Finding Optimal Probabilistic Generators for XML Collections

Serge Abiteboul, Yael Amsterdamer, Daniel Deutch, Tova Milo, Pierre Senellart

To appear in BDA 2011

Motivation

Adding probabilities to an XML Schema

- The Web is full of semi-structured documents
- Given a collection of XML documents, we would like to have a schema the documents conform to.
  - E.g., DTD or XSD
  - Restricts the structure, mostly parent-child node relations (using regular expressions)
- The schema may be very general (e.g., xhtml, RSS)
- We want to add probabilities to 'guide' the schema
  - Optimal probabilities – maximize the likelihood of a corpus

Motivation

Implementation Idea: XML Editor Auto-Completion

- Based on previous document versions / corpus of user documents / corpus of example documents –
- Suggest nodes / sub-trees / node values to the user
- For example:
- **Challenges:**
  - Allow editing in every part of the document
  - What kind of completion to suggest?
  - Finding the top-k best completions

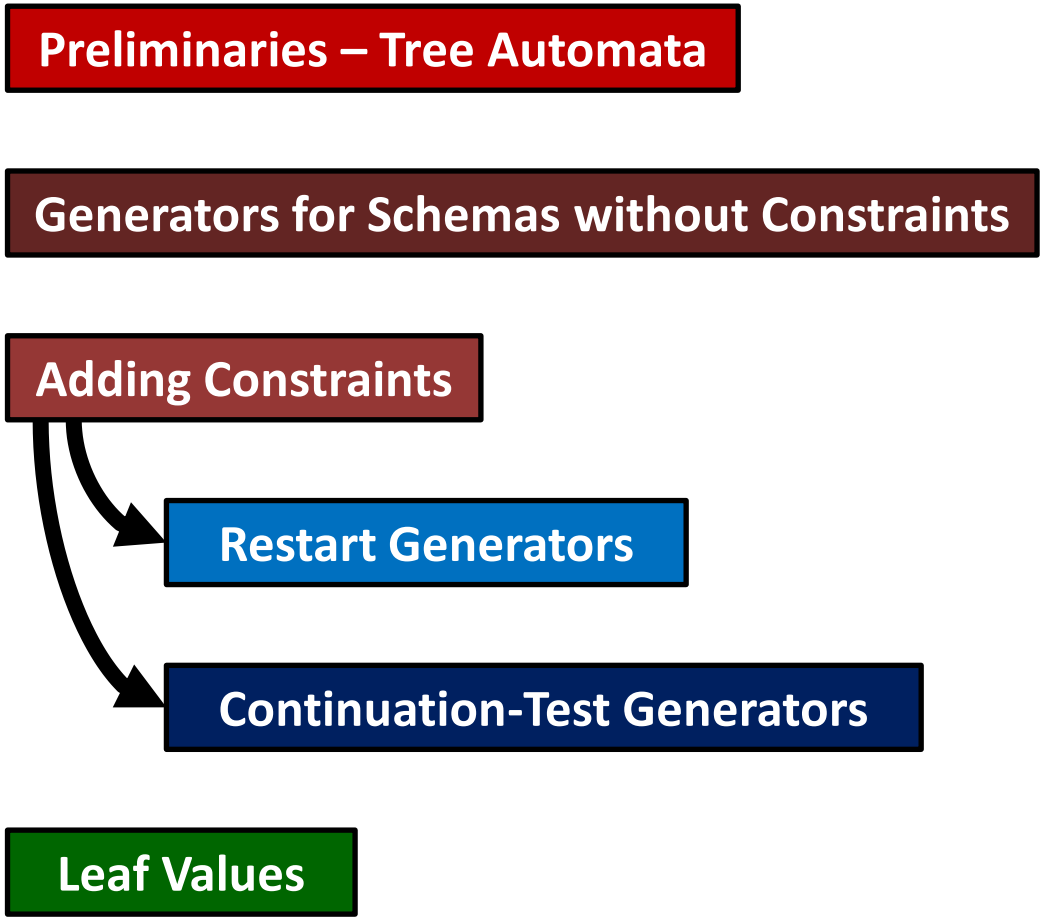
```
<MyPapers>
    <Paper>
        <title>Provenance Minimization</title>
        <author>Yael A.</author>
        <author>Daniel D.</author>
        <author>Tova M.</author>
        <author>Val T.</author>
    </Paper>
    <Paper>
        <title>Provenance for Aggregates</title>
        <author>Yael A.</author>
        <author>Daniel D./author>
        <author>Val T.</author>
    </Paper>
    <Paper>
        <title>      </title>
        <author>      </author>
        <author>      </author>
        <author>      </author>
    </Paper>
</MyPapers>
```

