Grammar vs. Rules

Diagnostics in XML Document Validation

Petr Nálevka

University of Economics, Prague
Dept. of Information and Knowledge Engineering

petr@nalevka.com
http://nalevka.com

This presentation is available at:
http://nalevka.com/resources/rules.pdf

Grammar or Rules

- Table of Contents
  - Regular Grammar-based Validation
  - Problems of Diagnostics
  - Rule-based Validation
  - Grammar vs. Rules
  - From Grammar to Rules

Regular Grammar-based Validation

- Mainstream approach
  - Relax NG, XML Schema, DTD
- Based on grammar theory
  - RG defines a language
  - For each RG a FSA can be constructed which accepts all words from this language but rejects all other words
  - A FSA is an algorithmic way how to decide whether a particular word is generated by a particular regular grammar.

Advantages

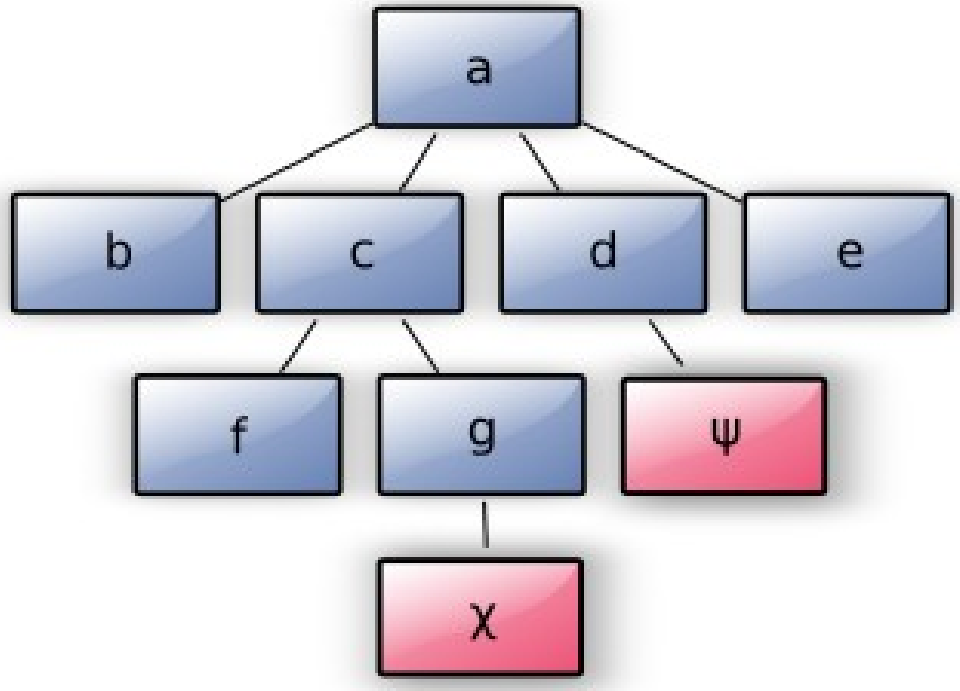
- Computational complexity efficient
  - Regular expressions (not regexps) may be evaluated in linear time
  - For any input word `N` of length `n` and a regular expression `R` of the length `m`; `N` may be matched against `R` in time `O(n+2m)`
- Keeping an expression language Regular Grammar based is convenient:
  - Whatever constructs are used the time to compute is always linear
  - You don't need to think about optimization
  - Important for validation
    - see XML editors

XML Validation and Hedges

- Regular Hedge Grammar
  - Same class of grammar as RG
  - Adjusting RG to be tailored for XML validation
- Hedge over a finite set of symbols `S` and variables `V`:
  - an null hedge `(ε)`,
  - a variable from `V`,
  - `s<h>` where `s` is a symbol from `S` and `h` is again a hedge,
  - or a concatenation of two hedges `i.j`.

