Ausarbeitung

Projektbericht im Rahmen eines Projektes

Projekt Kognitive Systeme

Zum Thema:

Transportation of the JEdit Plug-in ProXSLbE to Eclipse

Vorgelegt von:
Hieber, Thomas

Betreuer: Martin Hofmann

Bamberg, SoSe 2008
Abstract

Programming by example is the discipline of computer science which enables the machine to learn a procedure by a set of examples. Martin Hofmann ([1]) of the Cognitive Science group at the University of Bamberg has put together a plugin for JEdit which enables a user to provide examples for an inductive synthesis system. This project’s main concern is the transportation of this application to an eclipse plugin. The functionality is based upon the ProXSLbE plugin with the intention to increase usability, especially in combination with the generation of input/output examples. Two areas of improvement were distinguished - most importantly the replacement of the original ‘core-algorithm’ by a preprocessing with the help of the DIFF-algorithm as well as a possibility for the user to mark the important passages in a string - thus helping the system to generate feasible examples. Since the automated generation was somewhat unsatisfactory we decided to let the user have some influence on the generation of the examples. This report is entirely a technical documentation describing how the plugin works on the user level, how it works underneath and where there is still work to do.
# Contents

1 Introduction 1

2 Prerequisites 2

3 User Manual 3

3.1 Input/Output-Pair 3
3.2 Examples 4
3.3 Hypothesis-Hypotheses 6

4 Technical Documentation 7

4.1 Important Data Types 7
4.1.1 InputOutputPair 7
4.1.2 PluginCache 7
4.1.3 SamplePair 7
4.1.4 BlockSelectionMachine 7
4.1.5 MarkingController 7
4.1.6 PairGeneratorController 8
4.1.7 IgorController 8
4.1.8 DOMDocumentBuilder/XSLTOutputter 9
4.2 SWT 9
4.2.1 Outline View 9
4.2.2 TextChangeView 9
4.2.3 InOutPairListView 9
4.2.4 EditPairView 10
4.2.5 Sample Selection Dialog 10
4.2.6 Hypothesis Selection Dialog 10
4.2.7 Further Comments 10

5 Known Issues 11

5.1 Easy Fixes 11
5.2 Common Fixes ................................................. 11
5.3 Tough (but important) Fixes .................................. 11

6 ProXSLbE <-> autoXSL 12

7 Appendix ......................................................... 14

7.1 Example 1 - Hühnerbrühe ................................. 15
7.2 Replace ‘b’ in Context ........................................ 21
7.3 Erroneous Replace ‘1’ by ! ................................. 27
List of Figures

1  autoXSL ......................................................... 3
2  Text Change View ........................................... 3
3  Input/Output Pair View ..................................... 4
4  Markings View ............................................... 4
5  Markings View User Marking .............................. 5
6  Pair Selection ............................................... 5
7  Hypothesis Selection ....................................... 6
8  Multi Page Editor ........................................... 6
9  Block Selection Machine ................................. 14
1 Introduction

ProXSLbE was aimed to be a demonstration of how there can be a ‘WYSIWYD’ (‘what you see is what you do’) editor to enable a user to generate XSLT stylesheets simply by providing an inductive synthesis system - IGOR - with a number of examples for automated program synthesis. This focus has not changed with our eclipse plugin autoXSL, we just enable the user to have more influence on the generation of examples. For this purpose the ‘core-algorithm’ from ProXSLbE was replaced by two components. First an automated process tries to discover differences between the original text and the text entered by the user. If he or she is not satisfied with the outcome an option for some tailoring is provided by a marking function, allowing the user to mark the differences between the two strings. In this projekt report I am going to provide a short user manual, detailed description of the important processes within the plugin as well as a report on known bugs and hickups to pave the way for future elaboration.
2 Prerequisites

The plugin comes together with a properties file which will have to be altered according to the user’s settings. In order to run IGOR it has to be installed on the system. For this two components are necessary: the maude interpreter\(^1\) and IGOR\(^2\) itself. The plugin installs quite easy - just copy the jar file into your eclipse plugin directory. It is important that you then copy the ‘patternmatcher.xsl’ into the working directory where your xml files are located.

When you first start eclipse with the plugin loaded you should begin by setting the preferences for the plugin. In the eclipse preferences dialog should now be a new entry ‘autoXSLPreferences’. Here you have to set four parameters:

- \( \text{igorHOME} = \) path to your igor directory
- \( \text{maudeHOME} = \) path to your maude directory
- \( \text{appHOME} = \) path to your working directory (here the patternmatcher and the xml files must be located)
- \( \text{igorVersion} = \) igor version (best set on 2.3)

Now there is only one more thing to do - you have to open the views shipped together with the plugin. You might also create a new perspective in order to remember the configuration of the views. The last thing to do is to open an xml file in your working directory with the ‘autoXSL Editor’.

Note that the plugin was built on the ‘Ganymede‘ build of eclipse, trouble with digressing versions are currently not known.

---

\(^1\)available on http://maude.cs.uiuc.edu/
\(^2\)available on http://www.cogsys.wiai.uni-bamberg.de/effalip/download.html
3 User Manual

The manual is not actually a full manual, but a simple walkthrough by demonstration of a simple example. I am going to explain the steps and options the user is confronted with when working with the *autoXSL* plugin.

![autoXSL](image1.png)

**Figure 1: autoXSL**

### 3.1 Input/Output-Pair

First of all we have to generate our first input/output pair which we will do right away by selecting a text node in the outline view. After we have found a node we want to work with we will have to suggest a change to the system. For this we use the ‘Text Change’-view (Figure 2). Here we can do some insert/replace/delete operations, in order to stay consistent with the example used in Martin Hofmann’s thesis we are going replace the ‘ü’ in ‘Hühnerbrühe’ by ‘ue’.

![Text Change](image2.png)

**Figure 2: Text Change View**

As soon as we confirm this change with ‘OK’ we have generated our first input/output pair which will from now on be available in the list view on the left side of our workspace (Figure 3).
3.2 Examples

When we select the new pair from the list view, it is displayed in detail in the ‘Markings’ view.

You can also see that the diff algorithm has already correctly detected the alterations just made by simply underlining the differing parts of the strings (Figure 4). If we should not be satisfied with the system’s suggestions we can put hand on the marking ourselves. All we have to do is to ‘Clear Blocks’ and start to mark the substrings which are of importance to us. In order to have the BlockSelectionMachine allow markings we have to press and hold the **CTRL** key. When the little lamp in the bottom left corner flashes green it is ready to go. Remember that the procedure of marking a pair of strings must always be ended with pressing ‘next’. In case the user made a mistake at one out his markings he can redo the current pair as long as he wants until he finally concludes by ‘next’. To help remembering which field to mark next, an unmarked field’s text is highlighted red. The markings are not saved to the cache before the ‘Write Blocks’ key is pressed. As the BlockSelectionMachine 9 shows, even after the last marking you still have to press ‘next’ before a writing is allowed. This awkward control is rather clumsy and should be dropped in the next version.

