Connecting XML Processing and Term Rewriting with Tree Grammars

Stratego/XT

Martin Bravenboer

`martin@cs.uu.nl`

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

contents

- *introduce*
  xml, aterm, stratego
- *represent*
  coming to terms with xml
- *implement*
  xml transformations in stratego
- *exchange*
  data between xml and aterm tools
- *apply*
  the tools of our packages

stratego: quick intro

strategic term rewriting

- *terms* for program representations
- *rules* for basic transformation steps
- *strategies* to control the application of rules

additional goodies

- generic traversals
- concrete object syntax
- dynamic rules
- transformation tool composition
- complete applications

introducing the aterm format

- abstract data type for annotated terms
- created and maintained at UvA/CWI

```
Plus(
  Call(Var("f"), [Int(2), Int(3)])
, Var("a")
)
```

```
ClassBody(
  [ MethodDec(
      Head([], Void(), Id("hello"), [], None())
    , Block([])
    )
  ]
)
```

xml and aterm: similarities and differences

similarities

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

differences

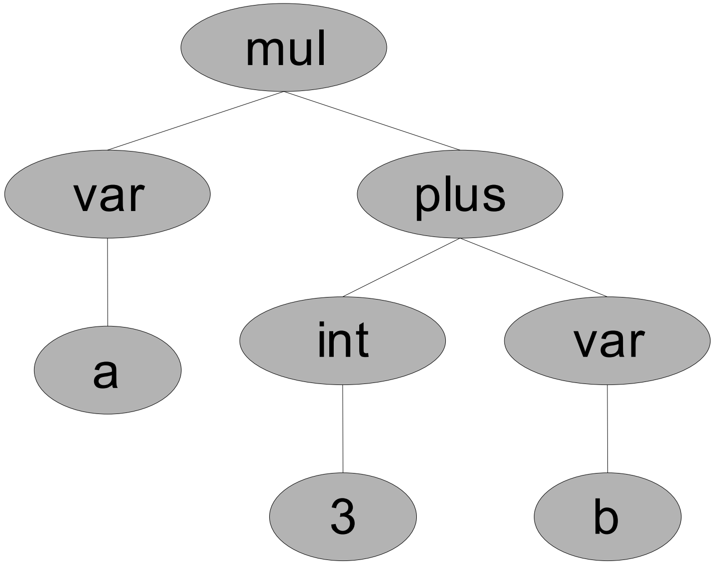
- aterm has:
  - explicit structure
  - primitive data types
  - structured annotations
- formalisms:
  - aterm format $\sim$ tree languages
  - xml $\sim$ hedge languages

xml and aterm: concepts

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

⇒ generic *syntax* for tree-like data

```
<mul>
  <var>a</var>
  <plus>
    <int>3</int>
    <var>b</var>
  </plus>
</mul>
```



```
mul(
  var("a")
, plus(
    int(3)
  , var("b")
  )
)
```

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, . . .

the xml syntax for tree-like data is

- platform,
- language,
- culture,
- and application independent.

xml and aterm: application

- *application programming interfaces (api)*
  - libraries for working with xml
  - sax, dom, pull
- *dedicated languages*
  - built-in support for xml
  - xpath, xquery, xslt, xduce, cduce
- *data binding*
  - natural representation in native data types
  - jaxb, castor, dtd2haskell, frank’s work

⇒ *how does stratego fit in?*

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

represent

levels of xml representation

- every application has its own essence of xml
- different needs, different representations
  - *xml-doc*
  - *xml-info*
  - *structured aterm*
- issues
  - namespace notation
  - character data constructs
  - empty elements
  - comments, processing instructions
  - ‘meta’ and default attributes

xml-doc

term representation of *actual syntax* of an xml document

```
<foo/>
```

```
EmptyElement(QName(None, "foo"), [])
```

```
<foo></foo>
```

```
Element(QName(None,"foo"),[],[],QName(None,"foo"))
```

```
<foo> bar  </foo>
```

```
Element(QName(None,"foo"),[],
  [Text([Literal("␣bar␣␣")])]
, QName(None,"foo")
)
```

xml-doc: character data

```
Asterix &amp; Obelix
```

```
Asterix <![CDATA[&]]> Obelix
```

```
Asterix &#x26; Obelix
```

```
Text(
  [ Literal("Asterix␣")
  , EntityRef("amp") | CDATASection("&") | HexCharRef("26")
  , Literal("␣Obelix")
  ]
)
```