Hedge Example

- Hedge notation
  - a<b c<f g<χ>> d<ψ> e>
- The same hedge in a diagram



More Real Life Hedge Example

- Hedge notation
  - html<head<title<'Example'> body<p<'Foo'>>
- The same hedge as XML tree

```
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <p>Foo</p>
  </body>
</html>
```

S = { html, head, title, body, p } and V = { 'Foo', 'Example' }

Validation Using Regular Hedge Grammar

- Analogical to validation using Regular Grammars
  - Regular Hedge Grammar is a mechanism how to generate Hedges
  - Hedge Automata is a mechanism how to algorithmically decide, whether a hedge is generated by a particular grammar
- Validation
  - **Schema** (RHG) defines a set of **XML documents** (Hedges) and **Validator** (HA) decides if an instance belongs to that particular set

Regular Hedge Grammar Definition

- Regular hedge grammar is a formal grammar defined as
  - a finite set of terminal symbols `S`,
  - a finite set of non-terminals `N`,
  - a finite set of variables `V`
  - a set of production rules `P`,
  - `r` which is a regular expression composed of non-terminals
- Production rules in `P` are only of the following form.
  1. `n → v`; where `n` is a non-terminal and `v` is a variable. Applying this production rules means a non-terminal is replaced by a variable.
  2. `n → s<r>`; where `n` is a non-terminal, `s` is a symbol and `r` is a regular expression composed of non-terminal symbols. If this production rule gets applied a non-terminal is replaced by a terminal symbol which contains a sequence of non-terminals matching the regular expression `r`.

Example Regular Hedge Grammar

Defining a grammar using DTD

```
<!ELEMENT  html (head, body)>
<!ELEMENT  body (p|img)*>
<!ELEMENT  head (title)>
<!ELEMENT  title (#PCDATA)>
<!ELEMENT  p (#PCDATA)>
<!ELEMENT  img EMPTY>
```

The same definition in the RHG notation.

```
S = {html, head, body, title, p, img }
V = { "#PCDATA" }
N = { N(html), N(head), N(body),
N(title), N(p), N(img), N(pcdata) }
P = { N(html)  →  html<N(head), N(body)>
      N(head)  →  head<N(title)>
      N(body)  →  body<(N(p) | N(img))*>
      N(title) →  title<N(pcdata)>
      N(p)     →  p<N(pcdata)>
      N(img)   →  img<ε>
      N(pcdata)→  #PCDATA }
```

Deterministic Hedge Automaton

- Validation of XML documents
  - a task to construct a Deterministic Hedge Automaton (DHA) and find out whether it accepts the input
- DHA is a Finite State Automaton (FSA)
    - finite set of symbols, states and transitional function
- DHA have just few slight modifications over FSA:
  - 2 transition functions – for symbols (Fs), for variables (Fv)
  - the result of Fs depends not only on the current state and the input symbol, but it depends on a set of states which are the target states for the child symbols or values in the hedge.

Example Validation

Schema in DTD

```
<!ELEMENT html (head,
body)>
<!ELEMENT body (p|img)*>
<!ELEMENT head (title)>
<!ELEMENT title
(#PCDATA)>
<!ELEMENT p (#PCDATA)>
<!ELEMENT img EMPTY>
```

```
<html>
  <head>
<title>Example</title>
  </head>
  <body><p>Foo</p>
  </body>
</html>
```


