GUARANTEEING STRONG (X)HTML COMPLIANCE FOR DYNAMIC WEB APPLICATIONS

Paul G. Talaga and Steve J. Chapin
Syracuse University, Syracuse, NY, U.S.A.

Keywords: W3C compliance, Web development, Haskell.

Abstract: We report on the embedding of a domain specific language, (X)HTML, into Haskell and demonstrate how this superficial context-free language can be represented and rendered to guarantee World Wide Web Consortium (W3C) compliance. Compliance of web content is important for the health of the Internet, accessibility, visibility, and reliable search. While tools exist to verify web content is compliant according to the W3C, few systems guarantee that all dynamically produced content is compliant. We present *CH-(X)HTML*, a library for generating compliant (X)HTML content for all dynamic content by using Haskell to encode the non-trivial syntax of (X)HTML set forth by the W3C. Any compliant document can be represented with this library, while a compilation or run-time error will occur if non-compliant markup is attempted. To demonstrate our library we present examples and performance measurements.

1 INTRODUCTION

Conformity of web content to the World Wide Web Consortium's (W3C) standards is a goal every web developer should aspire to meet. Conformity leads to *increased visibility* as more browsers can render the markup consistently, *increased accessibility* for disabled users using non-typical browsing styles(Chisholm, 1999), *more reliable Internet search* by presenting search engines with consistent page structures(Davies, 2005), and in some cases *compliance with legal requirements* (Brewer and Henry, 2006; Moss, 2010; Wittersheim, 2006; dda, 2010).

Unfortunately the majority of web content is non-compliant, with one study finding 95% of pages online are not valid(Chen et al., 2005). Not surprisingly, the majority of web frameworks do not guarantee generated content is compliant. Popular internet browsers perpetuate the problem by creatively parsing and rendering invalid code in an attempt to retain users.

While tools exist to check validity of static content, few systems exist that claim strong validity of *all* produced content. With dynamic web applications, it is harder to guarantee validity due to the dynamic nature of their outputs. Assuring compliance for specific inputs is possible, but proving compliance for all inputs is analogous to proof by example. Web frameworks using Model-View-Controller design practices provide some assurances based on compliant templates, but it remains easy for an unknowing developer or user input to break this compliance. Such deficiencies in frameworks can have security consequences as well(Hansen, 2009). Rather than make it easy for developers to produce invalid content, frameworks should make it impossible to be non-compliant.

1.1 Contributions

We present *CH-(X)HTML*, a Haskell library for building (X)HTML content with strong W3C compliance for all outputs. By using Haskell's recursive types, multiple parameter and functional dependency of type classes, web content is built by separating structure from content in a typed tree data structure way, much like the underlying (X)HTML. The resulting structure can be stored, manipulated, or serialized to a standard W3C compliant textual representation.

We identify five traits common to all W3C (X)HTML specifications which must be met for a document to be compliant and show how *CH-(X)HTML* enforces four of these at compile-time, with the fifth optionally at run-time.

The remainder of the paper is structured as follows. We analyze and categorize commonalities between different W3C (X)HTML specifications in Section 2, identifying requirements a W3C compliant producing system must possess. Section 3 provides an overview of *CH-(X)HTML* and discusses how it is

G. Talaga P. and J. Chapin S..
GUARANTEEING STRONG (X)HTML COMPLIANCE FOR DYNAMIC WEB APPLICATIONS.
DOI: 10.5220/0003348800710079
In *Proceedings of the 7th International Conference on Web Information Systems and Technologies* (WEBIST-2011), pages 71-79
ISBN: 978-989-8425-51-5

able to enforce the W3C specifications while being easy to use. Sample code is provided showing the use of the library, followed by a performance evaluation in Section 4. Related work and our conclusion are in Sections 5 and 6 respectively.

2 W3C COMPLIANCE

The W3C has set forth numerous variants of specifications of HTML and XHTML, with more on the way in the form of HTML5. Examples include HTML 3.2, HTML 4.01 Strict, and XHTML 1.00 Transitional. While conformance to a specific document type definition (DTD) is our goal, identifying commonalities will assure easy conversion to any HTML DTD. For example, the difference between HTML 4.01 Strict and HTML 4.01 Transitional is merely the allowance of certain tags. Likewise, HTML 4.01 Frameset and XHTML 1.00 Frameset differ in their document type: SGML and XML respectively(Group, 2002).

