jInfer BasicDTDExporter Module Description

Michal Klempa, Mário Mikula, Robert Smetana, Michal Švirec, Matej Vitásek
Advisors: RNDr. Irena Mlýnková, Ph.D., Martin Nečaský, Ph.D.

Praha, 2011

Target audience: developers willing to extend jInfer, specifically hack the DTD export.

Responsible developer:	Matej Vitásek
Required tokens:	none
Provided tokens:	cz.cuni.mff.ksi.jinfer.base.interfaces.inference.SchemaGenerator
Module dependencies:	Base
Public packages:	none

1 Introduction

This is a relatively simple implementation of a *SchemaGenerator* exporting the inferred schema to DTD.

2 Structure

The main class implementing `SchemaGenerator` inference interface and simultaneously registered as its service provider is `SchemaGeneratorImpl`. Its `start` method first topologically sorts all rules (elements) it got in the simplified grammar on input. This toposorting is necessary to avoid using anything not yet defined in the resulting schema. Afterwards, it creates their DTD string represenation.

Export of a single element is handled in the `elementToString()` method. First the actual `<!ELEMENT ...>` tag is exported, after that its attributes in a `<!ATTLIST .>` tag (if there are any).

2.1 Element content export

Elements are processed in method `elementToString()`, but the real work is done in method `regexpToString()`, which takes regexp and recursively converts it to string representation. But before, whole element is sent to method `expandIntervalsElement` of class `IntervalExpander`. Its purpose is to convert intervals on regexp and its children to those, that are representable in DTD. For example regular expression $(a\{2,5\}, b\{0,2\})$ would be transformed to $(a, a, a?, a?, a?, b?, b?)$. Class `IntervalExpander` works recursively. First, element is passed to `expandIntervalsElement`, which does (shortened):

```
public Element expandIntervalsElement(final Element treeBase) {
  return new Element(
    treeBase.getContext(),
    treeBase.getName(),
    treeBase.getMetadata(),
    expandIntervalsRegexp(treeBase.getSubnodes()),
    treeBase.getAttributes());
}
```

It calls private method `expandIntervalsRegexp` to handle regexp in that element. This private method makes big switch according to regexp type. For λ, it returns λ. Otherwise, it examines interval of regexp in method `isSafeInterval`. In DTD one can represent $+, ?, *$, so safe intervals are $\{1,\infty\}, \{0,1\}, \{0,\infty\}$ respectively. If interval is not safe, it has to be expanded. It is easy to do so, first output min-times the regexp itself - that is the minimum occurences, with interval set to $\{1, 1\}$. Then, if interval is bounded, output $max - min$-times the regexp itself with interval $\{0, 1\}$ - that is optional part. If it is unbounded, attach the regexp once, with interval $\{0, \infty\}$.

After intervals are expanded, further processing in `regexpToString` is divided by big `switch` statement of type of regexp. For λ, it simply returns `EMPY` as string. Tokens are first examined if they are `SimpleData`, if so, string `#PCDATA` is returned. If not, element name is returned. If interval of this regexp is different from $\{1, 1\}$, the interval `toString()` representation is appended.

Little complication is with complex regexps, that contain `SimpleData` somewhere inside tree. They are processed in `comboToString` method, it first checks, if there are no simple data in whole tree. If not, regexp can be outputted just as list, e.g. (a, b, c) or $(a|b|c)$ or $(a\&b\&c)$. If there is at least one simple data, flattening is applied. That means, all elements from regexp are collected into one flat list. All simple datas are trashed away. On output is string `(#PCDATA, a, b, c, d)*`, as this is the only way to represent mixed content in DTDs.

2.2 Attribute export

Code exporting attributes is in `attributeToString()`. First thing this method does is to assess the domain of a particular atribute: this is a map indexed by attribute values containing number of occurences for each such attribute. Type definition of an attribute is generated in the `DomainUtils.getAttributeType()` method. Based on a user setting, this might decide to enumerate all possible values of this attribute using the `(a|b|c)` notation, otherwise it just returns `#CDATA`.
Attribute requiredness is assessed based on `required` metadata presence. If an attribute is not deemed required, it might have a default value: if a certain value is prominent in the attribute domain (based on user setting again), it is declared default.