Motivation

Many Other Usages for a Probabilistic Schema

- **Testing** – e.g., generating many XML messages to simulate network load and test system performance.
- **Explaining** – e.g., the probabilistic schema for DBLP shows which types of publications are rarely used, which kinds of attributes are not filled for BibTex, etc.
- **Querying** – e.g., finding the probability that a paper has more than 3 authors.
  – e.g., finding the top-k best completions to a partial document.
- **Schema Evaluation** – how well a given schema describes a given corpus.

Our solution - An Outline

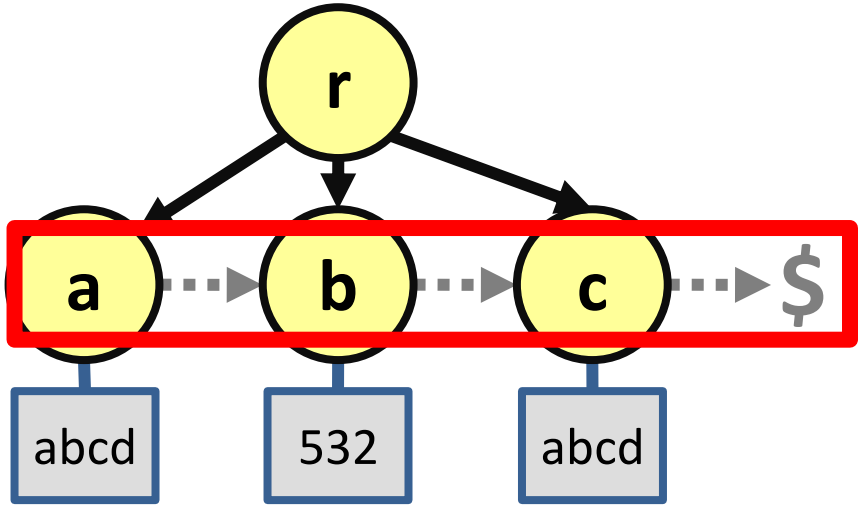


Preliminaries

Using a Tree Automaton for Schema Verification

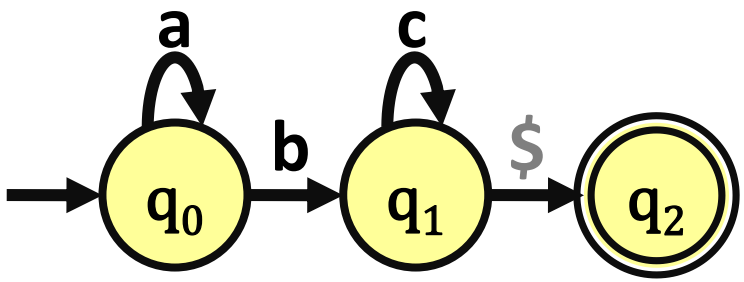
An XML document is modeled as an ordered tree.

Document d_0:



The children of *a*-labeled node are accepted by DFA A_a

Automaton A_r: (L(A_r) = a*bc*$)

