XML Processing and Term Rewriting

Stratego/XT

Martin Bravenboer

martin@cs.uu.nl

Institute of Information and Computing Sciences, University Utrecht, The Netherlands
monday

- relate xml and aterms
- impression of xml processing methods and languages
  - mainstream oriented
    - text/api based
    - xpath, xslt, xquery
  - research oriented
    - xen, xduce, xtatic, cduce
    - generic haskell

wednesday

- how to connect xml and aterm tools
- how to apply stratego for xml processing
why discuss xml processing

- *stratego*: program transformation

- transform *aterm* representations of the source code of programs

- why not *structured data* in general?

- that’s the topic of my master’s thesis:

  connecting *xml processing* and *term rewriting* using tree grammars
xml and aterm concepts
xml and aterm: concepts

- **xml web-services**
  - independent software tools
  - working together by exchanging xml

- **stratego/xt**
  - component-based transformation systems
  - exchanging program representations
  - in the aterm format

exchange of structured, tree-like data between

*software components*
xml and aterm: contribution

enables ‘generic’:

- **tools and libraries**
  parsers, pretty-printers, well-formedness checkers, validators, editors, browsers, . . .

- **languages**
  schema, query, transformation, style, dedicated, general purpose, . . .

xml syntax for tree-like data is

- platform,
- language,
- culture,
- and application independent.
xml and aterm: similarities and differences

similarities

• xml element $\sim$ aterm application
• xml character data $\sim$ aterm string
• xml attribute $\sim$ aterm annotation

differences

• aterm has:
  ◦ explicit structure
  ◦ primitive data types
  ◦ structured annotations

• formalisms:
  ◦ aterm format $\sim$ tree languages
  ◦ xml $\sim$ hedge languages
xml and aterm: concepts

- an xml document is not a tree
- an aterm is not a tree

⇒ generic syntax for tree-like data
xml processing in practice
text based xml processing

- xml in string literals
- xml in templates with embedded variables

```csharp
sw.WriteLine("<?xml version="1.0" encoding="Windows-1252"?>");
sw.WriteLine("<configuration>"nk);
sw.WriteLine("\t<appSettings>"nk);
sw.WriteLine("\t\t<add key="Main.ConnectionString" value=""nk
            + m_sConnectionString + "\" />"nk);
sw.WriteLine("\t\t</add>"nk);
sw.WriteLine("\t</appSettings>"nk);
sw.WriteLine("</configuration>"nk);
sw.Close();
```

(fragment of LLBLGen)
text based xml processing

- xml in string literals
- xml in templates with embedded variables

+
- xml in ‘concrete syntax’
- easy to start with

- no syntax checking: well formedness
- no transformation: requires interpretation
Element report = new Element("exception-report");

Element topic = new Element("topic");
topic.setText(_exception.getTopic());
report.addContent(topic);

Element userinfo = new Element("user-info");
userinfo.setText(_message.getBackgroundValue().getContent());
report.addContent(userinfo);

createExceptionElement(report, _exception.getException());

StringWriter writer = new StringWriter();
XMLOutputter outputter = new XMLOutputter("\t", true);
outputter.output(report, writer);
api based

produce or consume xml with a dom, sax or pull api

```java
void serialize(Date time, ContentHandler h) {
    _calendar.setTime(time);
    intElement(h, "day-of-month", _calendar.get(Calendar.DAY_OF_MONTH));
    intElement(h, "month", _calendar.get(Calendar.MONTH) + 1);
    intElement(h, "year", _calendar.get(Calendar.YEAR));
}

void intElement(ContentHandler handler, String elem, int val) {
    textElement(handler, elem, String.valueOf(val));
}

void textElement(ContentHandler handler, String elem, String text) {
    startElement(handler, elem);
    characters(handler, text);
    endElement(handler, elem);
}
```
api based

produce or consume xml with a dom, sax or pull api

+

• guarantees for well-formedness (not always)
• transformation possible if api allows

-

• verbose: does not scale to large fragments
• no xml specific language facilities
how to improve?