We have identified five classes of common requirements between different (X)HTML DTDs based on Thiemann's work(Thiemann, 2001). A system capable of supporting all requirement classes should be able to include support for all requirements in any of the W3C specifications. These classes include the following:

Well-formed. An (X)HTML document is well-formed if all tags have appropriate starting and ending characters, as well as an ending tag when needed. All attributes have the form `attribute="value"` inside a tag. All characters should be in the correct context. For example, all markup characters should only be used for markup including `<,>,&,"`.

Tag-conforming. An (X)HTML document is tag-conforming if all tags are defined and valid within that DTD. No browser specific tags should be used.

Attribute-conforming. An (X)HTML document is attribute-conforming if all attributes names are allowed for that specific tag. For example, the `p` tag can not contain an `href` attribute. Similarly, the value type of every attribute matches its DTD description. Required attributes are also provided.

Inclusion & Exclusion. An (X)HTML document obeys inclusion & exclusion if the nesting of all tags follow the specific DTD. For example, in HTML 4.01 no `a` tag can be a descendant of another `a` tag. Similarly, the `tr` tag requires a `td` tag to be its child. While SGML, of which HTML is a member, allows deep nesting rules, XML does not(Group, 2002). XML can specify what children are allowed, but not grandchildren or beyond. Thus, the XHTML 1.0 specification recommends the inclusion & exclusion of tags, but can not require it. We feel that since XHTML is fully based on HTML this requirement is important and should be enforced. In support, the W3C online validator marks inclusion & exclusion problems in XHTML as errors. The draft HTML5 specification broadens nesting rules by restricting *groups* of tags to be children(htm, 2010). For example, an `a` tag in HTML5 must not contain any *interactive content*, of which 15 tags are members.

Tag Ordering. An (X)HTML document obeys tag ordering if sibling tags are ordered as described in their DTD. As an example, the `head` tag must precede the `body` tag as children of the `html` tag.

3 CH-(X)HTML

Our system is built as an embedded domain-specific language, implemented in Haskell, capable of embodying the requirements set forth by the W3C. The use of a strongly typed language guarantees strong compliance of the application at *compile* time, while allowing easy representation of the embedded language. Any strongly typed language could be used for such a system, but Haskell's multiple parameter and functional dependency type classes cleans up the syntax for the developer.

CH-(X)HTML is available for download[1] or on Hackage[2]. XHtml 1.0 Strict(Group, 2002), Transitional, and Frameset are currently supported at this time.

3.1 Implementation Overview

CH-(X)HTML's design is outlined through a series of refinements guaranteeing each of the five W3C specification classes described above. Code examples are meant to convey design methods, not produce fully correct HTML.

Well-formed & Tag Conformance. At its core, *CH-(X)HTML* uses ordinary Haskell types to implement a recursively defined tree data structure representing the (X)HTML document. Each node in the tree represents a tag, with inner tags stored as a list

[1] http://fuzzpault.com/chxhtml

[2] http://hackage.haskell.org/package/CHXHtml

of children. Depending on the tag, the node may have none, or a variable number of children. Tag attributes are stored with each node. Character data is inserted using a `pcdata` constructor. An example of this scheme is given:

```
data Ent = Html Attributes [Ent] |
           Body Attributes [Ent] |
           P Attributes [Ent] |
           A Attributes [Ent] |
           Br Attributes |
           Cdata String | ...
data Attributes = [String]
render :: Ent -> String
```

Only defined tags for a specific DTD exist as constructors, thus forcing tag-conformance. When the data structure has been constructed and is ready to be serialized, a recursive function `render` traverses the structure, returning a string containing tags and properly formatted attributes and values. All character data (CDATA) is HTML escaped before rendering preventing embedding of HTML markup. Separating content from structure, along with HTML escaping, forces all produced content to be well-formed and tag-conforming.

Attribute Conformance. To limit allowed attributes and their types, each tag is given a custom attribute type allowing only valid attributes to be used. If an attribute is set more than once the `render` function will use only the last, guaranteeing a unique value for each attribute. Required attributes can either be automatically inserted at run-time to guarantee compliance, or rendered as-is with a warning if attributes are lacking. This run-time compliance issue is discussed in Section 3.2.

