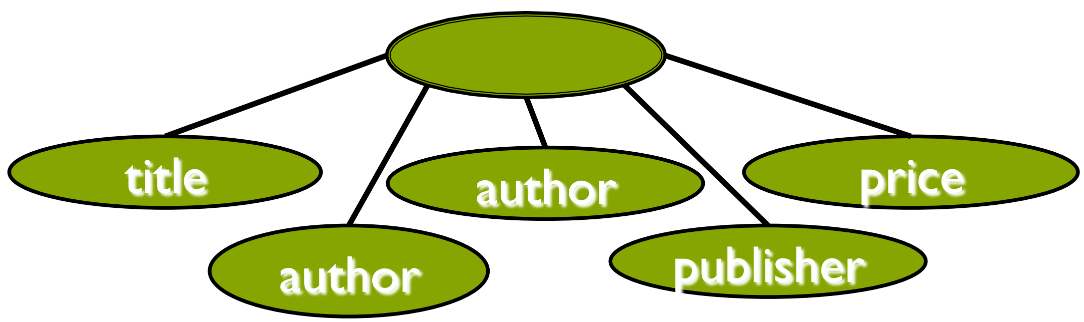
good news : this comes almost for free now!

- `check if the sequence defines a path`
  `in the graph which ends on a LastTag(r`$^{\#}$`)`