Problems of RG-based Diagnostics

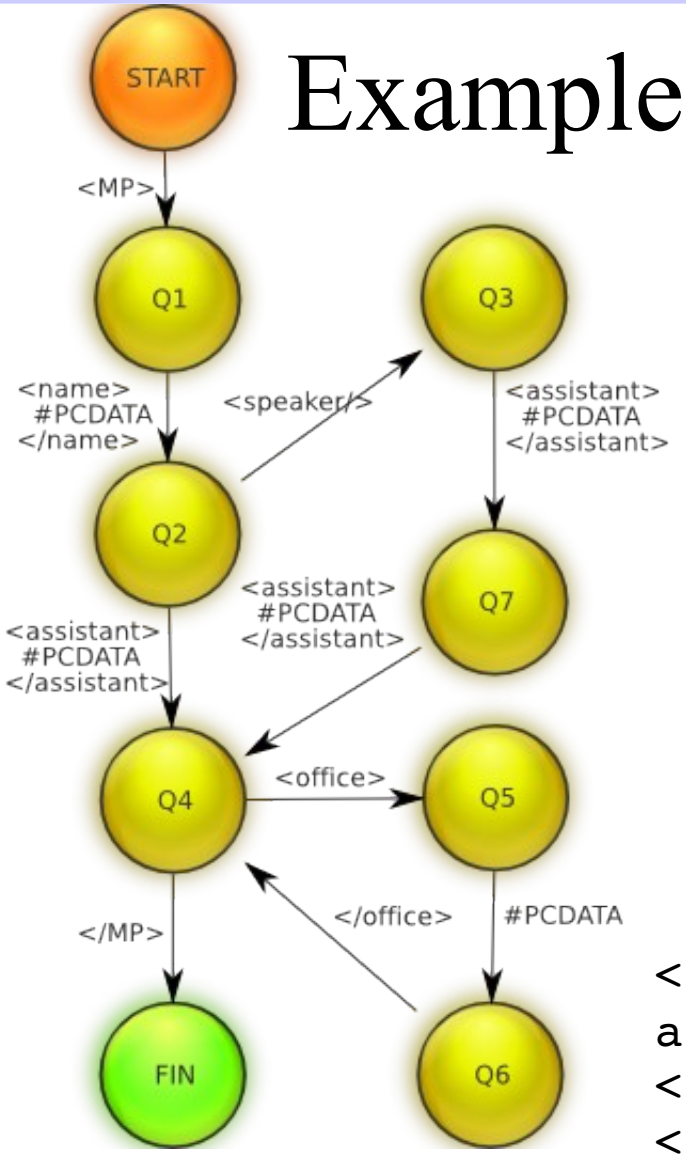
- Grammars are good at telling something is wrong
- But they bad at telling

  1) where exactly the problem occurred

  2) explaining the issue in a domain specific or human understandable form

- Diagnostics shall guide the author to resolve the issue
- Regular grammar-based validation is low-level
  - it only knows about the trees, but there is no knowledge about the modelled domain

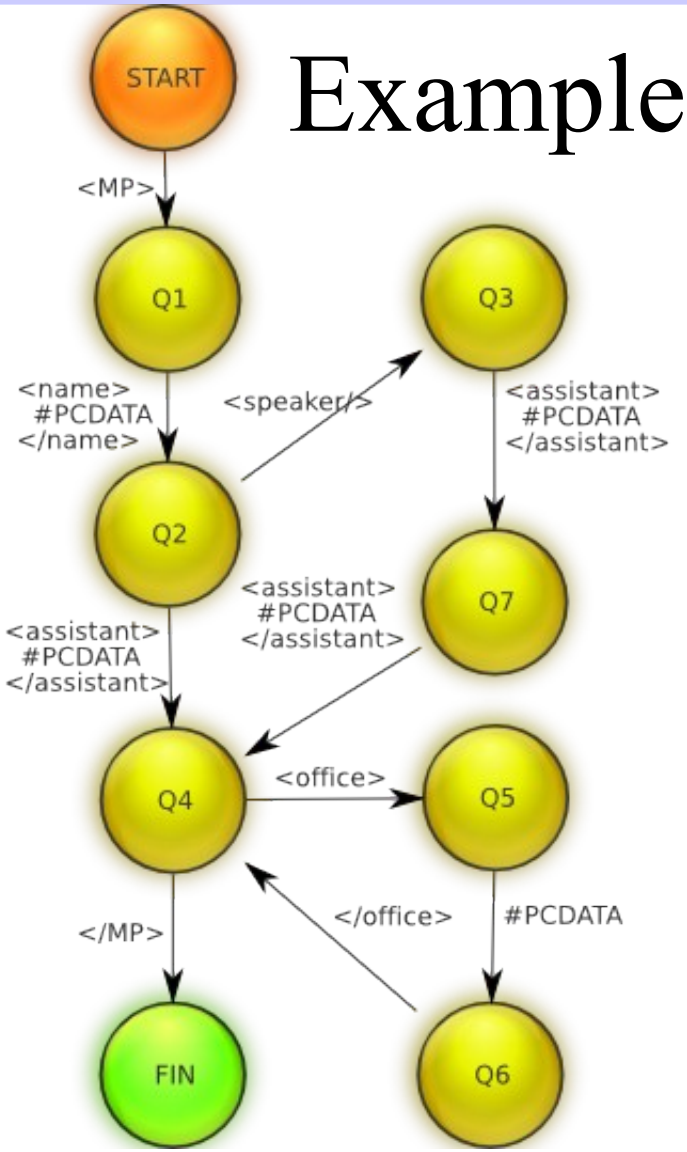
Example: Problem locating errors



- Simple MP evidence
  - MPs are identified by name which is required
  - each MP must have one assistant. Only the Speaker of the Parliament has two assistants.
  - The door number of the MPs office is optional.

```
<!ELEMENT  MP (name, (speaker, assistant)?,
assistant, office?)
<!ELEMENT  office     (#PCDATA)>
<!ELEMENT  assistant (#PCDATA)>
<!ELEMENT  speaker  EMPTY>
```