The example below implements attribute conformance described above.

```
data Ent = Html [Att_html] [Ent] |
           Body [Att_body] [Ent] |
           P [Att_p] [Ent] |
           A [Att_a] [Ent] |
           Br [Att_b] |
           Cdata String | ...
--
data Att_html = Lang_html String |
                Dir_html String  | ...
data Att_body = Lang_body String |
                Dir_body String  |
                Onload_body String | ...
...
render :: Ent -> String
```

Inclusion & Exclusion Conformance. Thus far any tag can be a child of any other. For inclusion & exclusion conformance we use new data types representing the context of those tags. Each DTD describes allowed children for each tag, required tags, as well as limits on all descendants. For example, Table 1 describes the inclusion & exclusion rules for some tags in HTML 4.01 Strict. A + in the required children column signifies at least one child must exist.

By using unique types for each tag only allowed children tags can be inserted. To enforce descendant rules, a new set of types are used which lack the forbidden tag. Thus, rather than one constructor specifying a tag, a set of constructors may, each valid in a different context. For example the following code correctly prohibits nesting of the `a` tag by effectively duplicating the `Ent3` type but lacking the `a`.

```
data Ent = Html Att_html [Ent2]
data Ent2 = Body Att_body [Ent3]
data Ent3 = A3 Att_p [Ent_no_a] |
            P3 Att_p [Ent3] |
            Br3 Att_b |
            Cdata3 String | ...
data Ent_no_a = P_no_a Att_p [Ent_no_a] |
            Br_no_a Att_b |
            Cdata_no_a String | ...
- Attributes same as above
render :: Ent -> String
```

In practice, preventing a deep descendant results in duplication of nearly all types. Had `Ent3` allowed some other child other than itself, then it too must be duplicated with a `_no_a` version, and so on. A combinatorial explosion could prove this approach unfeasible, but our analysis has shown otherwise. For example, by enumerating all possible valid nesting situations in HTML 4.0 Strict, a total of 45 groups of tags were needed to properly limit allowed children while preventing invalid descendant situations.

Assuring the required children tags exist is checked at run-time which is discussed in Section 3.2.

Tag Order Conformance. To validate or warn against tag-order conformance errors, a run-time checker `childErrors` can be used. Section 3.2 discusses the issues and trade offs involved with such a run-time system.

3.2 Complete Compile-time Compliance vs. Usability

A trade-off exists between complete compile-time compliance and usability with regard to tag ordering,

Table 1: Inclusion & Exclusion Examples in HTML 4.01.

Tag	Required Children	Allowed Children	Disallowed Descendants
html	head,body	head,body	
body	+	p,div,...	
p		a,br,cdata,...	
a		br,cdata,...	a
tr	+	th,td	
form	+	p,div,...	form
...	...	...	...

required children tags, and required attributes. A library's interface should be obvious, allowing existing HTML knowledge to be used easily in a new context. For usability we've decided to have four forms of tag constructors: two with children, and two without, detailed in Section 3.4. Children tags are specified as a regular Haskell list for those tags allowing children. Similarly, tag attributes are described in list form as well. This allows the developer to easily write markup and apply any list manipulation function to tags or attributes. Unfortunately there is no way to restrict the list elements or their order at compile-time. A run-time checker `childErrors`can scan the completed document for tag ordering, required children, or required attribute errors if needed.

The alternative would be to specify children or attributes as a tuple. Tuples allow different type elements, but all must be provided. List manipulation would not be possible, nor would all tags have a standard interface. Some tags may take a 3-tuple, while others may take a list, or some other combination thereof. Burdening the developer with such complexities and limitations was deemed too harsh given the complete compile-time guarantee benefit. The HaXml project contains a DtdToHaskell utility which enforces ordering in this manner using tuples(Wallace and Runciman, 1999).

To strengthen the compile-time guarantees with regard to W3C compliance, the `fix_page` function can attempt to manipulate the document to reach full W3C compliance. It will return a repaired document, a synopsis of changes made, and if it was able to reach full W3C compliance. Currently required attribute and tag errors are repaired, with tag ordering errors still in development.