When we are done with the markings or satisfied with what the system suggested it is time to generate the examples. Next to the ‘Generate’ button is a text field to adjust the recursion depth of the algorithm generating the examples (read more in 4.1.6). The default is ‘1’ which is enough in most of the cases. After that pressing ‘Generate’ opens
up the ‘Pair Selection View’.

This dialog (Figure 6) lets us browse the generated examples, select them to be used in the specification and even alter them at need. Since the variable in the examples has been designed to be a Greek letter ‘Psi’ the user would have to copy and paste the letter from other examples in order to use it. In this context ‘insert variable’ button might be a valuable add on for a future development, for now it can only be inserted with the workaround. As soon as we are happy with the example pairs it is about time to build the specification and confront IGOR with it. The ‘Start Igor’ button triggers this process, which in term will return a result in form of one ore more hypotheses. In case the synthesizing takes too long (even though it usually does take some time) the user can abort the running process anytime by pressing ‘Stop Igor’.
3.3 Hypothesis-Hypotheses

After IGOR has finished the program synthesis it will return a result back to the plugin in form of hypotheses. If there is only one hypothesis available, it is selected and the dialog will be closed. However if there are more than just one hypotheses available, it is for the user to decide which one will be used to generate the XSLT document. In this particular case a pop up dialog allows the user to select the hypothesis by entering the number of his choice (Figure 7).

![Figure 7: Hypothesis Selection](image)

Now the dialog is closed and the results are available in the editor view (Figure 8). In this view we can now browse through all the inputs and outputs from the original XML file to the resulting transformed xml file. Furthermore we can have a look at the specification, generic IGOR result and the resulting XSLT stylesheet.

![Figure 8: Multi Page Editor](image)
4 Technical Documentation

This documentation is first of all meant as a guide through the nuts and bolts of this plugin. The obvious parts will be left aside, assuming that a reader of this document knows how to work with the eclipse plugin development environment. I am going to line out the idea and usage of the data types bearing in mind that the java documentation provides additional information about them. Since the user interface is the important part of the plugin I will put a special focus on how the views are generated, how the marking process works and finally I am going to round up the communication with IGOR.

4.1 Important Data Types

Before we start with the details of the source code let me mention a few key datatypes which will be mentioned in the following document.

4.1.1 InputOutputPair

This pair is basically the text in one XML node along with the text after the user has made changes to the original text from the text node. Both are stored as strings in ‘text_old’ and ‘text_new’. In addition to the strings the class holds a ‘Difference’ object which stores the parameters determined by the diff algorithm, as well as some data structures to store the user’s markings on the pair.

4.1.2 PluginCache

The plugin cache is a singleton used as the plugin’s main memory. Inside all the generated InputOutputPairs are stored and maintained.

4.1.3 SamplePair

4.1.4 BlockSelectionMachine

This is basically a finite state machine controlling the user marking the old/new text strings within the InputOutputPair. It uses MachineState and MachineMoves in order to set up and evaluate valid moves within the finite state machine. (Figure 9)

4.1.5 MarkingController

The main part of this controller is to use the ‘BlockSelectionMachine’ to control the user’s steps along the process. An InputOutputPair is embedded in the controller as long as it is processed by the user. When the marking is finished the controller saves the pair in the PluginCache.
4.1.6 PairGeneratorController

The most important methods in this controller are the perm and the generate methods. In order to generate ExamplePairs from a list of InputOutputPairs the controller employs two steps. First a temporary string is created by perm to represent the pattern in which variables and constants are arranged. It starts with the base case 's' representing one constant. In the course of the recursion this string is permuted while variables and constants are appended to the left and right.

1. s
2. v
3. ss
4. v
5. ss
6. sv
7. vv
8. v
9. ...

Note that the examples 3 and 4 as well as 5 and 6 are not being distinguished, so they are generated only once. With the parameter depth the recursion depth of this algorithm can be controlled (default is 1). This string is then processed in generate which simply replaces all the 'v' by a variable character and the constants by all the possible values stored in 'separators' of the InputOutputPair. Those are called 'separators' as they split a string into constant parts which do not change. Every range marked by the user or the diff algorithm is thus a 'separator'.

Concerning variables it is important to understand that even though variables are used in our example pairs we still are not forwarding them correctly to IGOR. Since IGOR enables us to declare variables along with constants in a specification header why didn't I just do it with the variable 'psi'? The answer to that is that IGOR simply initiated the variable with any possible value during synthesis including the separators we marked in the string. This led naturally to a wrong program synthesis (or none at all) so we decided to simply translate our variable into a constant to IGOR. Now the problem of a variable taking the value of one of our separators was excluded at the cost of losing the generality in our examples we would have hoped for by using variables.

4.1.7 IgorController

The IgorController uses the specification provided by its constructor to run IGOR with a maude specification built around the SamplePairs provided in the specification. The interoperation with IGOR is executed by the package 'App2IgorInterop'. All we do is set
the environment variables for the maude and IGOR home directories and then provide an input string which is our specification translated into maude. When IGOR has finished and returned a result, this result is transferred to the DOMDocumentBuilder, which takes the output string and translates it into an XSLT Sheet.

4.1.8 DOMDocumentBuilder/XSLTOutputter

The 'DOMDocumentBuilder' takes the output generated by IGOR and converts it into an XSLT Document using the XSLTOutputter. This Outputter implements the Outputter generated by an antlr parser for the IGOR result hypotheses set.

4.2 SWT

The eclipse platform uses the Standard Widget Toolkit which is a join of SWT and AWT. For the plugin GUI I employ a perspective consisting of three views, one editor together with the eclipse outline view.

4.2.1 Outline View

The outline view is a default view in eclipse and can be used to display structures of any sort. For our plugin we had to display the structure of an XML document. For this a ContentOutlinePage had to be created, so the new class XMLContentOutlinePage (see javadoc) is in charge of turning an XML file into a structure interpretable by the outline view.

4.2.2 TextChangeView

The 'TextChangeView' (Figure 2) is very simple, it consists of a text field and a button. Additionally it implements the ISelectionListener which allows to inform the view when the user has selected a node in the outline view and display the according text in the text field. When the user has altered the text a new InputOutputPair (see javadoc) is generated and stored within the PluginCache.

4.2.3 InOutPairListView

Means of displaying (Figure 3) the InputOutputPairs residing in the PluginCache. It can be used to edit, delete or insert InputOutputPairs. From this view the ‘Sample Pair Dialog’ is opened and the ‘EditPairView’ fed with the selected InputOutputPair’s data. The structure of the list is provided by a TableViewer which is filled with the two strings from the InputOutputPair along with a flag if it is selected or not. The content has to be delivered by a ContentProvider. In order to make the table data editable we have to provide it with a CellModifier which, like the IOPairContentProvider can be found in the package xml_Editor.views.content. Along with them there is a IOPairSorter and a
TableViewLabelProvider which are responsible for sorting and formatting the content in the single columns of the TableViewer.