2.3 Preferences

All settings provided by *`BasicDTDExporter`* are project-wide, the preferences panel is in `cz.cuni.mff.ksi.jinfer.basicdtd.properties` package. As mentioned before, it is possible to set the following.

- Maximum attribute domain size which is exported as a list of all values (`(a|b|c)` notation).
- Minimal ratio an attribute value in the domain needs to have in order to be declared default.

3 Data flow

Flow of data in this module is following.

1. `SchemaGeneratorImpl` topologically sorts elements (rules) it got on input.
2. For each element, relevant portion of DTD schema is generated.
3. String representation of the schema is returned along with the information that file extension should be "dtd".

References

[Aho96] H. Ahonen. *Generating grammars for structured documents using grammatical inference methods*. PhD thesis, Department of Computer Science, University of Helsinki, Series of Publications A, Report A-1996-4, 1996.

[Bou] Ronald Bourret. Dtd parser, version 2.0. `http://www.rpbourret.com/dtdparser/index.htm`.

[HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*. Addison-Wesley, 2001.

[HW07] Yo-Sub Han and Derick Wood. Obtaining shorter regular expressions from finite-state automata. *Theor. Comput. Sci.*, 370(1-3):110–120, 2007.

[jun] Java universal network/graph framework. `http://jung.sourceforge.net/`.

[KMS$^+$a] Michal Klempa, Mário Mikula, Robert Smetana, Michal Švirec, and Matej Vitásek. *jInfer Architecture*.

[KMS$^+$b] Michal Klempa, Mário Mikula, Robert Smetana, Michal Švirec, and Matej Vitásek. *jInfer AutoEditor automaton visualization and editor module*.

[KMS$^+$c] Michal Klempa, Mário Mikula, Robert Smetana, Michal Švirec, and Matej Vitásek. *jInfer Base Module Description*.

[KMS$^+$d] Michal Klempa, Mário Mikula, Robert Smetana, Michal Švirec, and Matej Vitásek. *jInfer BasicDTDExporter Module Description*.

[KMS$^+$e] Michal Klempa, Mário Mikula, Robert Smetana, Michal Švirec, and Matej Vitásek. *jInfer BasicIGG Module Description*.

[KMS$^+$f] Michal Klempa, Mário Mikula, Robert Smetana, Michal Švirec, and Matej Vitásek. *jInfer BasicRuleDisplayer Module Description*.

[KMS$^+$g] Michal Klempa, Mário Mikula, Robert Smetana, Michal Švirec, and Matej Vitásek. jinfer javadoc. `http://jinfer.sourceforge.net/javadoc`.

[KMS$^+$h] Michal Klempa, Mário Mikula, Robert Smetana, Michal Švirec, and Matej Vitásek. *jInfer TwoStep simplifier design and implementation*.

[log] Apache log4j^{TM}. `http://logging.apache.org/log4j/`.

[loo] org.openide.util.class lookup. `http://bits.netbeans.org/dev/javadoc/org-openide-modules/org/openide/modules/doc-files/api.html`.

[mod] Module system api. `http://bits.netbeans.org/dev/javadoc/org-openide-modules/org/openide/modules/doc-files/api.html`.

[Nor] Theodore Norvell. A short introduction to regular expressions and context free grammars. `http://www.engr.mun.ca/~theo/Courses/fm/pub/context-free.pdf`.

[VMP08] Ondřej Vošta, Irena Mlýnková, and Jaroslav Pokorný. Even an ant can create an xsd. In *DASFAA'08: Proceedings of the 13th international conference on Database systems for advanced applications*, pages 35–50, Berlin, Heidelberg, 2008. Springer-Verlag.

[wik] Regular expression. `http://en.wikipedia.org/wiki/Regular_expression`.