Sequence Validation

r = `(title, (author+ | editor+ ), publisher, price )`

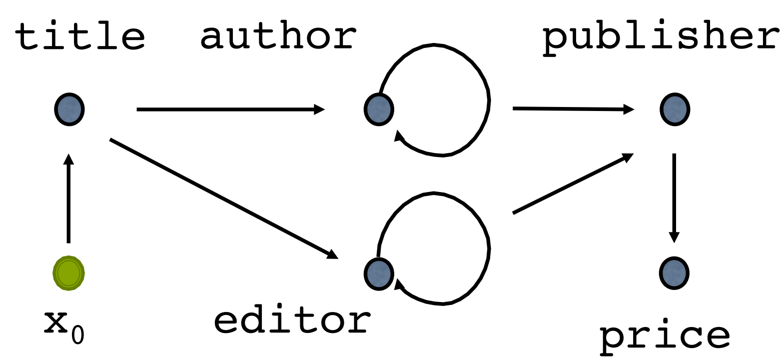


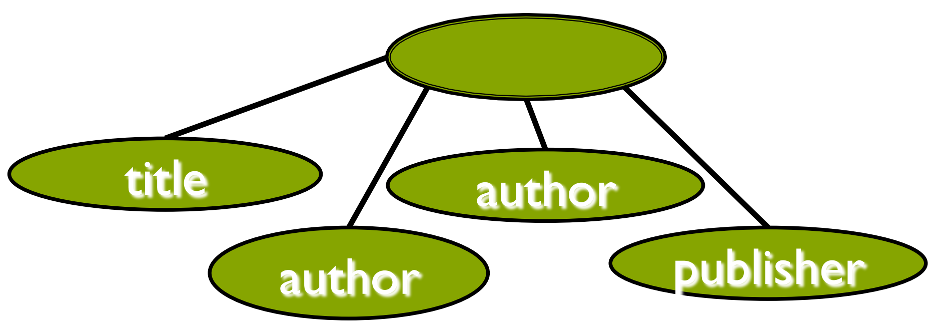


OK

Sequence Validation

`r = (title,  (author+ | editor+ ), publisher, price )`

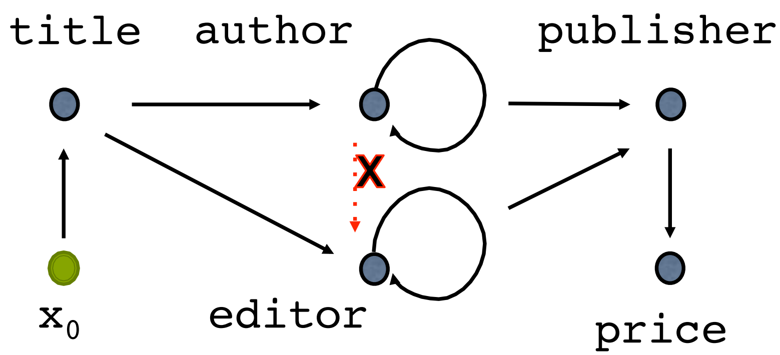


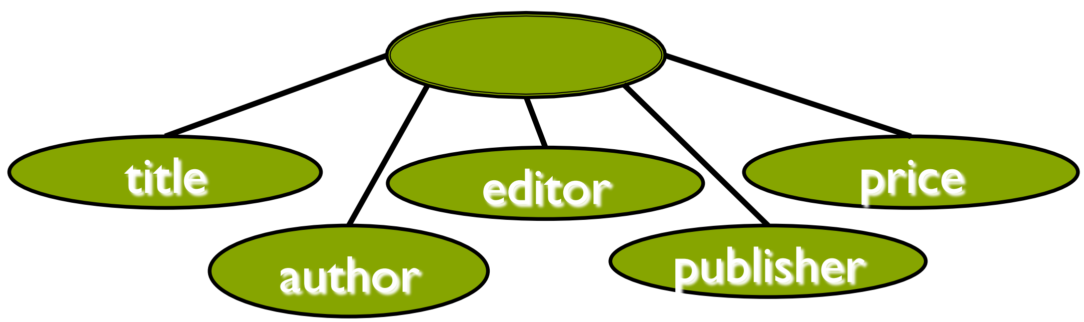


NO

Sequence Validation

r = `(title, (author+ | editor+ ), publisher, price )`



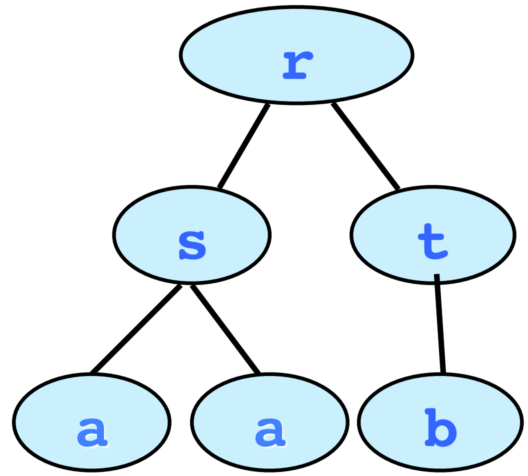


NO

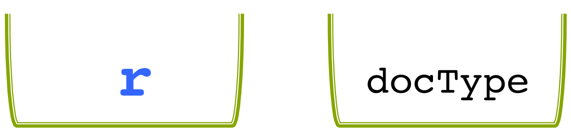
Document Validation Algorithm

- set **last** := root ; stackDTD.push(docType)
- for every node **n (!= root)** taken in a pre-order visit of the tree
- if **last** is the parent of **n** //moving down parent -> child
  - create new list **L** ; add n to **L**
  - stackXML.push(**L**) ; stackDTD.push(typeDTD(**last**))
- if n is the last of its siblings //next move up child -> parent
  - stackXML.top.add(**n**)
  - stackXML.top.isValid(stackDTD.top())
  - stackXML.pop() ; stackDTD.pop() //empty buffers
- else //move left child -> sibling
  - stackXML.top.add(**n**)

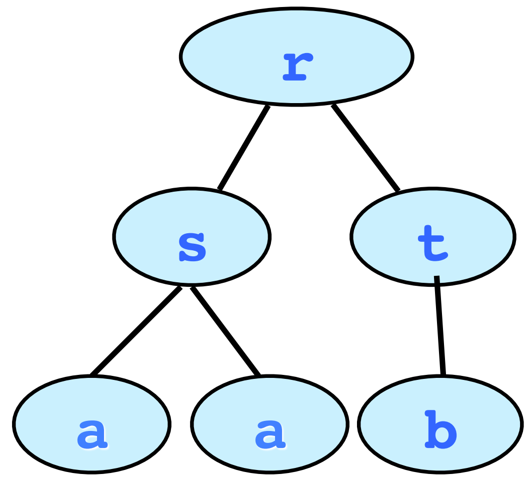
Validation Example

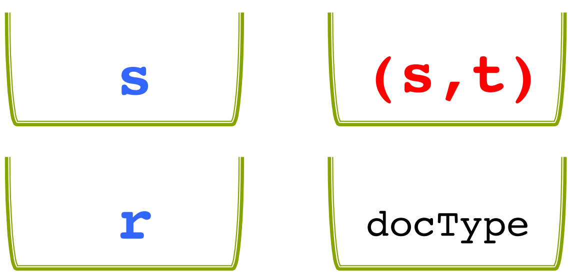



```
<!DOCTYPE r [
<!ELEMENT r (s,t)>
<!ELEMENT s (a*)>
<!ELEMENT t (b?)>
]>
```



