Proving the validity and accessibility of dynamic web pages

Proving the Validity and Accessibility of Dynamic Web-Pages

Dr R G Stone
Department of Computer Science
Loughborough University
England
44 (0)1509 222686
R.G.Stone@lboro.ac.uk

J Dhiensa
Department of Computer Science
Loughborough University
England
44 (0)1509 228222
J.Dhiensa@lboro.ac.uk

ABSTRACT
If a static web-page is checked for accessibility and passes then all is well. However checking the accessibility of the output from a dynamic (scripted) web-page is like testing a program to find errors. However many times a test succeeds it is always possible that the program will produce bad output next time. What is needed is something closer to a proof of correctness. This paper describes a first attempt to provide a proof of validity for dynamic web-pages which can be extended to a proof of accessibility.

Categories and Subject Descriptors

H.3.5 Online Information Services *Web-based services*

D.2.4 Software/Program Verification *Correctness proofs*

F.3.1 Specifying and Verifying and Reasoning about Programs *Mechanical verification*

General Terms
Reliability, Standardization, Languages, Verification.

Keywords
Proof, Validity, Accessibility, Dynamic Web-Pages

1. INTRODUCTION
Legal changes have placed a greater emphasis on web accessibility recently but authors have been concerned with the issue for some time [10]. There are accessibility guidelines [1] and there are several tools [2,4,7] available to check that an individual page or a whole site conforms to the standards. The checking of accessibility of static web-pages is arguably a routine process split between automated testing and human inspection. However the general trend in web-sites is for more and more pages to contain server-side scripting elements which are used to make the pages 'dynamic'. At present when an automated accessibility checker checks the output from a scripted page it is in fact only checking one possible output from the script. Even if this page is identified as accessible, the very next visit to the same page could potentially result in output which is not accessible.

Thus a method is sought by which a scripted page can be automatically analysed and the statement made that all possible output from the script is accessible.

Macromedia, the company responsible for the web document authoring package Dreamweaver, has put considerable effort into providing reminders and tools to help the web-page author [8]. It is perhaps all the more important to check scripted web-pages for accessibility because it is at the point at which the the author turns to hand-writing lines of script code that the support offered by a tool like Dreamweaver is at its weakest.

2. A RELATED PROBLEM
The problem outlined above is not the first problem that needs to be solved with scripted (dynamic) web-pages. A problem that arises earlier is that of validating a scripted web-page. That is to say that when a scripted web-page is intended to produce output using a recognised, tagged, mark-up language, the initial problem is to ensure that the script always produces syntactically-correct mark-up. A potential solution to this problem has been found and the technique used offers an approach to the accessibility problem as well.

3. SCRIPTED WEB-PAGES
When using a server-side scripting language like PHP [9] or ASP [3], programming statements are embedded inside tagged mark-up. The programming statements are recognizable by being enclosed in some kind of brackets (<?php … ?> or simply <? … ?> for PHP). Among the statements available in the chosen scripting language there will be one, often written *echo* or *print*, which can be used to generate extra, tagged mark-up. Here is an example using PHP:

```
<html>
<body>
<?php
   echo "<p>A first paragraph</p>";
   echo "<p>A second paragraph</p>";
```

```
?>
</body>
</html>
```

Furthermore the scripting language will contain conditional statements which can deliver alternatives such as

```
if(…)
        echo "<p>Either this</p>";
else
        echo "<p>Or this</p>";
```

and there will be iterative statements such as *for* loops or *while* loops to produce repetitive output, for example

```
while($i<10){
        echo "<p>Paragraph $i</p>";
        $i++;
}
```

Consider the following scripted page which makes a check on whether its user is an administrator (who may have special privileges) and ordinary user who may be accessing the pages during office hours or not.

```
<html><body>
<?php
if(is_admin_user())
        echo "<p><img src='A.jpg' … /></p>";
else if(is_in_office_hours())
        echo "<p><img src='B.jpg' … /></p>";
else
        echo "<p><img src='C.jpg' … /></p>";
?>
</body></html>
```

There are three strands of output coded into the script, only one of which will ever appear in any particular run depending on the status of the user and the day/time that they made the request. A conventional validity or accessibility test on this script would be to run it once and check the output. So if a normal user ran the test in office hours the 'B' output would be obtained and, for example, the <img src='B.jpg' …> tag could be checked to see if it had an 'alt' attribute. In order to even see the other <img…> tags in the output the script would have to be run twice more with different combinations of user and time of run.