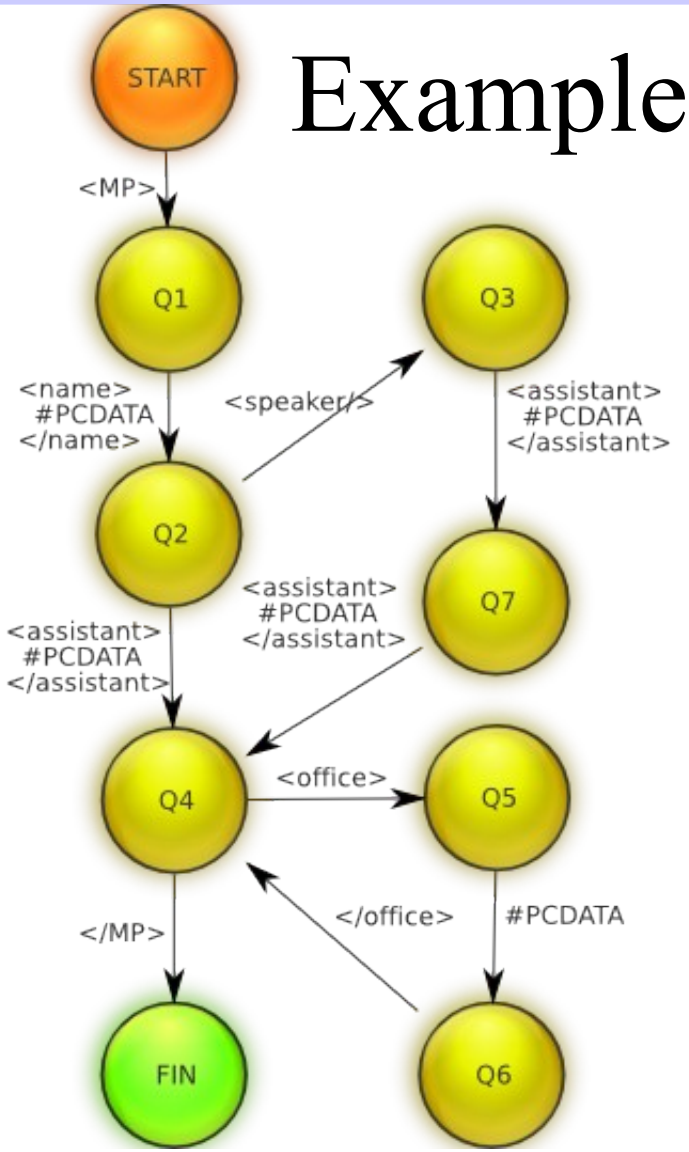
Example: Problem locating errors




```
<MP>
  <name>Miloslav Vlcek</name>
  <speaker/>
  <assistant>Petr Mazalek</assistant>
  <assistant>Veronika
Soumanova</assistant>
  <office>56</office>
</MP>

START → Q1 → Q2 → Q3 → Q7 → Q4 → Q4 → Q5
→ Q6 → Q4 → FIN  (VALID)
```