3.3 Cleanup

Writing (X)HTML content using complex constructors described in Section 3.1 becomes unwieldy quickly. By using multi-parameter type classes and functional dependencies we can hide this complexity while still retaining the compile-time guarantees. We construct a type class per tag such that a function correctly returns a constructor of the correct type based on context. The following example shows the type class for specifying the `p` tag.

```
class C_P a b | a -> b where
    p :: [Att_p] -> [b] -> a
instance C_P Ent3 Ent3 where
    p at r = P_1 at r
instance C_P Ent_no_a Ent_no_a where
    p at r = P_2 at r
```

The class instance used is determined by the context the function is called in, which determines what type children it may have provided by the functional dependency of classes. Thus, as long as the root of the recursive structure has a concrete type all children will be uniquely defined. Nesting errors manifest themselves as compile-time class instance type errors.

Attribute specification is handled in a similar way. Thus, during development, only the tag or attribute names must be specified, all complex constructor selection is done by the functional type classes.

3.4 Library Usage

Building an (X)HTML document is done by constructing the recursive data structure and serializing it using the `render` or `render_bs` functions. Content can be served to the web with any number of Haskell web servers such as HAppS(hap, 2010), Happstack(Elder and Shaw, 2010), MoHWS(Marlow and Bringert, 2010), turbinado(Kemp, 2010), SNAP(Collins et al., 2010), or via executable with CGI(cgi, 2010)(Bringert, 2010) or FastCGI(Saccoccio et al., 2010)(Bringert and Lemmih, 2010) Haskell bindings. *CH-(X)HTML* can be used anywhere a `String` type containing (X)HTML is needed in Haskell. For speed and efficiency the `render_bs` function returns a lazy ByteString representation suitable for CGI bindings.

All HTML tags are represented in lower case with an underscore _ before or after the tag text. Before assigns no attributes, while after allows a list of attributes. Tags which allow children then take a list of children.

Table 2: W3C Conformance Performance.

Library	Well-formed	Tag-conf.	Attribute-conf.	Inclusion/Exclusion	Tag-order-conf.
CH-(X)HTML	●	●	◐	◐	○
CH-(X)HTML w/runtime	●	●	●	●	●
Text.Html	●	●	◐	○	○
Text.XHtml	●	●	◐	○	○
BlazeHtml	●	●	◐	○	○
Hamlet	●	●	◐	○	○
Text.HTML.Light	●	●	◐	○	○
PHP	○	○	○	○	○

Attributes are represented in lower case as well, but suffixed with `_att`. This assures no namespace conflicts. Assigning an attribute which does not belong results in a compile-time class instance error.

Validating child ordering is done using the `childErrors` function which takes any node as an argument. A list of errors will be returned, if any, along with the ordering specification in the DTD which failed.

Figure 1 exhibits the obligatory Hello World page where `result` holds the resulting serialized HTML as a string.

For a more through description of the *CH-(X)HTML*'s usage see the `demo.hs` included with the library source.

4 LIBRARY PERFORMANCE

To gauge our library's performance against similar dynamic HTML creation systems, we compared it to six other libraries: Text.Html, Text.XHtml, BlazeHtml(Meier and der Jeugt, 2010), Hamlet(Snoyman, 2010), Text.HTML.Light, PHP. The first five are combinator libraries in Haskell used for building HTML content while PHP is a popular web scripting language. *CH-(X)HTML* was tested twice, once with run-time list checking, and once without.

Table 2 shows each's W3C compliance guarantees for all produced content with regard to the five W3C areas of compliance. Half-filled circles indicate partial compliance. For example, *CH-(X)HTML*without run-time list checking is not fully attribute conforming due to possible omission of required attributes. Other libraries fare even worse by allowing any attribute to be used with any tag.

It is clear *CH-(X)HTML* without run-time list checking offers stronger compliance than any library tested, with full compliance attainable with run-time checking and repairing enabled.

To test rendering performance, an XHTML 1.0 Strict 'bigtable' document was created containing a table with 1000 rows, each with the integers from 1 to 10 in each rowś column. `Head` and `title` tags were added for W3C compliance with no other content on the page. The table is generated dynamically leading to very short page generating code. The final page consisted of 11,005 tags and about 121kB total size.

To rule out web server performance each library is timed until the content is prepared in memory ready to be sent. The `criterion` library in Haskell is used to benchmark in this way while the cpu time was measured for PHP. Testing was done on a Fedora Core 11 server with an AMD 2.3Ghz Athlon 64 X2 processor with 2GB of RAM.