4.2.4 EditPairView

The ‘EditPairView’ (Figure 4) consists of three basic panels, one ‘ButtonPanel’, one ‘PairDisplayPanel’ and one ‘ToolboxPanel’. The first is self explaining, the second contains two ‘Marker Panels’ and the third merely the green lamp indicating the user is holding down the CTRL button. Within one ‘MarkerPanel’ lies a text field and a canvas in which the marking of the user is displayed as a line. For this purpose the ‘EditPairView’ implements the ‘MarkerPanel’ uses a ‘MarkSelectionListener’ which in turn calls the ‘MarkingController’ as soon as a text marking has been registered.

4.2.5 Sample Selection Dialog

This is much like the ‘InOutPairListView’ a scrolled list view (Figure 6). The content provider is basically the same as in the other list view, as well as the CellModifier and the LabelProvider. Through the dialog the synthesis can be started and aborted. It automatically closes after a hypothesis was selected and transformed.

4.2.6 Hypothesis Selection Dialog

Very plain view (Figure 7) much like the ‘TectChangeView’. One text field to read the user’s hypothesis selection and one button to apply the selection. The dialog’s title provides information about the number of available hypotheses.

4.2.7 Further Comments

There are two occasions where I used asynchronous calls to the GUI by a thread, namely the ‘HypoSelector’ in the ‘DOMDocumentBuilder’ and the ‘ResultWriter’ in the IgorController. This is because the GUI cannot be accessed with a writing command by any other thread than the one which initially triggered the GUI. This is why there has to be a second thread which asynchronously tries to access the GUI until such time that it is able to trigger whatever operation it wants to carry out. In our case these are the ‘ResultWriter’ which returns the IGOR results back to the GUI and the ‘HypoSelector’ which opens the ‘Hypothesis Selection Dialog’ and returns the selected value to the ‘DOMDocumentBuilder’.

The patternmatcher used in Martin Hofmann’s plugin has turned out to be not running with the saxon parser. Therefore it was adjusted to match the XSLT 2.0 standard, employing data types in variables and functions along with a differentiation between ‘value-of select’ and ‘sequence-of select’ (see in appendix 7.3).

For more detailed technical information see the java documentation appended to the source code as well as the comments within the sources themselves.
5 Known Issues

As this project turned out to be quite comprehensive and tricky in the end, there are some issues which had been left aside at the beginning but could not be taken care of at the very end. I have categorised them into three groups in order to ease the decision process of anyone starting to improve on this plugin.

5.1 Easy Fixes

- The graphics are not very cute, especially the ‘EditPairPanel’ - people with a proclivity towards aesthetics are strongly encouraged to brush up on this.

- The selection of a pair from the ‘InOutListView’ in order to change the markings always sets the ‘chosen’ attribute of the pair to ‘false’ even though it is still marked in the list. This leads to an empty sample pair dialog unless you check the pair again.

- As already mentioned, the BlockSelectionMachine forces the user to press ‘next’ before saving a pair, even if the user has just marked his last pair. This should be changed by combining the ‘next’ step with the ‘write’ step.

5.2 Common Fixes

- The TreeViewer of the XMLContentProvider struggles with too big XML Documents. Since the TreeViewer needs the elements created before adding to the outline view it will run out of memory on big XML files at the moment. There must be a way around this - I just didn’t find it in time.

- The ‘Hypothesis Selection Dialog’ is not very elegant - there could be another way of solving this, maybe by presenting the single hypotheses to the user having him decide whether he wants it applied or not. Even though the IGOR result is displayed in the editor, it is not very straightforward to force the user to switch the windows before deciding.

5.3 Tough (but important) Fixes

- Martin Hofmann’s original version was able to perform insert, replace and above all reverse operations. These are no longer possible in the current version of this plug in. This may be because we decided to skip Martin’s approach using ‘separators’ and ‘constants’ in order to build the examples by skipping the ‘constants’ for ‘variables’. Thus we had to mask the variables as constants to IGOR which is against intuition. There should be a new approach which makes use of the possibility to declare variables in IGOR.

- Sometimes there are hypotheses which build a wrong boolean expression for the ‘test’ parameter of the ‘if-statement in the XSLT sheet which causes the transformer to throw an exception. I have tried fixing these statements but the problem is that
they cannot be cleanly generated by the current version of the ‘XSLTOutputter’. The focus of any improvement on this plugin should be to work this out.

6 ProXSLbE <-> autoXSL

Finally, it seems appropriate to draw a comparison between the two plugins and find out how their functionality corresponds and how they differ. First of all, both can handle delete, insert and replace, with some restrictions. Some ‘special’ replacement problems cause erroneous program outputs (see appendix 7.3). The list operations cause additional problems since the generation of examples becomes now quite tricky when using the marking and generation approach. This is why the ‘Sample Pair Dialog’ has been kept open for the user to edit the examples entirely. Until the completion of the plugin I have not been able to successfully provide examples for a list operation of any kind. This is why any next version of this plugin should begin with a close look at the example generation and the building of the IGOR specification. At this moment the new fancy way has to be abandoned and the user has to do a little more thinking than we wanted him to do initially. While improving the user interface for simple insert, replace and delete problems we still have to sort out those more complex problems. So all in all you might call ‘autoXSL’ a partial improvement of ‘ProXSLbE’ in all the aspects mentioned. But as for functionality both versions do not differ in a general observation.
References


7 Appendix

Figure 9: Block Selection Machine
7.1 Beispiel 1 - Hühnerbrühe

Listing 1: Initial XML File

```xml
<tests>
  <test id="1">
    <replace>Hühnerbrühe</replace>
    <replace>_2>Wörterbücher</replace_2>
    <insert_1>a b c e f g h i j k</insert_1>
    <delete>Text und so &lt;!-- --&gt; weiter Text &lt;!-- --&gt; und Text</delete>
    <insert_2>aaa! bbb! ccc!</insert_2>
    <list_1>1-2.3-4.5</list_1>
    <list_2>1-2-3-4-5-6</list_2>
    <list_3>10/09/2007</list_3>
  </test>
  <test id="2">
    <replace>Hühnerbrühe</replace>
    <replace>_2>b ä d e l c</replace_2>
    <insert_1>a b c e f g h i j k</insert_1>
    <delete>Text und so &lt;!-- --&gt; weiter Text &lt;!-- --&gt; und Text</delete>
    <insert_2>111!222!333!</insert_2>
    <list_1>11-22.33-44.55</list_1>
    <list_2>3-4-5-6-7-8-9-10</list_2>
    <list_3>10/09/2007</list_3>
  </test>
  <test id="3">
    <replace>Hühnerbrühe</replace>
    <replace>_2>b ä d e l c</replace_2>
    <insert_1>a b c e f g h i j k</insert_1>
    <delete>Text &lt;!-- --&gt; noch mehr Text &lt;!-- --&gt;</delete>
    <insert_2>aaa! bbb! ccc!</insert_2>
    <list_1>1-2.3-4.5</list_1>
    <list_2>1-2-3-4-5</list_2>
    <list_3>10/09/2007</list_3>
  </test>
  <test id="4">
    <replace>Hühnerbrühe</replace>
    <replace>_2>b ä d e l c</replace_2>
    <insert_1>a b c e f g h i j k</insert_1>
    <insert_2>aaa! bbb! ccc!</insert_2>
    <delete>Text &lt;!-- --&gt; noch mehr Text &lt;!-- --&gt;</delete>
    <insert_2>aaa! bbb! ccc!</insert_2>
    <list_1>1-2.3-4.5</list_1>
    <list_2>1-2-3-4-5</list_2>
    <list_3>10/09/2007</list_3>
  </test>
</tests>
```
Listing 2: Maude Specification