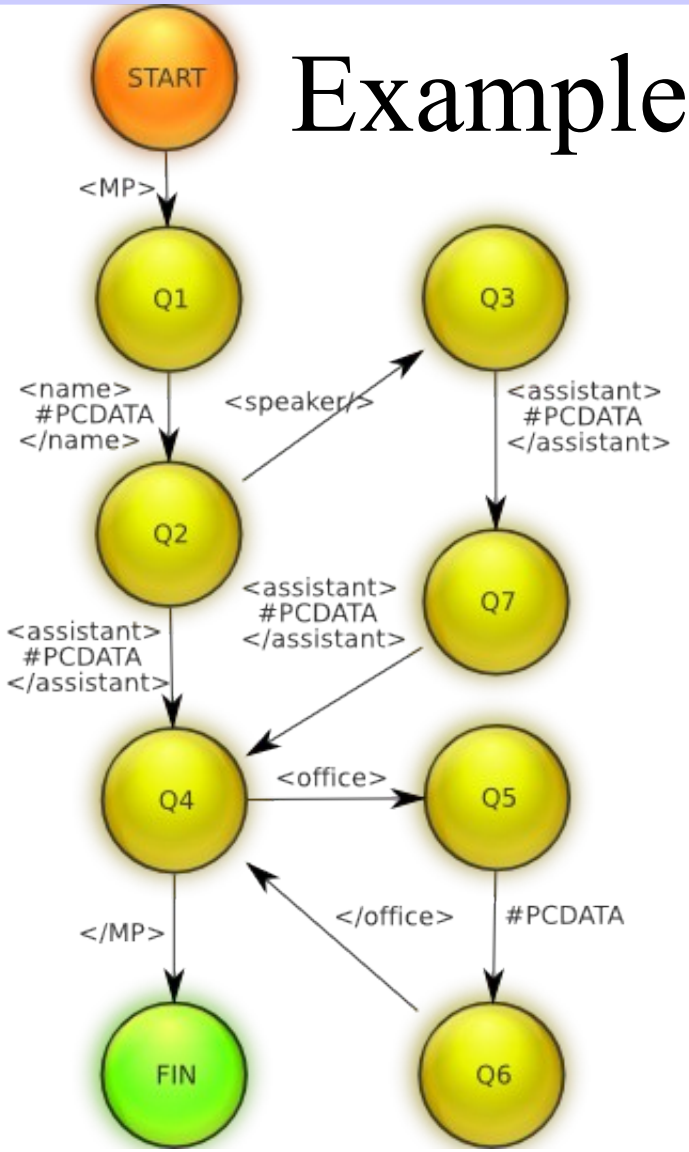
Example: Problem locating errors




```
<MP>
  <name>Petr Bradsky</name>
  <assistant>Jaroslava
Pokorna</assistant>
</MP>

START → Q1 → Q2 → Q4 → FIN  (VALID)
```

Example: Problem locating errors

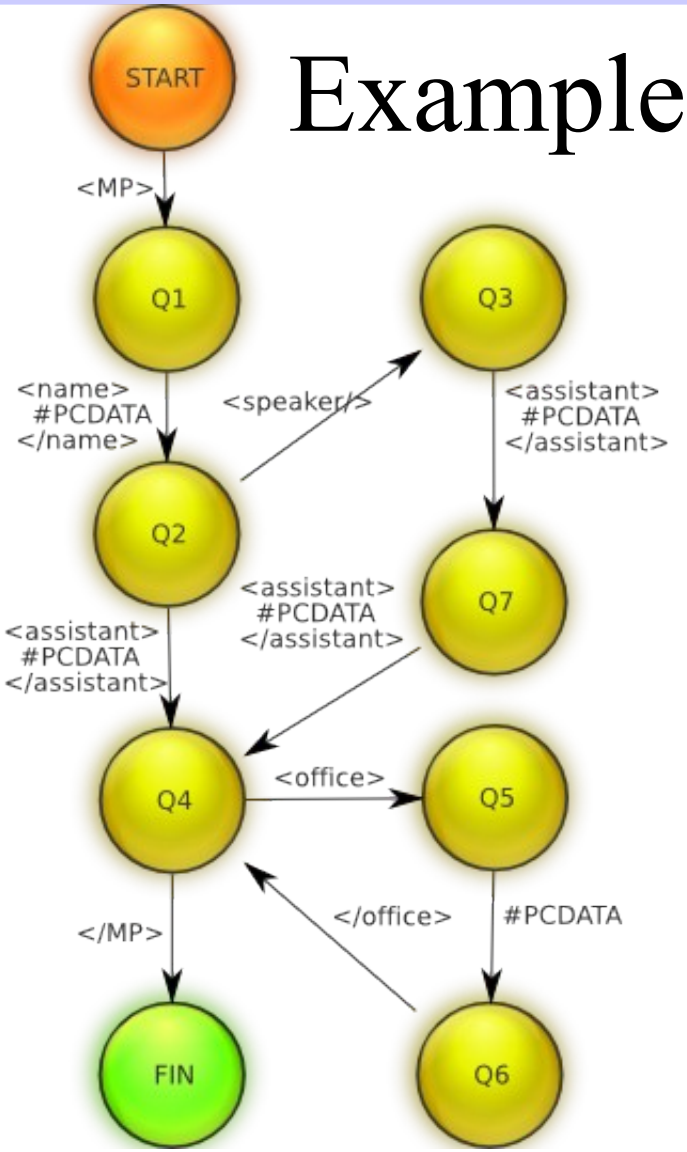



```
<MP>
  <name>Miloslav Vlcek</name>
  <speaker/>
  <assistant>Petr Mazalek</assistant>
</MP>

START → Q1 → Q2 → Q3 → Q7  (INVALID)

ERROR: Expected <assistant> but was
</MP>
```