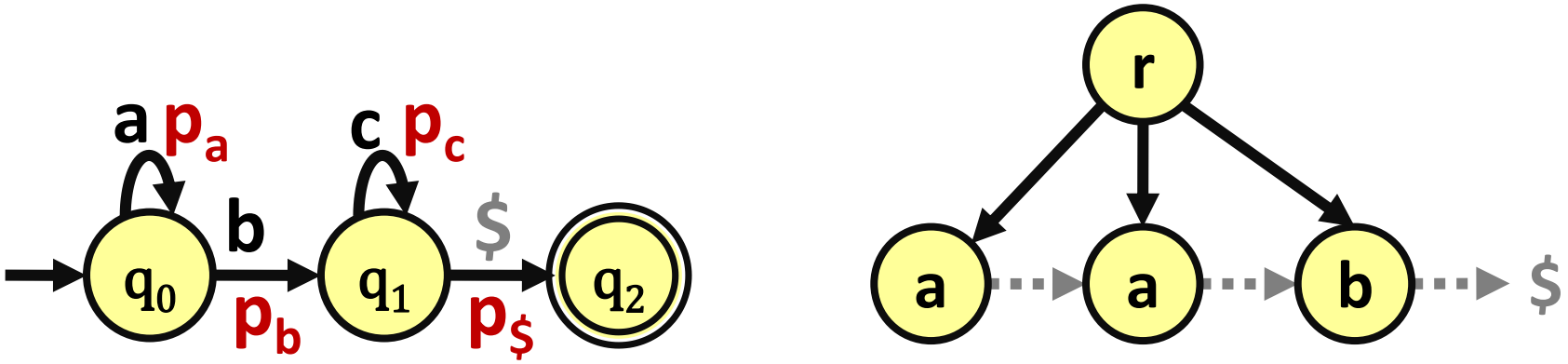


This is done for every inner node in a fixed order (BF-LTR)

Without Constraints

Using the Schema as a (Probabilistic) Generator

- Each transition is assigned a probability



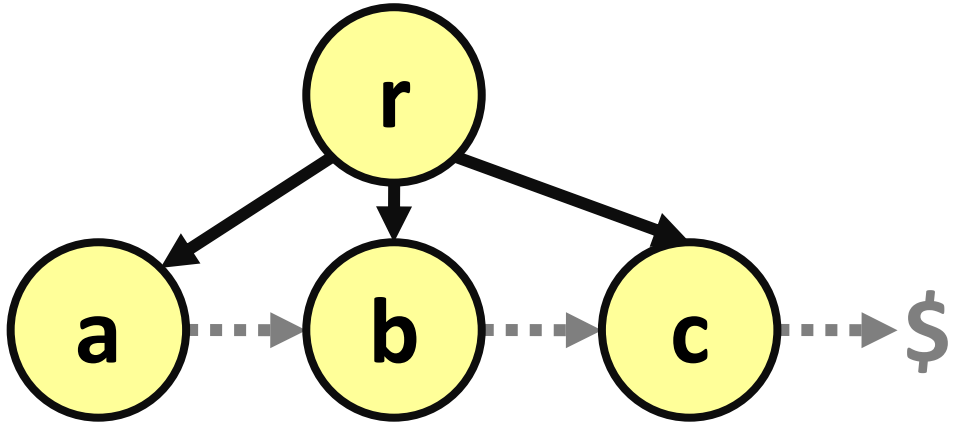
- We assume independent choices, thus the document probability is the product of all t-probs.
- Thus, PR(d)=$p_a \cdot p_a \cdot p_b \cdot p_\$$
- The schema and generator ignore leaf values (for now!)

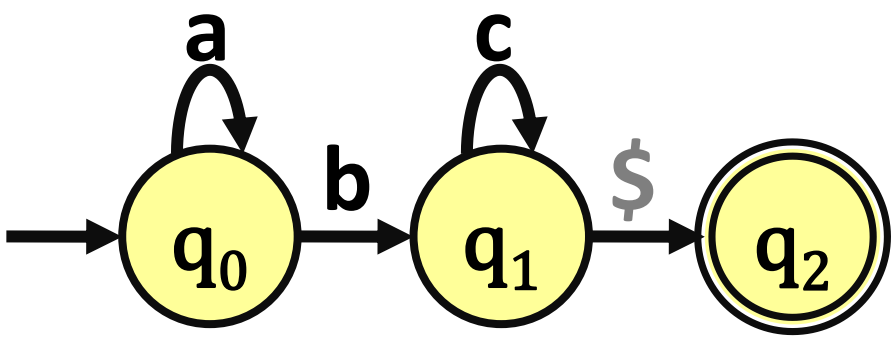
Without Constraints

An Algorithm for Probabilities Learning

The frequency each transition is chosen during the corpus verification process is recorded.