Speed results are shown in Fig 2. Hamlet was the fastest, with BlazeHtml and PHP slightly slower at 10ms. *CH-(X)HTML* without run-time tag ordering validation beats out Text.Html and Text.XHtml showing *CH-(X)HTML* is able to compete with generic HTML productions systems while enforcing four of the five compliance classes. Adding tag order validation adds significant time, requiring 115(ms) to render bigtable in our current implementation. This is due to an external regular expression library being called for each tag; for this case 11,005 times. Further benchmarking with different size tables showed a perfectly linear relation between tag count and page rendering time for *CH-(X)HTML* with tag ordering validation.

To gauge real-world performance for *CH-(X)HTML* with tag order validation, a sample of 31,000 random HTML pages were downloaded from the web and analyzed. The average tag count per page was 801, with the 50th percentile having 330 tags. Average page size for the sample was 60kB, with the 50th percentile being 23kB. While bigtable may stress a dynamic web content generator, it does not represent a typical web page with respect to tag count or size. A new average page benchmark was constructed by using 30 rows from the bigtable benchmark and adding plain text in a `p` tag resulting in 335 tags and about 60kB total size. Figure 3 shows the vastly different results. PHP now leads, closely followed by *CH-(X)HTML* without tag order checking. Even with tag order checking *CH-(X)HTML* does quite well beating out three other libraries.

```
page name = _html [_head [_title [pcdata "Hello " ++ name]],
                _body [_h1 [pcdata "Hello " ++ name ++ "!"],
                   _hr,
                       _p [pcdata "Hello " ++ name ++ "!"],
                      ]
                   ]
result :: String
result = render (page "World")
```

Figure 1: Hello World implementation in *CH-(X)HTML*.

While speed is not our goal, *CH-(X)HTML* preforms on par with similar dynamic HTML production systems for most pages while providing more guarantees on compliant output. For pages containing atypically large amounts of tags, additional speed can be gained by not running run-time list checking.

5 RELATED WORK

There exist two areas related to our work: XML creation and manipulation, and general HTML production. XHTML has now joined these two areas.

Numerous projects have embedded XML into other languages and allowed for its manipulation. Web content creation is not their main goal, but rather generic XML with custom schema.

The mainstream language Java has JAXB(jax, 2010), which can create a set of Java classes based on an XML schema, as well as the inverse. Data can marshaled in either direction easily allowing dynamic schema to be accessed in Java. If used for XHTML production, inclusion/exclusion errors could still be present as well as possible invalid characters. XMLBeans is a similar tool(xml, 2010).

The automatic generation of Haskell types from DTD's or schema are covered in the HaXml project: a set of Haskell utilities for parsing, filtering, transforming, and generating XML(Wallace and Runciman, 1999). Their DtdToHaskell utility produces a Haskell source file containing datatypes conforming to the provided XML DTD. DtdToHaskell's generic XML to Haskell type production system works with XHTML DTDs and even guarantees tag-ordering, but at the price of usability. Every element (tag) attribute must be specified even if not used due to their record syntax implementation. Elements (tags) requiring an ordered or specific number of children aren't specified using list syntax like most tags, but n-tuples, requiring one to reference either the DTD or datatypes to resolve compilation errors. Our specialized solution of XHTML uses lists for all children, simplifying the syntax. Child ordering can be validated at run-time if needed. XML's lack of tag inclusion/exclusion restriction prevents HaXml from enforcing it as well. *CH-(X)HTML* is generated with a similar tool to DtdToHaskell from a raw DTD, but is able to interpret hybrid DTDs containing nesting restrictions.

HSXML is an XML serialization library for functional languages(Kiselyov, 2010). It is part of a larger effort to parse XML into S-expressions in functional languages such as Scheme and Haskell, with HSXML preforming the reverse. S-expressions are a natural way of representing nested data with its roots in Lisp, thereby guaranteeing a well-formed and tag-conforming document. The library's current implementation can handle Inline vs Block context restrictions, but no other inclusion/exclusion or child ordering restrictions are enforced.