Example: Problem locating errors




```
<MP>
  <name>Miloslav Vlcek</name>
  <assistant>Petr Mazalek</assistant>
  <assistant>Veronika Soumanova</assistant>
</MP>

START → Q1 → Q2 → Q4   (INVALID)
```

ERROR: Expected </MP> or <office> but was <assistant>

In this case somebody did obviously forgotten to mark the MP as the Speaker of the Parliament. RG-based diagnostics does not help to resolve such issue.

Problem of explaining errors

- Insufficient diagnostics (completely useless, misleading and wrongly positioned)
  - "Expected `</MP>` or `<office>` but was `<assistant>`"
- Ideal diagnostics
  - "Missing `<speaker/>` element. '*Miloslav Vlcek*' has 2 assistants defined but he is not marked as the Speaker of the Parliament using the `<speaker/>` tag. Only the Speaker is entitled to have two assistants, regular members may have only one. Either add the `<speaker>` element as the first child of the `<MP>` element or remove one of the `<assistant>` elements."

Using rules

- Alternative approach to Grammars
- Becoming more and more popular
- Principle
  - Matching input documents against a set of schema author defined patterns (rules)
  - The author has full control over the patterns and over diagnostics attached to them

Schematron

- ISO standard
- Only 6 main elements, easy to learn (but needs to know the underlying assertion language)
  - schema, pattern, rule, assert/report, value-of
- Different languages to express rules
  - XPath (perfect for expressing contexts in XML documents)
  - JavaScript
  - any other language able to address nodes in XML

Schematron Rule Example

```
<sch:pattern name="MP Evidence">
    <sch:rule context="MP">
        <sch:assert report="count(assistant) > 1 and not(speaker)">
Missing speaker element.
'<sch:value-of select="name"/>' has <sch:value-of
select="count(assistant)"/> assistant(s) defined but he is not marked
as the Speaker of the Parliament using the speaker tag. Only the
Speaker is entitled to have two assistants, regular members may have
only one. Either add the speaker element as the first child of the MP
element or remove one of the assistants.
        </sch:assert>
    </sch:rule>
</sch:pattern>
```


Problems

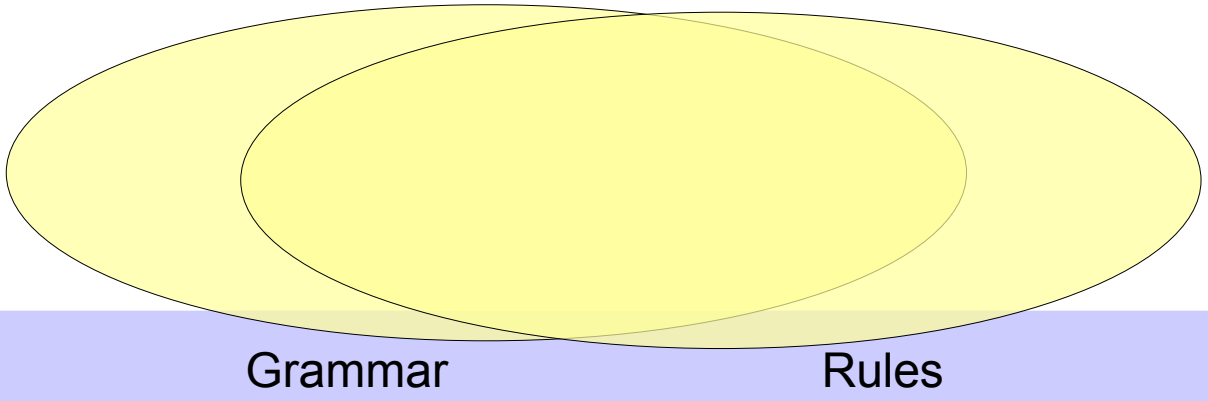
- Low-level control over validation process
  - Authors need to express more about how to validate the documents
    - Not defined = Allowed (RG – Not defined = Forbidden)
    - Consistency!
  - Relationships which are simple to model in RG-based schemas are more difficult to express
    - several rules are needed for something which is implicit in RG-based schemas
- Execution time depends on rules used (RG – linear time)
  - Schema authors need to optimize
  - Huge area for future performance optimisation
- Difficult to be used for XML editors code completion