1 fmod autoMOD is
2 sorts Subs SubsList InVec .
3
4 ∗∗∗ −−− String to Hex-Value −−−
5 ∗∗∗ ü → #x00fc
6 ∗∗∗ ue → #x0075#x0065
7 ∗∗∗ [Psi] → #x003a8
8
9 ops #x00fc #x0075#x0065 #x003a8 : → Subs [ctor] .
10 op <> : → SubsList [ctor] .
12 op in : SubsList → InVec [ctor] .
13 op autoFUN : SubsList → SubsList [metadata "induce"] .
14 eq autoFUN(<> <> <> ) = <> .
15 eq autoFUN((#x00fc <>)) = (#x0075#x0065 <>) .
16 eq autoFUN((#x0075#x0065 <>)) = (#x0075#x0065 <>) .
17 eq autoFUN((#x00fc (#x003a8 <>))) = (#x0075#x0065 (#x003a8 <>)) .
18 eq autoFUN((#x0075#x0065 (#x003a8 <>))) = (#x0075#x0065 (#x003a8 <>)) .
19 eq autoFUN((#x003a8 (#x00fc <>))) = (#x003a8 (#x0075#x0065 <>)) .
20 eq autoFUN((#x003a8 (#x0075#x0065 <>))) = (#x003a8 (#x0075#x0065 <>)) .
21 eq autoFUN((#x00fc (#x00fc <>))) = (#x0075#x0065 (#x0075#x0065 <>)) .
22 eq autoFUN((#x0075#x0065 (#x0075#x0065 <>))) = (#x0075#x0065 (#x0075#x0065 <>)) .
23 eq autoFUN((#x003a8 <>)) = (#x003a8 <>) .
24 eq autoFUN((#x003a8 (#x003a8 <>))) = (#x003a8 (#x003a8 <>)) .
25 endfm
Listing 3: IGOR Output


Cognitive Science Group - University of Bamberg 17
Listing 5: Transformed XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tests>
  <test id="1">
    <replace>Huehnerbrühe</replace>
    <replace_2>Wörterbücher</replace_2>
    <insert_1>abcde</insert_1>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <delete>Text und so &lt;--&gt; weiter Text &lt;--&gt; und Text</delete>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <list_1 sep_count="1">1-2-3-4.5</list_1>
    <list_2 sep_count="2">1-2-3-4-5-6</list_2>
    <list_3 sep_count="2">10/09/2007</list_3>
  </test>
  <test id="2">
    <replace>Huehnerbrühe</replace>
    <replace_2>bäntel</replace_2>
    <insert_1>abcde</insert_1>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <delete>Text &lt;--&gt; weiter Text &lt;--&gt; und Text</delete>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <list_1 sep_count="1">11-22.33-44.55</list_1>
    <list_2 sep_count="2">3-4-5-6-7-8-9-10</list_2>
    <list_3 sep_count="2">10/09/2007</list_3>
  </test>
  <test id="3">
    <replace>Huehnerbrühe</replace>
    <replace_2>bäntel</replace_2>
    <insert_1>abcde</insert_1>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <delete>Text &lt;--&gt; weiter Text &lt;--&gt; und mehr Text &lt;--&gt;</delete>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <list_1 sep_count="1">1-2.3-4.5</list_1>
    <list_2 sep_count="2">1-2-3-4-5</list_2>
    <list_3 sep_count="2">10/09/2007</list_3>
  </test>
  <test id="4">
    <replace>Huehnerbrühe</replace>
    <replace_2>bäntel</replace_2>
    <insert_1>abcde</insert_1>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <delete>Text &lt;--&gt; weiter Text &lt;--&gt; und mehr Text &lt;--&gt;</delete>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <list_1 sep_count="1">1-2-3-4.5</list_1>
    <list_2 sep_count="2">1-2-3-4-5</list_2>
  </test>
</tests>
```
7.2 Replace ‘b’ in Context

Listing 6: Initial XML File