xml-info

term representation of *relevant information* of an xml document

```
<foo/>
<foo></foo>
```

```
Element(Name(None, "foo"), [], [])
```

```
<foo xmlns="http://fred.org">
  <bar/>
</foo>
```

```
Element(
  Name(Some("http://fred.org"), "foo")
, []
, [Element(Name(Some("http://fred.org"), "bar"), [], [])]
)
```

xml-info: character data

```
Asterix &amp; Obelix
```

```
Asterix <![CDATA[&]]> Obelix
```

```
Asterix &#x26; Obelix
```

are all represented by

```
Text("Asterix␣&␣Obelix")
```

structured aterm

natural term representation of the *data* of an xml document

```
<section>
  <title>Tom Bombadil</title>
  <para>foo</para>
  <para>bar</para>
</section>
```

```
section(
  Some(title("Tom␣Bombadil"))
, [ para("foo")
  , para("bar")
  ]
)
```

⇒ comparable to xml data binding

implement

xml-doc in stratego: term edition

```
<title>Tom Bombadil</title>
```

```
module tom
imports xml-doc options
strategies

  main =
    output-wrap(title)

  title =
    !Element(
       QName(None, "title")
     , []
     , [Text([Literal("Tom␣Bombadil")])]
     , QName(None, "title")
     )
```

xml-doc in stratego

- any object syntax can be embedded in stratego
- concrete object syntax replaces term notation

⇒ embed xml syntax in stratego

- *quotation* – xml as stratego

```
"%>" Document  "<%" -> StrategoTerm {cons("ToTerm")}
"%>" Content   "<%" -> StrategoTerm {cons("ToTerm")}
```

- *anti-quotation* – stratego as xml

```
"<%" StrategoStrategy "%>" -> Content  {cons("FromApp")}
"<%" StrategoStrategy "%>" -> AttValue {cons("FromApp")}
```

xml-doc in stratego: xml edition

```
<title>Tom Bombadil</title>
```

```
module tom
imports xml-doc options
strategies

  main =
    output-wrap(title)

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

xml-info in stratego: term edition

```
<section>
  <title>Tom Bombadil</title>
  <para>foo</para>
  <para>bar</para>
</section>
```

```
  section =
   !Element(
      Name(None, "section")
    , []
    , [ <title>
      , Element(Name(None, "para"), [], [Text("foo")])
      , Element(Name(None, "para"), [], [Text("bar")])
      ])

  title =
    !Element(Name(None, "title"), [], [Text("Tom␣Bombadil")])
```

xml-info in stratego: xml edition

- same syntax as xml-doc in stratego
- rewrite xml-doc fragments to xml-info

```
section =
 !%><section>
      <% title %>
      <para>foo</para>
      <para>bar</para>
    </section><%

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

structured aterm in stratego

- encoding in xml irrelevant
- stratego transforms structured aterms
- comparable to data binding

```
section =
  !section(
     Some(<title>)
   , [ para("foo"), para("bar") ]
   )

title =
  !title("Tom␣Bombadil")
```

exchange

what is a structured aterm?

```
<foo><bar/><bar></bar></foo>
```

`foo (bar bar)`	`foo(bar(), bar())`
`foo (bar*)`	`foo([bar(), bar()])`
`foo (bar bar*)`	`foo(bar(), [bar()])`
`foo (bar fred* bar)`	`foo(bar(), [], bar())`
`foo (bar fred? bar)`	`foo(bar(), None(), bar())`

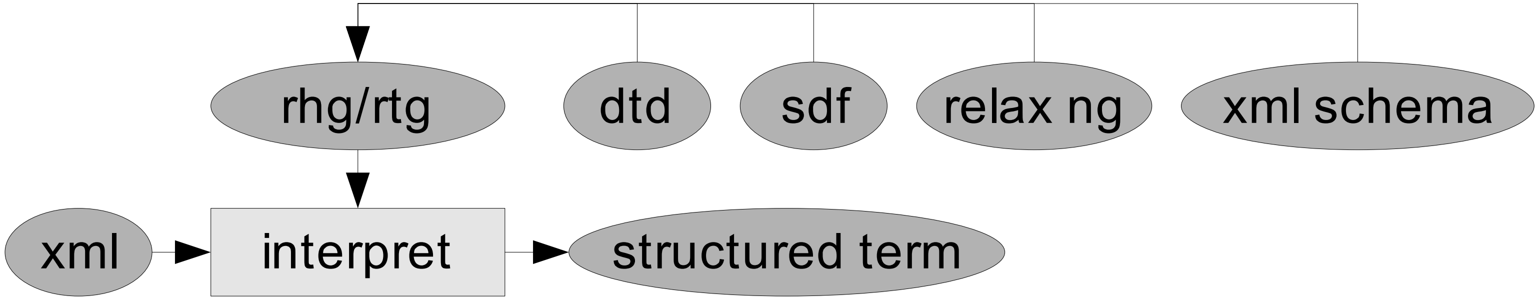
structure depends on language definition

how to structure xml

- implement structuring *by hand*

- *generate from schema* in specific schema language
  - dtd, w3c xml schema, relax ng, stratego signature, . . .
  - duplication, limitation, subsetting
  - sometimes inevitable