We now develop a method of obtaining a generalized output from a script like this which contains all possibilities and go on to show that a validity or accessibility tester can be relatively easily extended to check the extended output. This allows the possibility of proving the validity or accessibility of a scripted web-page.

The basis of the idea is to try and match the structure of the script statements with the structures imposed on the script output by the document type being produced. So there is a link between a while-loop producing a sequence of paragraphs and a part of the definition of the output document type that requires or allows a sequence of paragraphs. There is a link between a conditional statement and a part of the output document structure that is optional or has alternatives.

4. CAPTURING GENERALISED OUTPUT FROM A SCRIPT

The solution to both of the identified problems starts with having a notation to capture not just a single instance of the output from a script, but some kind of generalised output expression which represents every possible output. A commonly used notation that can represent sequencing, alternation and iteration is that of regular expressions

xy	x followed by y
(x)?	an optional x
(x|y)	x or y
(x)*	zero, one or more repetitions of x
(x)+	one or more repetitions of x

This notation can be used to capture the generalised output from a script. For example, relating to the earlier examples we have

```
( <p>Either this</p> | <p>Or this</p> )

( <p>Paragraph $i</p> )*
```

At the expense of brevity, the notation can be rendered in ‘tag’ style which may be more suited to the task at hand

(x)*	could become	<LIST0> x <LIST0>
(x)+	could become	<LIST1> x <LIST1>

In the following these invented tags, representing regular expression notation, will be referred to as meta-tags.

5. SOLVING THE VALIDATION PROBLEM IN PRINCIPLE

The validation problem can be stated as checking that a tagged file matches the structure defined by a Document Type Declaration (DTD [6]). The DTD uses the regular expression notation to define the content of tags. To take one of the simplest DTDs in common use, the DTD for WML [11], we find the following top-level element definitions

```
<!ELEMENT wml ( head?, template?, card+ )>
<!ELEMENT card (onevent*, timer?, (do | p)*)>
```

Simplifying these elements a little (and ignoring attribute definitions completely), we consider

```
<!ELEMENT wml ( card+ )>
<!ELEMENT card ( p* )>
```

This is read as stating that a wml element contains at least one card element which in turn contains zero or more paragraph elements. So current validators should accept

```
<wml>  <card>  <p>1</p>
               <p>2</p>
       </card>
</wml>
```

and we propose that they should now have to deal with, and accept in addition, tag structures which include meta-tags such as

```
<wml>  <LIST1> <card>  <LIST0> <p>$i</p>
                       </LIST0>
               </card>
       </LIST1>
</wml>
```

As an indication of the kind of script that might be rejected as invalid, consider the situation where the DTD requires tag . If the PHP code is

```
if (…) echo "<p>…</p>"; else echo "<table>…</table>";
```

we obtain

```
<CHOICES>
  <CHOICE><p>…</p></CHOICE>
  <CHOICE><table>…</table></CHOICE>
</CHOICES>
```

which can be represented as (p | table) where p is required which is unacceptable.

As an indication of the kind of script which might produce a warning report rather than an error, consider the situation where the DTD requires p+, i.e. a sequence of one or more p tags. If the PHP code is

```
while(…){ echo "<p>…</p>"; }
```

we obtain

```
<LIST0><p>…</p></LIST0>
```

which can be represented as p*. Since p* allows the possibility of no tag , validity in this case can not proved by a simple structural argument. A much more sophisticated system, capable of analysing the loop expression and the preceding context, would be needed to argue that the loop expression is never false on the first iteration, as for example in

```
$i=0; while($i<10){…}
```

If the PHP code had been written using a repeat loop as

```
do{ echo "<p>…</p>"; }while(...);
```

then we would have obtained <LIST1> … </LIST1> which is equivalent to p+ and no validity problems would be raised.

By considering slightly more complex scripts e.g.

```
echo "<p>...</p>";
while(…){ echo "<p>…</p>"; }
echo "<p>...</p>";
```

it is clear that there is the need to apply a little algebra to rewrite various combinations that might occur in sequence. For example any of pp*, p*p, pp+ or p+p can be rewritten as p+ in order to achieve the proof of validation.