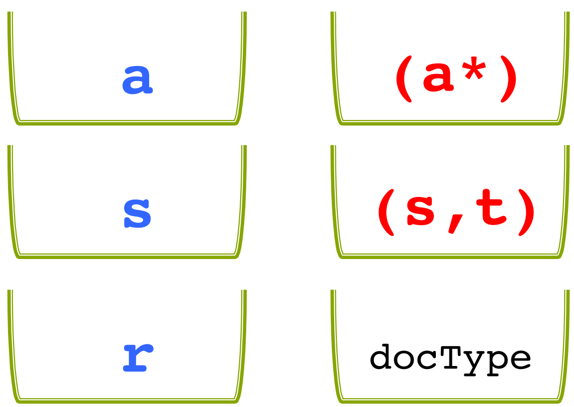
Validation Example

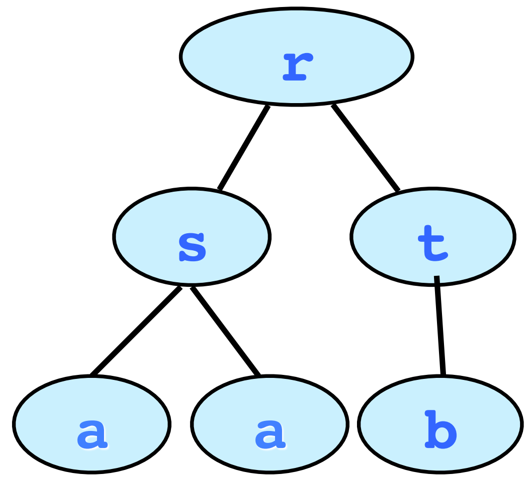





```
<!DOCTYPE r [
<!ELEMENT r (s,t)>
<!ELEMENT s (a*)>
<!ELEMENT t (b?)>
]>
```

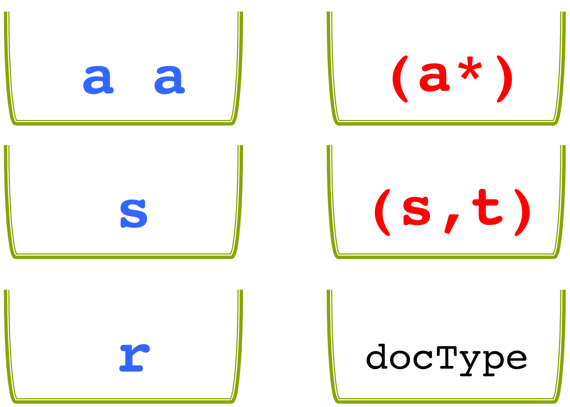
Validation Example

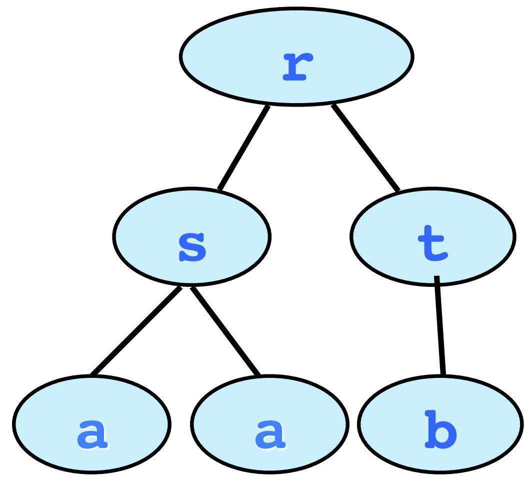





```
<!DOCTYPE r [
<!ELEMENT r (s,t)>
<!ELEMENT s (a*)>
<!ELEMENT t (b?)>
]>
```