- *separate concerns*, use basic principles



⇒ modular and reusable data binding tools

xml: hedge languages and grammars

- *hedge* – sequence of trees
- children of xml element – sequence of *varying length*

```
regular hedge grammar
  start Section
  productions
    Section -> section (Title? Para*)
    Title   -> title   (<string>)
    Para    -> para    (<string>)
```

```
regular hedge grammar
  start Exp
  productions
    Exp -> Plus (Exp Exp)
    Exp -> Call (Var Exp*)
    Exp -> Var
    Var -> Var  (<string>)
```

aterm: tree languages and grammars

- tree language – subset of terms over *ranked alphabet*
- aterm application – *fixed number* of children

```
regular tree grammar
  start Section
  productions
    Section -> section (Title?, [Para])
    Title   -> title   (<string>)
    Para    -> para    (<string>)
```

```
regular tree grammar
  start Exp
  productions
    Exp -> Plus (Exp, Exp)
    Exp -> Call (Var, [Exp])
    Exp -> Var
    Var -> Var  (<string>)
```

interpretation against rhg

- *interpretation* – how is a document ∈ language of rhg
- adds the implicit structure of the language definition

```
<Call> <Var>f</Var> <Var>x</Var> <Var>y</Var> </Call>
```

```
appl(nonterm("Exp"), term("Call")
, iseq(
    isym(appl(nonterm("Var"), term("Var"), F, []))
  , istar(
      [ isym(appl(nonterm("Var"), term("Var"), X, []))
      , isym(appl(nonterm("Var"), term("Var"), Y, []))
      ]
    )
  )
, []
)                      F = isym(string("f"))
```

irhg to irtg to aterm

map

- sequence of terms → term
- mapping
  - star, plus → list
  - seq → tuple
  - tuple, and string, int → string, int

implode

- irtg is comparable to an exploded aterm
- implode irtg results in the ‘structured aterm’

irhg to irtg to aterm

```
<Call> <Var>f</Var> <Var>x</Var> <Var>y</Var> </Call>
```

⇒ *interpretation*

```
appl(
  nonterm("Exp"), term("Call")
, iseq(isym( ... ), istar([ isym( ...  ), isym( ... )]))
, [])
```

⇒ *irhg to irtg*

```
appl(
  nonterm("Exp"), term("Call")
, [ appl( ... ), list([ appl( ... ), appl( ... )]) ]
)
```

⇒ *implode irtg*

```
Call(Var("f"),[Var("x"),Var("y")])
```

demo: structured java term

```
public class HelloWorld {
  void hello() { }
}
```

```
<CompilationUnit>
  <ClassDec>
    <Public/>
    <Id>HelloWorld</Id>
    <ClassBody>
      <MethodDec>
        <Head> <Void/> <Id>hello</Id> </Head>
        <Block></Block>
      </MethodDec>
    </ClassBody>
  </ClassDec>
</CompilationUnit>
```

demo: structured java term

```
CompilationUnit(
 None
, []
, [ ClassDec(
   [Public]
  , Id("HelloWorld")
  , None
  , None
  , ClassBody(
    [ MethodDec(
      Head([], Void, Id("hello"), [], None)
     , Block([])
     )
    ]
   )
  )
 ]
)
```

conclude and apply

apply: xml-tools and stratego-regular

- *exchange* – interoperability

  → *aterm* tools as *xml* tools using generic *data2xml*

  ← *xml* tools as *aterm* tools using *xml-interpret*

- *implement* – rewrite xml using Stratego
  - *generate/transform xml* using xml syntax
  - *transform* a structured representation of xml

- *exchange* – generate tree grammars from
  - dtd, sdf concrete syntax definition, stratego signatures

- *validate* – an aterm against an rtg
  - check output of transformation tools
  - rtg will be contract language of xtc

future work

- more representation, more tools
- aterm, stratego, sglr and sdf2: unicode
- extended rhg/rtg: structure of strings
- disambiguation of concrete object syntax
- list-matching in stratego