```xml
<tests>
  <test id="1">
    <replace>Hühnerbrühe</replace>
    <replace_2>Wörterbücher</replace_2>
    <insert_1>abcdefghijkl</insert_1>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <delete>Text und so &lt;!--&lt;--&gt; samen Text &lt;!--&lt;--&gt; und Text</delete>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <list_1 sep_count="1">1-2.3-4.5</list_1>
    <list_2 sep_count="2">1-2-3-4-5-6</list_2>
    <list_3 sep_count="2">10/09/2007</list_3>
  </test>
  <test id="2">
    <replace>Hühnerbrühe</replace>
    <replace_2>bändel</replace_2>
    <insert_1>abcdefghijkl</insert_1>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <delete>Text und so &lt;!--&lt;--&gt; samen Text &lt;!--&lt;--&gt; und Text</delete>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <list_1 sep_count="1">11-22.33-44.55</list_1>
    <list_2 sep_count="2">3-4-5-6-7-8-9-10</list_2>
    <list_3 sep_count="2">10/09/2007</list_3>
  </test>
  <test id="3">
    <replace>Hühnerbrühe</replace>
    <replace_2>bändel</replace_2>
    <insert_1>abcdefghijkl</insert_1>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <delete>Text &lt;!--&lt;--&gt; samen Text &lt;!--&lt;--&gt; und Text</delete>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <list_1 sep_count="1">1-2.3-4.5</list_1>
    <list_2 sep_count="2">1-2-3-4-5</list_2>
    <list_3 sep_count="2">10/09/2007</list_3>
  </test>
  <test id="4">
    <replace>Hühnerbrühe</replace>
    <replace_2>bändel</replace_2>
    <insert_1>abcdefghijkl</insert_1>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <delete>Text &lt;!--&lt;--&gt; samen Text &lt;!--&lt;--&gt; und Text</delete>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <list_1 sep_count="1">1-2.3-4.5</list_1>
    <list_2 sep_count="2">1-2-3-4-5</list_2>
  </test>
</tests>
```
Listing 7: Maude Specification

1 fmod autoMOD is
2 sorts Subs SubsList InVec .
3
4 ∗∗∗ −−− String to Hex−Value −−−
5 ∗∗∗ !b −−−> #x0021#x0062
6 ∗∗∗ !w −−−> #x0021#x0077
7 ∗∗∗ [Psi] −−−> #x003a8
8
9 ops #x0021#x0062 #x0021#x0077 #x003a8 : −> Subs [ctor] .
10 op <> : −> SubsList [ctor] .
12 op autoFUN : SubsList −> SubsList [metadata "induce"] .
13
14 eq autoFUN(<> <> = <> .
15 eq autoFUN(#x0021#x0062 <>)) = (#x0021#x0077 <>) .
16 eq autoFUN(#x0021#x0077 <>)) = (#x0021#x0077 <>) .
17 eq autoFUN(#x0021#x0062 (#x003a8 <>))) = (#x0021#x0077 (#x003a8 <>))) .
18 eq autoFUN(#x0021#x0077 (#x003a8 <>))) = (#x0021#x0077 (#x003a8 <>))) .
19 eq autoFUN(#x003a8 (#x0021#x0062 <>))) = (#x003a8 (#x0021#x0077 <>))) .
20 eq autoFUN(#x003a8 (#x0021#x0077 <>))) = (#x003a8 (#x0021#x0077 <>))) .
21 eq autoFUN(#x0021#x0062 (#x0021#x0062 <>))) = (#x0021#x0077 (#x0021#x0077 <>))) .
22 eq autoFUN(#x0021#x0077 (#x0021#x0077 <>))) = (#x0021#x0077 (#x0021#x0077 <>))) .
23 eq autoFUN(#x003a8 <>)) = (#x003a8 <>)) .
24 eq autoFUN(#x003a8 (#x003a8 <>))) = (#x003a8 (#x003a8 <>))) .
25 endfm
Listing 8: IGOR Output

1 hypo (true, 3, eq 'Sub2 [' X1: Subs, 'X2: SubsList ] = 'autoFUN [ 'Sub53 [ 'X1: Subs, 'X2: SubsList ] ] [ none ]).
3 eq 'autoFUN ' [ X1: Subs, 'X2: SubsList ] = 'Sub2 [ 'X1: Subs, 'X2: SubsList ] [ none ].
9 eq 'autoFUN ' [ X1: Subs, 'X2: SubsList ] = 'Sub2 [ 'X1: Subs, 'X2: SubsList ] [ none ].

Listing 9: XSLT Output

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <xsl:output xmlns:xsl="http://www.w3.org/1999/XSL/Transform" method="xml" indent="yes" encoding="UTF-8" />
  <xsl:include href="/home/kaiserfranze/workspace/XMLEditor//patternmatcher.xsl" />
  <!-- Definition of used substringlist functional characters -->
  <xsl:variable name="separators" select="('!', ',', ' ')">
    <!-- Copy non-target nodes -->
    <xsl:template match="/*|@*|
```
<xsl:with-param name="input" select="$Sub1_arg" />
</xsl:call-template>
<xsl:variable name="Sub2_arg" as="xs:string">
<xsl:call-template name="Sub2">
<xsl:with-param name="input" select="$Sub2_arg" />
</xsl:call-template>
</xsl:variable>
<xsl:when>
<xsl:choose>
<xsl:template name="Sub1">
<xsl:param name="input" as="xs:string" />
<xsl:choose>
<xsl:when test="igor:match-pattern(('[X]' , 'Xs') , $input)= true()">
<xsl:variable name="X1_Subs" select="$input[1]" as="xs:string" />
<xsl:variable name="X2_SubsList" select="subsequence($input, 2)" as="xs:string" />
<xsl:variable name="cond_arg0" as="xs:string">
<xsl:sequence select="$X1_Subs" />
</xsl:variable>
<xsl:variable name="cond_arg1" as="xs:string">
<xsl:sequence select="'!b'" />
</xsl:variable>
<xsl:variable name="cond_arg4" as="xs:string">
<xsl:sequence select="'!b'" />
</xsl:variable>
<xsl:variable name="cond_arg5" as="xs:string">
<xsl:sequence select="'!w'" />
</xsl:variable>
<xsl:variable name="cond_arg6" as="xs:string">
<xsl:sequence select="'!w'" />
</xsl:variable>
<xsl:variable name="cond_arg7" as="xs:string">
<xsl:sequence select="'!b'" />
</xsl:variable>
<xsl:variable name="cond_arg9" as="xs:string">
<xsl:sequence select="'!w'" />
</xsl:variable>
<xsl:variable name="cond_arg10" as="xs:string">
<xsl:sequence select="'!b'" />
</xsl:variable>
<xsl:variable name="cond_arg11" as="xs:string">
<xsl:sequence select="'!w'" />
</xsl:variable>
<xsl:variable name="cond arg0" as="xs:string">
<xsl:sequence select="'!w'" />
</xsl:variable>
<xsl:variable name="cond arg1" as="xs:string">
<xsl:sequence select="'!b'" />
</xsl:variable>
<xsl:variable name="cond arg4" as="xs:string">
<xsl:sequence select="'!b'" />
</xsl:variable>
<xsl:variable name="cond arg5" as="xs:string">
<xsl:sequence select="'!w'" />
</xsl:variable>
<xsl:variable name="cond arg6" as="xs:string">
<xsl:sequence select="'!w'" />
</xsl:variable>
<xsl:variable name="cond arg7" as="xs:string">
<xsl:sequence select="'!b'" />
</xsl:variable>
<xsl:choose>
<xsl:when test="($cond_arg0=$cond_arg1) and ($cond_arg4=$cond_arg5)">
<xsl:variable name="X1_Subs" select="$input[1]" as="xs:string" />
<xsl:variable name="X2_SubsList" select="subsequence($input, 2)" as="xs:string" />
<xsl:variable name="cond_arg0" as="xs:string">
<xsl:sequence select="'!w'" />
</xsl:variable>
<xsl:variable name="cond_arg1" as="xs:string">
<xsl:sequence select="'!b'" />
</xsl:variable>
<xsl:variable name="cond_arg4" as="xs:string">
<xsl:sequence select="'!b'" />
</xsl:variable>
<xsl:variable name="cond_arg5" as="xs:string">
<xsl:sequence select="'!w'" />
</xsl:variable>
<xsl:variable name="cond_arg6" as="xs:string">
<xsl:sequence select="'!w'" />
</xsl:variable>
<xsl:variable name="cond_arg7" as="xs:string">
<xsl:sequence select="'!b'" />
</xsl:variable>
<xsl:variable name="cond arg0" as="xs:string">
<xsl:sequence select="'!w'" />
</xsl:variable>
<xsl:variable name="cond arg1" as="xs:string">
<xsl:sequence select="'!b'" />
</xsl:variable>
<xsl:variable name="cond arg4" as="xs:string">
<xsl:sequence select="'!b'" />
</xsl:variable>
<xsl:variable name="cond arg5" as="xs:string">
<xsl:sequence select="'!w'" />
</xsl:variable>
<xsl:variable name="cond arg6" as="xs:string">
<xsl:sequence select="'!w'" />
</xsl:variable>
<xsl:variable name="cond arg7" as="xs:string">
<xsl:sequence select="'!b'" />
</xsl:variable>
<xsl:choose>
</xsl:when>
</xsl:choose>
</xsl:template>
</xsl:when>
</xsl:choose>
</xsl:template>
</xsl:stylesheet>
Listing 10: Transformed XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tests>
  <test id="1">
    <replace>Hühnerbrühe</replace>
    <replace_2>Wörterbücher</replace_2>
    <insert_1>1abc1efgh1ijkl</insert_1>
    <insert_2>aaa!wbb!ccc!</insert_2>
    <text>Text und so &lt;!-- --&gt; weiter Text &lt;!-- --&gt; und Text</text>
    <delete>
      <insert_2>aaa!wbb!ccc!</insert_2>
      <list_1 sep_count="1">1-2-3-4-5</list_1>
      <list_2 sep_count="2">1-2-3-4-5-6</list_2>
      <list_3 sep_count="2">10/09/2007</list_3>
    </delete>
  </test>