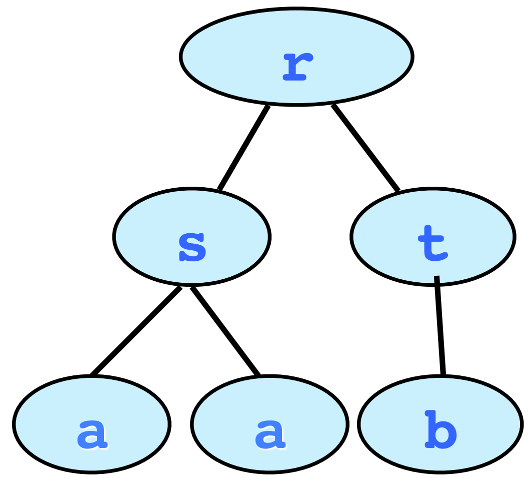
Validation Example

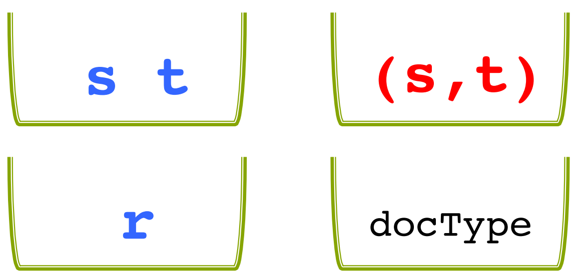





```
<!DOCTYPE r [
<!ELEMENT r (s,t)>
<!ELEMENT s (a*)>
<!ELEMENT t (b?)>
]>
```

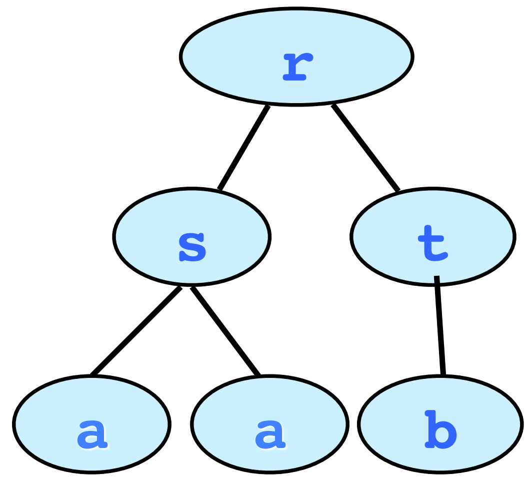
Validation Example

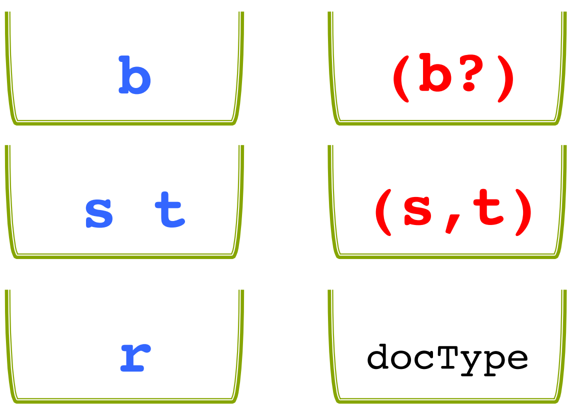





```
<!DOCTYPE r [
<!ELEMENT r (s,t)>
<!ELEMENT s (a*)>
<!ELEMENT t (b?)>
]>
```

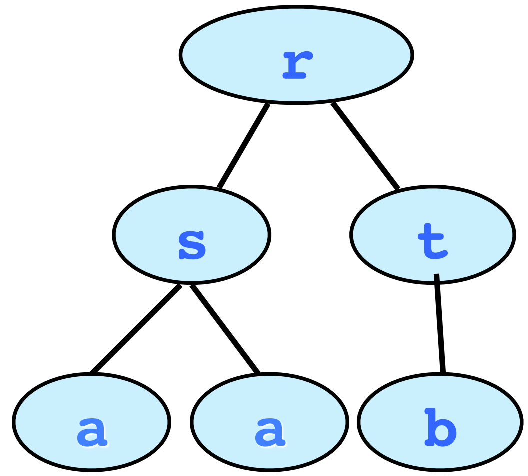
Validation Example



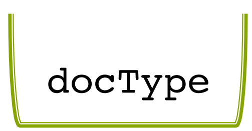



```
<!DOCTYPE r [
<!ELEMENT r (s,t)>
<!ELEMENT s (a*)>
<!ELEMENT t (b?)>
]>
```

Validation Example




```
<!DOCTYPE r [
<!ELEMENT r (s,t)>
<!ELEMENT s (a*)>
<!ELEMENT t (b?)>
]>
```



Research Highlights

Checking Determinism

- Quadratic algorithm [Brueggemann-Klein]
- (best) Linear algorithm [Groz, Staworko, Maneth '11]

Checking Validity

- (best) Sublinear space algorithm [Konrad, Magniez '11]