(q_0, a)	1
(q_0, b)	1
(q_1, c)	1
$(q_1, \$)$	1





Without Constraints

An Algorithm for Probabilities Learning (Cont.)

This is repeated for every node in every corpus document.

We set the probability of each transition to be its relative frequency.

(q_0, a)	1/2
(q_0, b)	1/2
(q_1, c)	1/2
$(q_1, \$)$	1/2

These probabilities maximize the likelihood of generating the corpus – optimal generator (similar result in PCFGs)

Without Constraints

Our Results

- Relative frequencies make optimal probabilities.
- Optimal probability learning and document generation can be done efficiently.
- Additional non-trivial result:
  Generation terminates with probability 1.
  - Guaranteed because of the choice of probabilities according to the corpus.

Adding Constraints

Integrity Constraints

- We want to allow the use of the following constraints in the schema:
  - **Key Constraint:** the leaves of *a*-labeled leaves have unique values (unary key)
  - **Inclusion Constraint**: the values of *a*-labeled leaves are contained in those of *b*-labeled leaves
  - **Domain Constraint**: the values of *a*-labeled leaves belong to some (finite or infinite) domain

Restart Generators

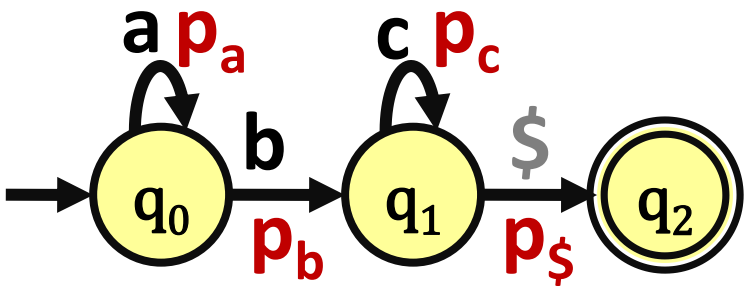
- A naïve idea:
  - Use a probabilistic generator to generate a document
  - Check if it has a value assignment valid w.r.t. the constraints
  - If not, 'restart' and try again until a valid document is generated
- **Good news:** Checking the existence of a valid assignment is in PTIME
- **Bad news:** number of restarts can be unboundedly large in an optimal generator
  - A different quality measure for restart generators?

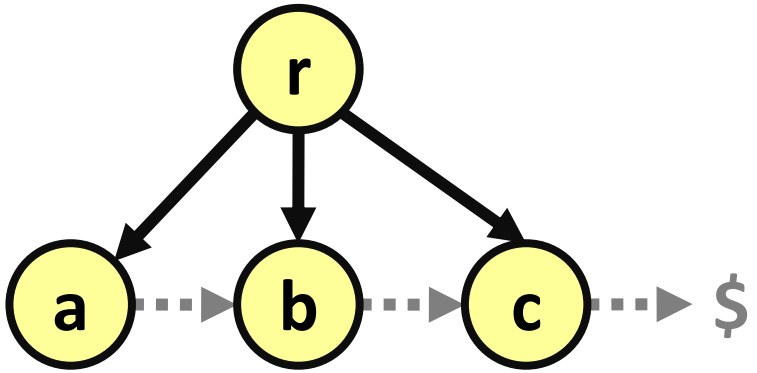
Adding Constraints

Continuation-test Generators

- Never make choices that lead to a 'dead end', thus always generate a valid document.
- We use a binary test to check if a choice has a continuation.
- Add to the schema of d_0 the constraints:
  - *c* is included in *a*
  - *c* is unique
- The generation process:

$$\Pr(d) = p_a \cdot p_b \cdot p_c \cdot \mathbf{1}$$





Adding Constraints

Algorithm for Learning Probabilities with Constraints

- The probabilities are again relative frequencies, but –
  only in cases where there was an alternative choice.

(q_0, a)	1/2
(q_0, b)	1/2
(q_1, c)	1/1
$(q_1, \$)$	0/1

$(q_1, \$)$ was chosen only when (q_1, c) was not available.