  <test id="2">
    <replace>Hühnerbrühe</replace>
    <replace_2>bändel</replace_2>
    <insert_1>1abc1efgh1ijkl</insert_1>
    <insert_2>aaa!wbb!ccc!</insert_2>
    <text>Text und so &lt;!-- --&gt; weiter Text &lt;!-- --&gt; und Text</text>
    <delete>
      <insert_2>aaa!wbb!ccc!</insert_2>
      <list_1 sep_count="1">11-22.33-44.55</list_1>
      <list_2 sep_count="2">3-4-5-6-7-8-9-10</list_2>
      <list_3 sep_count="2">10/09/2007</list_3>
    </delete>
  </test>

  <test id="3">
    <replace>Hühnerbrühe</replace>
    <replace_2>bändel</replace_2>
    <insert_1>1abc1efgh1ijkl</insert_1>
    <insert_2>aaa!wbb!ccc!</insert_2>
    <text>Text &lt;!-- --&gt; noch mehr Text &lt;!-- --&gt;</text>
    <delete>
      <insert_2>aaa!wbb!ccc!</insert_2>
      <list_1 sep_count="1">1-2-3-4-5</list_1>
      <list_2 sep_count="2">1-2-3-4-5</list_2>
      <list_3 sep_count="2">10/09/2007</list_3>
    </delete>
  </test>

  <test id="4">
    <replace>Hühnerbrühe</replace>
    <replace_2>bändel</replace_2>
    <insert_1>1abc1efgh1ijkl</insert_1>
    <insert_2>aaa!wbb!ccc!</insert_2>
    <text>Text &lt;!-- --&gt; noch mehr Text &lt;!-- --&gt;</text>
    <delete>
      <insert_2>aaa!wbb!ccc!</insert_2>
      <list_1 sep_count="1">1-2-3-4-5</list_1>
      <list_2 sep_count="2">1-2-3-4-5</list_2>
    </delete>
  </test>
</tests>
```
7.3 **Erroneous Replace ‘1’ by !**

Listing 11: Initial XML File

```xml
<tests>
  <test id="1">
    <replace>Hühnerbrühe</replace>
    <replace_2>Wörterbücher</replace_2>
    <insert_1>labcdefghijk</insert_1>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <delete>Text und so &lt;!--&gt; weiter Text &lt;!--&gt; und Text</delete>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <list_1 sep_count="1">1-2.3-4.5</list_1>
    <list_2 sep_count="2">1-2-3-4-5-6</list_2>
    <list_3 sep_count="2">10/09/2007</list_3>
  </test>
  <test id="2">
    <replace>Hühnerbrühe</replace>
    <replace_2>bändel</replace_2>
    <insert_1>labcdefghijk</insert_1>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <delete>Text und so &lt;!--&gt; weiter Text &lt;!--&gt; und Text</delete>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <list_1 sep_count="1">11-22.33-44.55</list_1>
    <list_2 sep_count="2">3-4-5-6-7-8-9-10</list_2>
    <list_3 sep_count="2">10/09/2007</list_3>
  </test>
  <test id="3">
    <replace>Hühnerbrühe</replace>
    <replace_2>bändel</replace_2>
    <insert_1>labcdefghijk</insert_1>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <delete>Text &lt;!--&gt; noch mehr Text &lt;!--&gt;</delete>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <list_1 sep_count="1">1-2.3-4.5</list_1>
    <list_2 sep_count="2">1-2-3-4-5</list_2>
    <list_3 sep_count="2">10/09/2007</list_3>
  </test>
  <test id="4">
    <replace>Hühnerbrühe</replace>
    <replace_2>bändel</replace_2>
    <insert_1>labcdefghijk</insert_1>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <delete>Text &lt;!--&gt; noch mehr Text &lt;!--&gt;</delete>
    <insert_2>aaa!bbb!ccc!</insert_2>
    <list_1 sep_count="1">1-2.3-4.5</list_1>
    <list_2 sep_count="2">1-2-3-4-5</list_2>
  </test>
</tests>
```
Listing 12: Maude Specification

fmod autoMOD is
sorts Subs SubsList InVec .

*** String to Hex-Value ***
ops #x0031 #x0021 #x003a8 : -> Subs [ctor] .
op <> : -> SubsList [ctor] .
op in : SubsList -> InVec [ctor] .
op autoFUN : SubsList -> SubsList [metadata "induce"] .
eq autoFUN(<>) = <> .
eq autoFUN((#x0031 <>)) = (#x0021 <>) .
eq autoFUN((#x0031 ( #x003a8 <> ))) = (#x0021 ( #x003a8 <> )) .
eq autoFUN((#x0021 ( #x003a8 <> ))) = (#x0021 ( #x003a8 <> )) .
eq autoFUN((#x003a8 ( #x0021 <> ))) = (#x003a8 ( #x0021 <> )) .
eq autoFUN((#x003a8 ( #x0021 <> ))) = (#x003a8 ( #x0021 <> )) .
eq autoFUN((#x003a8 ( #x003a8 <> ))) = (#x003a8 ( #x003a8 <> )) .
endfm
Listing 13: IGOR Output

Listing 14: XSLT Output