- **embed the xml syntax** in general purpose language
  - syntax for api calls or data
  - not (yet) applied in practice
  - tiger, java, c# with xml syntax

- **xml data binding**
  - natural representation of xml data
  - jaxb, castor, dtd2haskell
  - applied in practice

- **dedicated xml language**
  - built-in support for xml
  - xpath, xslt, xquery, xslt, xduce, cduce
  - applied in practice
xpath: successful mini language

- select nodes in an xml document
- no variable binding

+ easy to use syntax, based on a set of axes
- can be reused in many languages

- verbose pattern matching
  
  BinOp[PLUS and *[position() = 3 and name(.) = 'BinOp']/PLUS ]

- lack of variable binding sometimes annoying
**xslt: xml transformation language**

- *templates rewrite* a node that matches an xpath
- *recursively apply templates* to nodes selected by and xpath
- stateless ‘functional’ language

```xml
dxslt:template match="category">
  <li>
    <h2><xsl:value-of select="@name"/></h2>
    <ul><xsl:apply-templates/></ul>
  </li>
</xslt:template>

<dxslt:template match="link">
  <li>
    <a href="{@url}" alt="{@name}"
      <xsl:value-of select="@name"/>
    </a>
  </li>
</dxslt:template>
```
xslt: xml transformation language

+ 
  • easy to use if you know xpath
  • most widely applied functional language!

- 
  • limited set of functions (use EXSLT)
  • difficult to create abstraction
  • transformation of ‘results’ not allowed (EXSLT, XSLT 2.0)
  • abused to generate ‘plain text’

• compared to stratego
  ◦ no separation of rules and strategies
  ◦ no first class pattern matching
  ◦ no support for implementing full xml applications
**XQuery**: XML query language

**FLOWR expressions**

- **for** - select nodes using XPath
- **let** - bind nodes to variables
- **where** - apply conditions
- **order by** - sort the results
- **return** - construct new nodes

- very easy to learn
- more declarative, not operational
  - like Java, XSLT, Stratego, Haskell
- less convenient for transformations (duh)
xml processing in research
research xml processing languages

focus of research:

- *type systems*
  - xduce, xtatic, xquery, xen, cduce

- *performance*
  - pattern matching compilation

limited research:

- *generic programming*
  - focus on getting a basic type system right

- *traversals*
  - xml data is not typical data in functional languages

- *composition and interaction*
xen

- Microsoft Webdata, Research / Cambridge
- extension of C#: more general data model
  - streams: $T?$, $T!$, $T$, $T^*$, $T^+$
  - tuples: sequence
  - unions: choice
- xml is object literal syntax for this extended data model
- more operations: filter, apply to all

read the articles for more info:
- unifying tables, objects and documents
- programming with circles, triangles and rectangles
generic haskell

- generic extension of haskell
- type-safe access to the structure of data
- xml programming is xml data binding to haskell data types
- subject of *generic programming* course

+ 
  - amazing abstraction and reuse

- 
  - data structures must be known at compile time
    ⇒ generic programming, but not ‘any’ data
  - no dedicated features for xml like data

more info: [http://www.generic-haskell.org](http://www.generic-haskell.org)
cduce

- designed at two universities in France
- typed, xml-oriented, functional language
- goal: develop more complex applications completely in cduce

main areas of interest:
- type system: structural typing
  - type: set of values
  - $t_1$ subtype $t_2$ if $e_1$ subset of $e_2$
- type-based pattern matching
- generic programming
cduce: type system

- universal type: *Any*
- native scalar types: *Int* (infinite), *Char* (Unicode), *Atom*
- constructed types
  - product type: \((t_1, t_2)\)
  - open record type: \(\{a_1 = t_1, \ldots, a_n = t_n\}\)
  - closed record type: \(\{|a_1 = t_1, \ldots, a_n = t_n|\}\)
  - xml type: \(< t_1 t_2 > t_3\)
  - functional type: \(t_1 \rightarrow t_2\)
- boolean operations on types
  - union: \(t_1 | t_2\)
  - intersection: \(t_1 \& t_2\)
  - difference: \(t_1 \setminus t_2\)
- singleton types: a scalar or constructed value is a type
cduce: type system

- encoded types
  - sequence: \([v_1, v_2, \ldots, v_n]\) is \((v_1, (v_2, (\ldots, (v_n, \text{‘nil’})))))\)
  - strings are sequences of chars

- overloaded functions

\[
\begin{align*}
\text{let } & \text{fun } f(t_1 \rightarrow s_1; \ldots; t_n \rightarrow s_n) \\
& \mid p_1 \rightarrow e_1 \\
& \mid \ldots \\
& \mid p_m \rightarrow e_m
\end{align*}
\]
cduce: patterns

- **pattern-matching expression**

  \[
  \text{match } e \text{ with } \\
  \begin{align*}
  p_1 & \rightarrow e_1 \\
  \ldots & \\
  p_n & \rightarrow e_n
  \end{align*}
  \]

- \text{let } p = e_1 \text{ in } e_2 \text{ is defined as } \text{match } e_1 \text{ with } p \rightarrow e_2