Constructing web content by means of a DOM-like data structure isn't new, but libraries guaranteeing near or full HTML validity are scarce. Many HTML libraries use HTML like syntax, unlike the above XML tools, allowing easy construction of pages for the developer, but with little guarantees to the validity of the output. Peter Thiemann's work on W3C compliance is the closest in the Haskell WASH/CGI suite(Thiemann, 2002b; Thiemann, 2005; Thiemann, 2001), which includes a HTML & XML content production system using types to enforce some validity. The use of element-transforming style in the library allows Haskell code to look similar to HTML while still being valid Haskell source. The author documents different classifications of validity, which our analysis in Section 2 is based on, followed by a discussion of enforcement of those classifications in his system. The Inclusion & Exclusion issue is raised and discussed briefly in his 2002 work, concluding the type class system is unable to handle inclusion & exclusion in their implementation due to the inability to handle disjunctions of types. As a result, their library does not support inclusion or exclusion with the excuse of extreme code size, difficulty in usability, and a lack of strict guidelines for inclusion & exclusion in the XHTML specification.

Further work by Thiemann explores an alternate way of dealing with the inclusion & exclusion issue in Haskell by way of proposed extensions providing

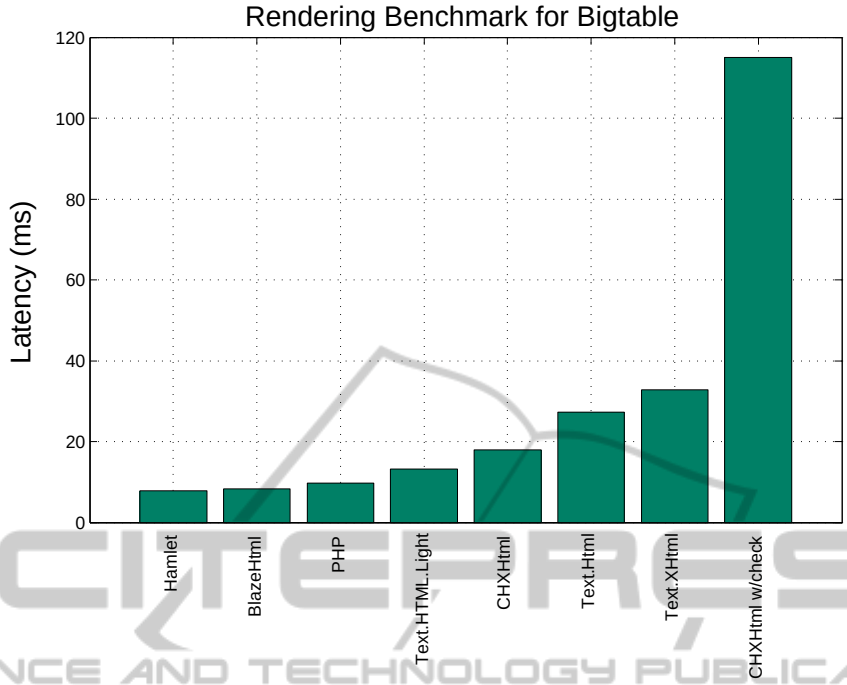

Figure 2: Rendering times for XHTML libraries for bigtable.

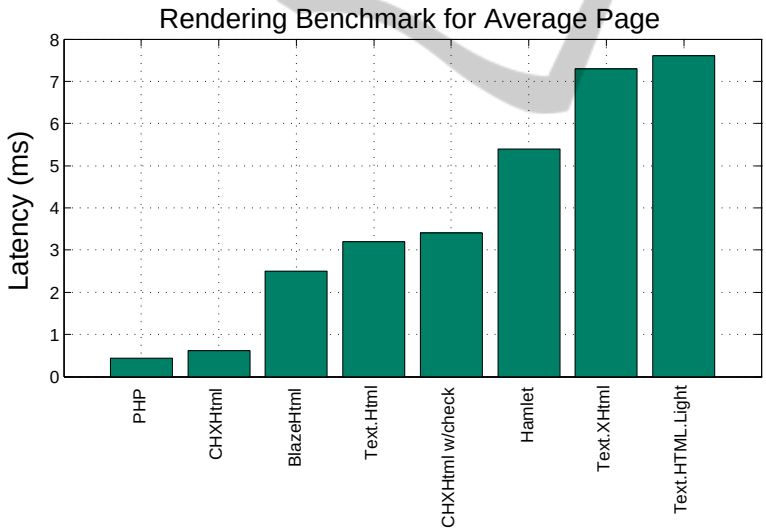

Figure 3: Rendering times for XHTML libraries for an average page.