```xml
<?xml version="1.0" encoding="UTF-8"?>
   <xsl:output xmlns:xsl="http://www.w3.org/1999/XSL/Transform" method="xml" indent="yes" encoding="UTF-8" />
   <xsl:include href="/home/kaiserfranz/workspace/XMLEditor//patternmatcher.xsl" />
   <!-- Definition of used substringlist functional characters-->
   <xsl:variable name="separators" select="(\'1\', \'!\')" />
   <!--Copy non-target nodes-->
   <xsl:template match="/\[1\]\*|\[@*\]|text()">
      <xsl:apply-templates select="*|@*|text()"/>
   </xsl:template>
   <!--Entrance function for target node-->
   <xsl:template match="insert_1">
      <!--Copy everything else but text-->
      <xsl:copy>
         <xsl:choose>
            <xsl:when test="igor:match-pattern((\'X\', \'Xs\') , $input) = true()">
               <xsl:variable name="X1_Subs" select="$input[1]" as="xs:string"/>
               <xsl:variable name="X2_SubsList" select="subsequence($input,2)" as="xs:string"/>
            </xsl:when>
            <xsl:when test="igor:match-pattern((\'E\') , $input) = true()">
               <xsl:sequence select="'*'"/>
            </xsl:when>
            <xsl:when test="igor:match-pattern((\'X\', \'Xs\') , $input) = true()">
               <xsl:variable name="X1_Subs" select="$input[1]" as="xs:string"/>
               <xsl:variable name="X2_SubsList" select="subsequence($input,2)" as="xs:string"/>
            </xsl:when>
            <xsl:when test="igor:match-pattern((\'E\') , $input) = true()">
               <xsl:sequence select="'*'"/>
            </xsl:when>
            <xsl:when test="igor:match-pattern((\'X\', \'Xs\') , $input) = true()">
               <xsl:variable name="X1_Subs" select="$input[1]" as="xs:string"/>
               <xsl:variable name="X2_SubsList" select="subsequence($input,2)" as="xs:string"/>
            </xsl:when>
            <xsl:when test="igor:match-pattern((\'E\') , $input) = true()">
               <xsl:sequence select="'*'"/>
            </xsl:when>
         </xsl:choose>
      </xsl:copy>
   </xsl:template>
   <xsl:template name="Sub2">
      <xsl:param name="input" as="xs:string"/>
      <xsl:when test="igor:match-pattern((\'X\', \'Xs\') , $input) = true()">
         <xsl:call-template name="autoFUN">
            <xsl:with-param name="input" select="igor:separate(text() , ($separators))"/>
         </xsl:call-template>
      </xsl:when>
      <xsl:copy>
         <xsl:choose>
            <xsl:when test="igor:match-pattern((\'X\', \'Xs\') , $input) = true()">
               <xsl:variable name="Sub53_arg" as="xs:string"/>
               <xsl:variable name="Sub53" as="xs:string"/>
               <xsl:sequence select="$Sub53"/>
               <xsl:variable name="Sub53_string" as="xs:string"/>
               <xsl:sequence select="$Sub53_string"/>
            </xsl:when>
            <xsl:when test="igor:match-pattern((\'E\') , $input) = true()">
               <xsl:sequence select="'*'"/>
            </xsl:when>
            <xsl:when test="igor:match-pattern((\'X\', \'Xs\') , $input) = true()">
               <xsl:variable name="X1_Subs" select="$input[1]" as="xs:string"/>
               <xsl:variable name="X2_SubsList" select="subsequence($input,2)" as="xs:string"/>
            </xsl:when>
            <xsl:when test="igor:match-pattern((\'E\') , $input) = true()">
               <xsl:sequence select="'*'"/>
            </xsl:when>
         </xsl:choose>
      </xsl:copy>
   </xsl:template>
   <xsl:template name="Sub53">
      <xsl:param name="input" as="xs:string"/>
      <xsl:when test="igor:match-pattern((\'X\', \'Xs\') , $input) = true()">
         <xsl:call-template name="autoFUN">
            <xsl:with-param name="input" select="$autoFUN_arg"/>
         </xsl:call-template>
      </xsl:when>
      <xsl:copy>
         <xsl:choose>
            <xsl:when test="igor:match-pattern((\'X\', \'Xs\') , $input) = true()">
               <xsl:variable name="Sub53_arg" as="xs:string"/>
               <xsl:variable name="Sub53" as="xs:string"/>
               <xsl:sequence select="$Sub53"/>
               <xsl:variable name="Sub53_string" as="xs:string"/>
               <xsl:sequence select="$Sub53_string"/>
            </xsl:when>
            <xsl:when test="igor:match-pattern((\'E\') , $input) = true()">
               <xsl:sequence select="'*'"/>
            </xsl:when>
            <xsl:when test="igor:match-pattern((\'X\', \'Xs\') , $input) = true()">
               <xsl:variable name="X1_Subs" select="$input[1]" as="xs:string"/>
               <xsl:variable name="X2_SubsList" select="subsequence($input,2)" as="xs:string"/>
            </xsl:when>
            <xsl:when test="igor:match-pattern((\'E\') , $input) = true()">
               <xsl:sequence select="'*'"/>
            </xsl:when>
         </xsl:choose>
      </xsl:copy>
   </xsl:template>
   <xsl:template name="autoFUN">
      <xsl:param name="input" as="xs:string"/>
      <xsl:when test="igor:match-pattern((\'E\') , $input) = true()">
         <xsl:sequence select="'*'"/>
      </xsl:when>
      <xsl:copy>
         <xsl:choose>
            <xsl:when test="igor:match-pattern((\'X\', \'Xs\') , $input) = true()">
               <xsl:variable name="X1_Subs" select="$input[1]" as="xs:string"/>
               <xsl:variable name="X2_SubsList" select="subsequence($input,2)" as="xs:string"/>
            </xsl:when>
            <xsl:when test="igor:match-pattern((\'E\') , $input) = true()">
               <xsl:sequence select="'*'"/>
            </xsl:when>
            <xsl:when test="igor:match-pattern((\'X\', \'Xs\') , $input) = true()">
               <xsl:variable name="X1_Subs" select="$input[1]" as="xs:string"/>
               <xsl:variable name="X2_SubsList" select="subsequence($input,2)" as="xs:string"/>
            </xsl:when>
            <xsl:when test="igor:match-pattern((\'E\') , $input) = true()">
               <xsl:sequence select="'*'"/>
            </xsl:when>
         </xsl:choose>
      </xsl:copy>
   </xsl:template>
</xsl:stylesheet>
```

Cognitive Science Group - University of Bamberg
<xs:with-param name="input" select="$Sub1_arg" />
</xs:call-template>
<xs:variable name="Sub2_arg" as="xs:string"/>
<xs:sequence select="$X1_Sub"/>
<xs:sequence select="$X2_SubList"/>
</xs:variable>
<xs:call-template name="Sub2">