6. THE SOLUTION OF THE VALIDATION PROBLEM IN PRACTICE

The solution to the validation problem is proposed in two parts. The first step is to build or alter a PHP parser/interpreter so that it generates the meta-tagged generalised output of the script rather than one specific instance. The second step is to build or alter a validation application so that it accepts and validates meta-tagged expressions. This has been trialled using the YACC/LEX [12] parser generator tools associated with Unix to build simple versions of the applications required.

Experience suggests that the tags that appear in the output from scripted pages are mostly obtained by relatively simple code. Thus in the trial version, tags to be output via echo statements are expected to be visible as constants, not hidden by string operators or function calls. So for example we expect

```
echo "<p>";
```

not

```
echo "<"."p".">";
```

where "." is the PHP string concatenation operator. Accordingly a parser was built for a simplified subset of PHP. Compiling actions were added to implement the echo statement. Actions were also added to produce the relevant meta-tokens, For example a while-

loop generates <LIST0>...</LIST0>, a repeat loop generates <LIST1>...</LIST1> and conditionals generate <CHOICES>...</CHOICES>. Extended parsers were also built for XHTML and WML directly from the relevant DTDs. These parsers enforce such rules as <!ELEMENT card (p*)> by allowing p* to be made up of individually tagged p elements and meta-tagged p elements.

7. THE SOLUTION TO THE ACCESSIBILITY PROBLEM

It seems to us that the accessibility issue for dynamic pages can be tackled in exactly the same way as the validity problem. The same first application is used to produce a generalised output expression for the script. The second step requires an accessibility application to be built or altered so that it works on the generalised output rather than one specific instance.

Where an accessibility application normally makes a check on each element in a sequence (e.g. in a sequence of paragraph elements) it would now need to be able in addition to check a condensed sequence of paragraph elements expressed as

```
<LIST0> <p> … </p> </LIST0>
```

and indeed a mixture of these, such as

```
<p> 1 </p>
<LIST0> <p> … </p> </LIST0>
<p> 10 </p>
```

Where an accessibility application normally made a check on an individual element (e.g. a paragraph element) it would now need to be able to check through all the alternatives expressed as

```
<CHOICES>
   <CHOICE> <p> 1 </p> </CHOICE>
   <CHOICE> <p> 2 </p> </CHOICE>
</CHOICES>
```

Since an accessibility checker is built to recognize tag structures [5] it is thought that this would be a relatively simple modification to make.

Consider the consequences of applying this extended accessibility check to the

```
if(...) {A} else if(...) {B} else {C}
```

example of section 3. Because the accessibility check is testing all the possible contributions to the output introduced by a conditional and not just one as is the case normally, confidence can be gained that on any execution of the script an accessible document is produced.

8. CONCLUSION

By adapting a solution to the problem of validating a dynamic web-page, we have shown how a dynamic web-page can be checked for accessibility. The strength of this approach is that a scripted page is being checked once-and-for-all, producing a guarantee that all runs of the script will produce valid and accessible output. This is in contrast to the present situation where individual instances of script output are checked.

A trial version of the validation checker suitable for scripts that have been written in a subset of PHP has proved encouraging. The biggest obstacle to scalability is the ability to obtain the generalised regular expression output corresponding to the script if the full version of the scripting language is being used. It may be that, in order to obtain the validation proof for a script, programmers may be content to restrict themselves to a subset of the scripting language.

9. REFERENCES

[1] Accessibility guidelines http://www.w3.org/WAI/Resources/#gl

[2] APROMPT http://aprompt.snow.utoronto.ca/

[3] ASP http://msdn.microsoft.com/library/default.asp?url=/nhp/default.asp?contentid=28000522

[4] Bobby http://bobby.watchfire.com/bobby/html/en/index.jsp

[5] Cooper, M. & Rejmer, P. Case study: localization of an accessibility evaluation, http://citeseer.ist.psu.edu/571304.html

[6] DTD http://www.w3schools.com/dtd/default.asp

[7] LIFT, UsableNet http://www.usablenet.com

[8] Macromedia Accessibility http://www.macromedia.com/macromedia/accessibility

[9] PHP http://www.php.net/

[10] Sierkowski, B. Achieving web accessibility, Proceedings of the 30th annual ACM conference on User services, 2002 http://portal.acm.org/citation.cfm?id=588725

[11] WML http://www.w3schools.com/wap/wml_dtd.asp

[12] YACC/LEX http://dinosaur.compilertools.net/