- The learned generator will generate as many *c*-s as *a*-s

Adding Constraints

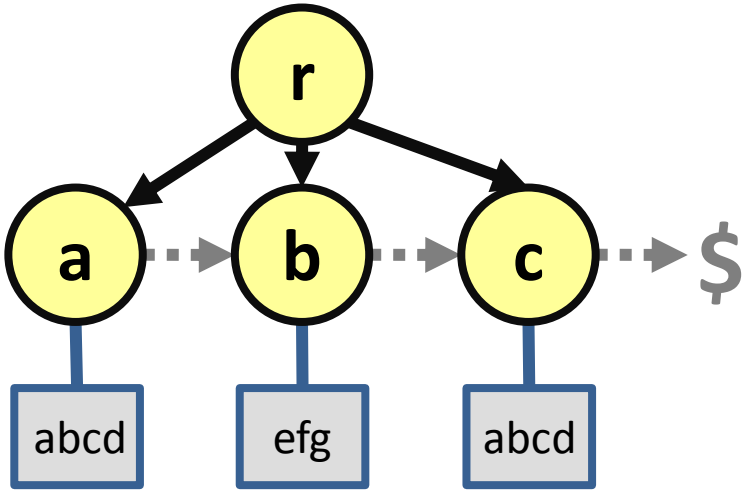
Our Results

- The algorithm learns an optimal continuation-test generator, for automata with binary choices.
  - Extensions to non-binary are discussed in the paper
- **Bad News:** Continuation-test is NP-Complete
  - But only in the size of the schema; it is polynomial in the document size (not so bad?)
  - Based on schema satisfiability test [David et al. 2011]
- **More Bad News:** probability of termination may be arbitrarily small!
  - Even for simple, non-recursive schemas
  - Can be handled by adding a constraint on the document size.
  - Sub-classes of schemas that guarantee termination?

Leaf Values

Suggested Algorithm

- We start with a valid document skeleton

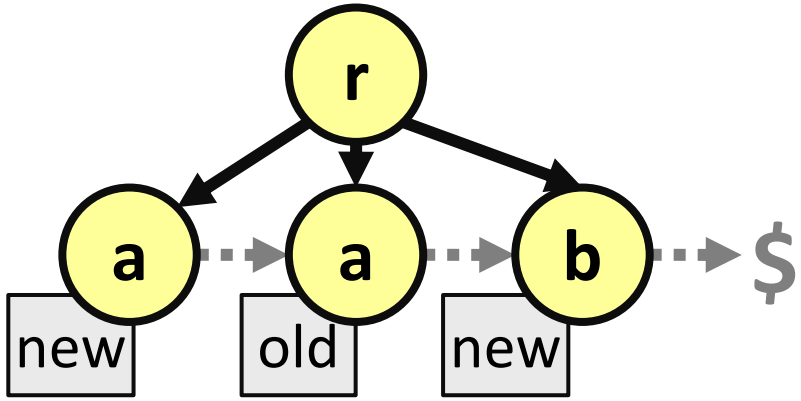


- Order labels by inclusion constraints (e.g., ***c, b, a***)
- Choose a leaf from the 'smallest' (most included) label, and including leaves
- Draw a value (from the domain) according to a given distribution.
- Use PTIME test to verify validity, if not revert the step

Leaf Values

Possible improvement to the basic algorithm

- Annotate the leaves with 'old' or 'new'



- For 'old' *a*-labeled leaves choose values already chosen for some a-labeled leaf
- For 'new' choose a value unused by *a*-labeled leaves yet
- Annotations can be learned from the corpus, and generated:
  - **Offline** – after the document generation, using a PTIME validity test
  - **Online** – during document generation, using a continuation test.
  - Both methods are incomparable in terms of quality

Implementation

Ideas for Experimental Study on Probabilistic generators

- How many schemas in practice may require continuation tests?
- How many have termination probability < 1?
- Continuation test is expensive, but how expensive is it in practice?
- 'Competition' between restart and continuation-test generators
- More?

Thank You!

Questions, Ideas?…