<xs:with-param name="input" select="$Sub2_arg" />
</xs:call-template>
</xs:when>
</xs:choose>
</xs:template>
<xs:template name="Sub1">
<xs:param name="input" as="xs:string"/>
<xs:choose>
<xs:when test="igor:match-pattern ( ( 'X', 'Xs' ) , $input ) = true ()">
<xs:variable name="X1_Sub" select="$input[1]" as="xs:string"/>
<xs:sequence select="$X1_Sub"/>
</xs:variable>
<xs:variable name="cond_arg_0" as="xs:string"/>
<xs:sequence select="$X1_Sub"/>
</xs:variable>
<xs:variable name="cond_arg_1" as="xs:string"/>
<xs:sequence select="'1'"/>
</xs:variable>
<xs:variable name="cond_arg_2" as="xs:string"/>
<xs:sequence select="$X1_Sub"/>
</xs:variable>
<xs:variable name="cond_arg_3" as="xs:string"/>
<xs:sequence select="'1'"/>
</xs:variable>
<xs:variable name="var" as="xs:string"/>
<xs:sequence select="$X1_Sub"/>
</xs:variable>
<xs:if test="($cond_arg_0=$cond_arg_1) and ($cond_arg_2=$cond_arg_3)">
<xs:variable name="X1_Sub" select="$input[1]" as="xs:string"/>
</xs:variable>
<xs:variable name="X2_SubList" select="subsequence($input,2)" as="xs:string"/>
</xs:if>
<xs:variable name="cond_arg_6" as="xs:string"/>
<xs:sequence select="$X1_Sub"/>
</xs:variable>
<xs:variable name="cond_arg_7" as="xs:string"/>
<xs:sequence select="'1'"/>
</xs:variable>
<xs:if test="not($cond_arg_6=$cond_arg_7)">
<xs:variable name="X1_Sub" select="$input[1]" as="xs:string"/>
</xs:variable>
<xs:value-of select="$X1_Sub"/>
</xs:if>
</xs:when>
</xs:choose>
</xs:template>
</xs:stylesheet>
Listing 15: Transformed XML

1 <?xml version="1.0" encoding="UTF-8"?>
2 <tests>
3     <test id="1">
4         <replace>Hühnerbrühe</replace>
5         <replace_2>Wörterbücher</replace_2>
6             <insert_1>!abc!efgh!ijk!</insert_1>
7         <insert_2>aaa!bbb!ccc!</insert_2>
8         <delete>Text und so &lt;/i&gt; --- &lt;i&gt;Text --- &lt;/i&gt; und Text&lt;/delete>
9             <insert_2>aaa!bbb!ccc!</insert_2>
10        <list_1 sep_count="1">1-2-3-4-5</list_1>
11        <list_2 sep_count="2">1-2-3-4-5-6</list_2>
12        <list_3 sep_count="2">10/09/2007</list_3>
13     </test>
14     <test id="2">
15         <replace>Hühnerbrühe</replace>
16         <replace_2>bändel</replace_2>
17             <insert_1>!abc!efgh!ijk!</insert_1>
18         <insert_2>aaa!bbb!ccc!</insert_2>
19         <delete>Text &lt;i&gt;--- &lt;/i&gt; --- &lt;i&gt;noch mehr Text --- &lt;/i&gt;&lt;/delete&gt;
20             <insert_2>aaa!bbb!ccc!</insert_2>
21        <list_1 sep_count="1">11-22-33-44-55</list_1>
22        <list_2 sep_count="2">3-4-5-6-7-8-9-10</list_2>
23        <list_3 sep_count="2">10/09/2007</list_3>
24     </test>
25     <test id="3">
26         <replace>Hühnerbrühe</replace>
27         <replace_2>bändel</replace_2>
28             <insert_1>!abc!efgh!ijk!</insert_1>
29         <insert_2>aaa!bbb!ccc!</insert_2>
30         <delete>Text &lt;i&gt;--- &lt;/i&gt; --- &lt;i&gt;noch mehr Text --- &lt;/i&gt;&lt;/delete&gt;
31             <insert_2>aaa!bbb!ccc!</insert_2>
32        <list_1 sep_count="1">1-2-3-4-5</list_1>
33        <list_2 sep_count="2">1-2-3-4-5</list_2>
34        <list_3 sep_count="2">10/09/2007</list_3>
35     </test>
36     <test id="4">
37         <replace>Hühnerbrühe</replace>
38         <replace_2>bändel</replace_2>
39             <insert_1>!abc!efgh!ijk!</insert_1>
40         <insert_2>aaa!bbb!ccc!</insert_2>
41         <delete>Text &lt;i&gt;--- &lt;/i&gt; --- &lt;i&gt;noch mehr Text --- &lt;/i&gt;&lt;/delete&gt;
42             <insert_2>aaa!bbb!ccc!</insert_2>
43        <list_1 sep_count="1">1-2-3-4-5</list_1>
44        <list_2 sep_count="2">1-2-3-4-5</list_2>
45     </test>
46 </tests>

Listing 16: New Patternmatcher

1 <xs:def:stylesheet version="2.0" xmlns:xs="http://www.w3.org/1999/XSL/Transform" xmlns:xm...
<xs:variable name="Empty" select="'E'"/>
<xs:variable name="Rest" select="'Xs'"/>

<xsl:choose>
  <xs:when test="(empty($pattern) = true()) and (empty($substrseq) = true())">
    <xs:value-of select="true()"/>
  </xs:when>
  <xs:when test="empty($pattern) = true()">
    <xs:value-of select="false()"/>
  </xs:when>
  <xs:when test="empty($substrseq) = true()">
    <xs:value-of select="true()"/>
  </xs:when>
  <xs:choose>
    <xs:when test="($pattern[1] = $Empty) or ($pattern[1] = $Rest)">
      <xs:value-of select="true()"/>
    </xs:when>
    <xs:otherwise>
      <xs:value-of select="false()"/>
    </xs:otherwise>
  </xs:choose>
  <xs:when test="($pattern[1] = $Elem)">
    <xs:value-of select="igor:match-pattern(subsequence($pattern, 2), subsequence($substrseq, 2))"/>
  </xs:when>
  <xs:when test="($pattern[1] = $Rest)">
    <xs:value-of select="true()"/>
  </xs:when>
  <xs:when test="($pattern[1] = $substrseq[1])">
    <xs:value-of select="igor:match-pattern(subsequence($pattern, 2), subsequence($substrseq, 2))"/>
  </xs:when>
  <xs:otherwise>
    <xs:value-of select="false()"/>
  </xs:otherwise>
</xs:choose>

<xsl:function name="igor:separate" as="xs:string *">
  <xs:param name="toSeparate" as="xs:string *"/>  
  <xs:param name="separators" as="xs:string *"/>
  <xs:choose>
    <xs:when test="(count($separators) = 0)"/>
    <xs:when test="(count($separators) = 1)"/>
    <xs:otherwise>
      <xs:sequence select="igor:separateOne($separators[1], $toSeparate)"/>
    </xs:otherwise>
  </xs:choose>

  <xs:choose>
    <xs:otherwise>
      <xs:variable name="partialResult" select="igor:separate($toSeparate, subsequence($separators, 2))" as="xs:string *"/>
      <xs:sequence select="igor:separateOne($separators[1], $partialResult)"/>
    </xs:otherwise>
  </xs:choose>
</xs:function>

<xsl:choose>
  <xs:for-each select="$toSeparate">
    <xs:variable name="current" select="." as="xs:string *"/>
    <xs:variable name="tokens" select="tokenize($current, $separator)" as="xs:string *"/>
  </xs:for-each>
</xs:choose>

<xsl:if test="starts-with($current, $separator)">
  <xs:write back first token/>
  <xs:sequence select="$tokens[1]"/>
</xs:if>

<xsl:for-each select="subsequence($tokens, 2)">
  <xs:write back separator and token/>
</xs:for-each>

<xsl:sequence select="$separator"/>
<xsl:sequence select="."/>
</xsl:for-each>
<xsl:if test="ends-with($current,$separator) = true()">
<!-- current ended with separator, so write it back -->
<xsl:sequence select="$separator"/>
</xsl:if>
</xsl:for-each>
</xsl:function>
</xsl:stylesheet>