Schematron Performance Optimisation

- Optimising of assertions
- Reuse rule contexts for several rules
  - Rules are evaluated relatively to their pattern context
- Organize patterns into Phases
  - Executed in certain order, only if the document passes a phase the next phase is proceeded

Grammar vs. Rules: Expressiveness

- Theory
  - Set of XML documents able to be described by rules or grammars
    - Significant intersection but none is subset of the other
- Praxis
  - Many useful restrictions which may be modelled using rules but not using any RG-based language
  - Rules operate on multiple contexts across the document



Rules inexpressible by grammar, Example

- Web Content Accessibility Guidelines – how to write accessible Web pages
  - Many important rules, but unable to automatically validate them using grammars
  - No automatic validation = No compliance!
  - The Relaxed project (and others) used Schematron to express what is inexpressible using grammars

```
<sch:rule context="html:abbr">
  <sch:report test="not(@title) and not(preceding::html:abbr[. =
string(current())][@title])">
   WCAG 1.0 Checkpoint 4.2 (Priority 3) First occurrence of
abbreviation in a document needs to have an title defined.
  </sch:report>
</sch:rule>
```

Combining Grammar and Rules

- To gain advantages of both approaches
- Expressing different restriction in the most suitable approach and validate against both
  - Schema languages may be combined
    - Relax NG + Schematron
    - XML Schema + Schematron
- The Relaxed project
  - Formalized verbal restrictions in HTML specs
    - allowed automated validation of many additional rules