- \text{\_ refers to } \text{Any}

- matching must be exhaustive

- exceptions can be used to make ’dynamic type errors’ \textit{explicit} in the code

  (compare to Maybe and NullableException)
cduce: pattern variables

- capture variables: bind values
- multiple occurrences of a variable: multiple values
- $x \ & t_1$ adds type constraint $t_1$ to capture variable $x$.
- $p_1 | p_2$ matches $p_1$ or $p_2$.
- $x := c$ sets a default value for a capture variable.
- $x :: R$ sequence capture variable for regular expression $R$.
- recursive patterns
  - $P$ where $P = (\_, P) | (x \ & \text{Int}, \_)$
  - $P$ where $P = (x \ & \text{Int}, \_ | (\_, P)$
  - $P$ where $P = (x \ & \text{Int}, P) | (\_, P) | (x := 'nil)$
cduce: exceptions

• raise an exception:
  
  \( \text{raise } e \)

• catch an exception:

  \[ \text{try } e \text{ with } \]
  
  \[ \begin{align*}
  & p_1 \rightarrow e_1 \\
  & \ldots \\
  & p_n \rightarrow e_n
  \end{align*} \]

application: loading an xml file

\[
\text{let } e : \text{Exp} = \\
\text{match load_xml "program.xml" with} \\
\quad x \text{ & Exp} \rightarrow x \\
\quad _ \rightarrow \text{raise "program.xml is not of type Exp"}
\]
cduce: map and (x)transform

currently cduce does not support parametric polymorphism

or in other words: you cannot define a foldr, map, etc.
(that preserves the type of the expression)

- \textit{map} \( e \) with
  \[
  p_1 \rightarrow e_1 \mid \ldots \mid p_n \rightarrow e_n
  \]

- \textit{transform} \( e \) with
  \[
  p_1 \rightarrow e_1 \mid \ldots \mid p_n \rightarrow e_n
  \]
  \( \sim \text{filter(...); concat} \)

- \textit{xtransform} \( e \) with
  \[
  p_1 \rightarrow e_1 \mid \ldots \mid p_n \rightarrow e_n
  \]
  \( \sim \text{alltd(...)} \)
cduce: generic programming

• cduce provides access to the structure of an XML element

• function can accept *Any* type and constructed types containing *Any*

• implement the operation for the possible constructed and scalar types.

more info:  http://www.cduce.org
xml processing in stratego
xml, terms and stratego: why?

exchange

→ from xml systems invoke term tools
← invoke xml tools from term systems

implement

more complex xml transformations using
  ◦ strategic rewriting
  ◦ dynamic rules
  ◦ general traversals
  ◦ concrete object syntax
what representation to transform?

- every application has its own essence of xml

- different needs, different representations
  - xml-doc
  - xml-info
  - implicitly structured aterm
  - explicitly structured aterm

- issues
  - namespace notation
  - character data constructs
  - empty elements
  - comments, processing instructions
  - ‘meta’ and default attributes
levels of representation

- *xml-doc*
  actual syntax of an xml document

- *xml-info*
  relevant informatie of an xml document

- *implicitly structured aterm*
  drop xml, no explicit structure

- *explicitly structured aterm*
  natural data of an xml document
  ⇒ what is natural?
xml-doc in Stratego

- xml is a concrete syntax for xml-doc
- embed the xml syntax in stratego

_meta programming with concrete object syntax_

```stratego
module tom
imports xml-doc options
strategies

main =
  output-wrap(title)

title =
  !%><title>Tom Bombadil</title>%%

Meta([Syntax("Stratego-xml")])
```
xml-info in Stratego

- information oriented transformations
- same syntax of xmlin stratego
- process xml-doc fragments to xml-info
- allows declaration of module namespaces

```
parse-stratego-xml-info -i select-bars.str
   | process-stratego-xml-doc
   | process-stratego-xml-info
   | meta-explode
   | stratego-desugar -o select-bars.rtree
```
does this scale to real programs?

- **XDoc** – Rob Vermaas
  - exendible documentation generator
  - instantions for stratego, java, sdf

- **XWeb** – Niels Janssen
  - transformation tool demo
  - uses xml-info in Stratego

- **Relation algebra to MathML** – Martin Bravenboer

- **Misc. small tools** – Martin Bravenboer
  - samples package
  - daily build system overview
  - xml-tools themselves!
structured aterm in Stratego

- make structure explicit

- should be nothing special to tell about

- little experience; applied to
  - java
  - lecture results
  - xml-rpc
  - svn log