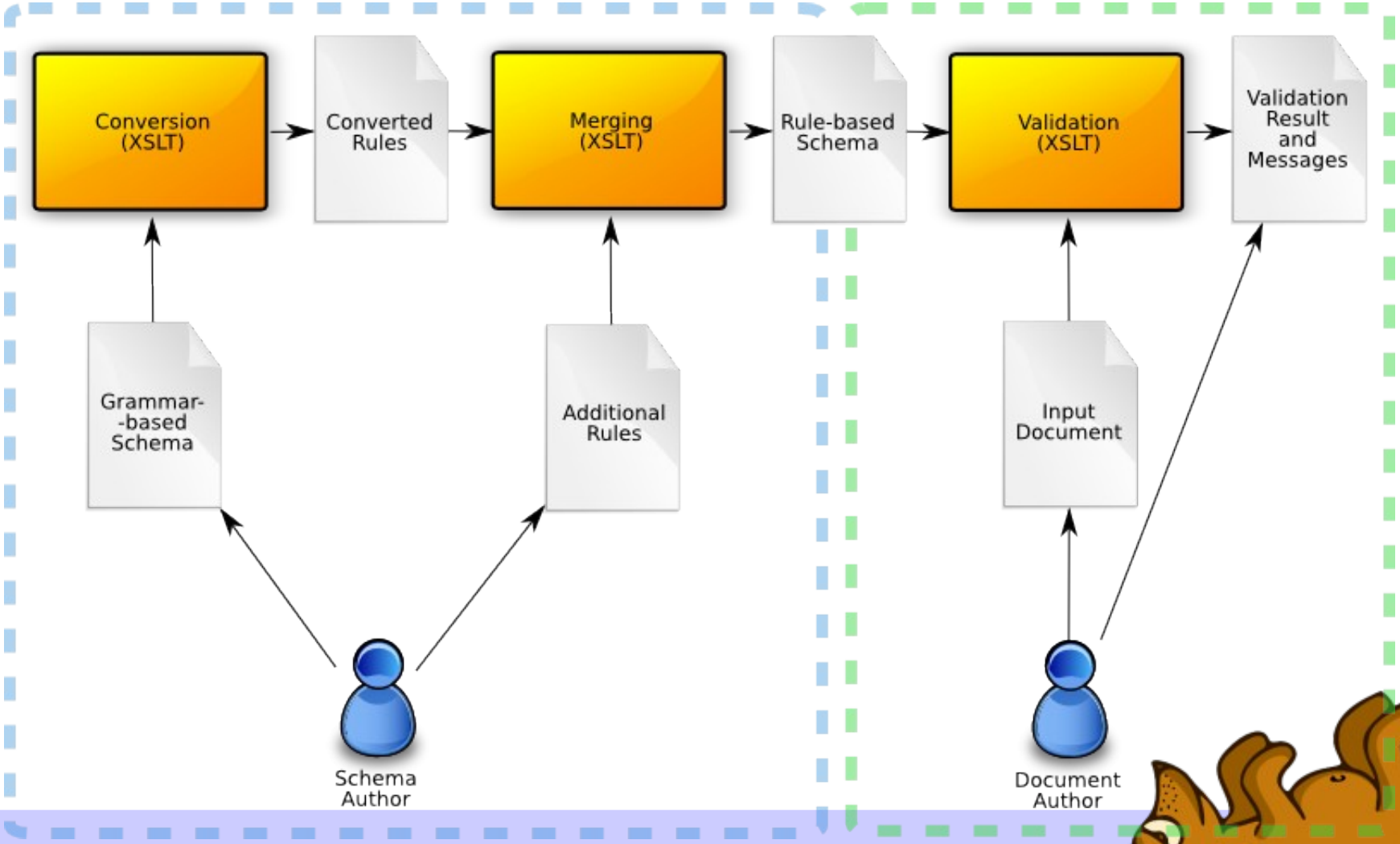
Converting Grammar to Rules

- Another approach
  - Converting Grammar to Rules
  - Schema languages may be converted
    - Relax NG → Schematron
    - XML Schema → Schematron
  - May enhance diagnostics by keeping simplicity of expression (better maintenance)

Converting Grammar to Rules

- Advantages
  - Enhanced diagnostics
    - Rules may expresses additional restrictions
    - Rules may operate on multiple contexts
  - Automatically creates a rule-based foundation which can be later easily enhanced by domain specific diagnostics
  - Platform independent validation (XSLT processor)
  - Simple integration with other specific rules
    - merging generated rules
    - only one validation process

Conversion Flow



How to Convert Grammar to Rules

- Problem
  - Unequally expressive – conversion can't be exhaustive, but all *reasonable* constructs in grammar-based schemas may be expressed using rules
- No simple algorithm to do so
- The simple approach – using regexp support in XPath 2.0
  - Validation is only a XSLT transformation
  - But no enhanced diagnostics

How to Convert Grammar to Rules

- More sophisticated approach
  - Handle constructs one by one specifically
  - Element and attribute names
    - Grammar-based schema enumerates all different elements and attributes
    - Rule: for every element and attribute from the input document check that their names belong into the set of allowed names
      - otherwise unknown element is thrown
  - Content models of elements
    - Grammar-based schema defines content models
    - Rule: for each element context check if only allowed children are present
  - And so on...

Real Life Use-case

- United Kingdom Tax Office
  - Employee taxes communicated with Tax Office using XML
  - Forms are mapped to XML, validated and sent to the server
  - Validation errors need to be mapper back to forms
  - Users need to be explained what they did wrong to be able to correct the mistake
- Schematron is the right alternative
  - but there are already huge XML Schemas
  - they need conversion and merging

Problems

- Merging of rules
  - Duplicate rule detection
  - Consistency
- Automatically generated rules have better diagnostics but no additional knowledge about the domain!
  - Domain specific knowledge still needs to be attached manually
    - using Schematron it is a low-level task

Modelling the Domain

- Another approach to attach domain specific diagnostics
  - Directly model the validated domain directly in a suitable language
    - Description logic is a candidate (RDF/OWL)
  - Define mapping from XML to RDF
  - Convert XML instances to RDF and check consistency using an Ontology

Thank You for Attention!

For more info visit:
http://nalevka.com
Ask questions at:
petr@nalevka.com