functional logic overloading, anonymous type functions, and rank-2 polymorphism. With these they are able to accurately encode and enforce the inclusion & exclusion properties specified in the DTD(Thiemann, 2002a). A strong symmetry exists between our work and the suggested extensions. The ability to embed regular expressions on types is analogous to our generous use of recursive types and run-time child validation. While extending the type system further may lead to more enhancements, *CH-(X)HTML* can be used currently without any additional extensions. The LAML(Nørmark, 2005) package for Scheme provides a set of functions representing HTML tags and attributes capable of generating HTML programatically. Their goal is bringing abstraction and web authoring to Scheme rather than standards compliance. Their functions are easy to use and provide well-formed, tag-conforoming, and some attribute conforming content while not preventing inclusion & exclusion, or tag ordering errors.

A common Haskell HTML library is Text.Html(tex, 2010) and relative Text.XHtml used above, which uses element-transforming style to build pages. Produced content is well-formed and tag-conforming due to their structured building method and HTML escaping of text content. Any attribute can be added to any tag, thus not being attribute-conforming. All tags are of the same type and can be added in any order leading to tag ordering and inclusion/exclusion violations. Blaze-html(Meier and der Jeugt, 2010) and Hamlet(Snoyman, 2010) are similar Haskell libraries, but unfortunately they also suffer from the same lack of compliance guarantees.

XMLC for Java allows an application developer to manipulate a DOM structure obtained from parsing a HTML or XML template file(xml, 2008). Manipulation of the DOM is therefore similar to DOM manipulations in JavaScript. When all transformations are complete the DOM is serialized and sent to the user. XMLC does not restrict operations which would result in invalid content being sent to the user.

Separating structure from content in a web setting is advantageous for security as well. Robertson & Vigna(Robertson and Vigna, 2009) explore using a strongly typed system for HTML generation as well as producing SQL queries in the web application. Their goal is to increase security by preventing injection attacks targeting the ad-hoc mixing of content and structure in SQL by representing structure in a typed way and filtering inserted content. Thus, the client or SQL server's parser will not be fooled by the attempted injection attack. Our work similarly mitigates injection attacks but does not address web application vulnerabilities relating to a database.

6 CONCLUSIONS

We have shown how (X)HTML W3C compliance can be achieved by Haskell while performing on par with more mature dynamic (X)HTML production systems. We generalize the W3C (X)HTML specifications into five classes of requirements a web production system must be able to enforce to produce compliant output. The inclusion & exclusion nesting requirement of nearly all (X)HTML DTD's has proven difficult to enforce and thus ignored by web production libraries. Our (X)HTML library, *CH-(X)HTML*, is able to partially enforce four of the five classes of requirements at compile-time, including inclusion & exclusion, with full compliance attainable at run-time. Use of the library is straightforward due to multi-parameter type classes and functional dependencies allowing a coding style similar to straight (X)HTML, while guaranteeing strong compliance for all produced content.

REFERENCES

Brewer, J. and Henry, S. L. (2006). Policies relating to web accessibility. http://www.w3.org/WAI/Policy/.

Bringert, B. (2010). cgi: A library for writing cgi programs. http://hackage.haskell.org/package/cgi.

Bringert, B. and Lemmih (2010). fastcgi: A haskell library for writing fastcgi programs. http://hackage.haskell.org/package/fastcgi.

CGI (2010). The common gateway interface. http://hoohoo.ncsa.illinois.edu/cgi/.

Chen, S., Hong, D., and Shen, V. Y. (2005). An experimental study on validation problems with existing html webpages. In *International Conference on Internet Computing*, pages 373–379.

Collins, G., Beardsley, D., yu Guo, S., and Sanders, J. (2010). Snap: A haskell web framework. http://snapframework.com/.

Davies, D. (2005). W3c compliance and seo. http://www.evolt.org/w3c-compliance-and-seo.

Directgov (2010). The disability discrimination act (dda). http://www.direct.gov.uk/en/DisabledPeople/RightsAndObligations/DisabilityRights/DG_4001068.

Elder, M. and Shaw, J. (2010). Happstack. http://happstack.com/index.html.

Group, W. H. W. (2002). Xhtml 1.0: The extensible hypertext markup language (second edition). http://www.w3.org/TR/xhtml1/, http://www.w3.org/TR/xhtml1/.

Hansen, R. (2009). Xss (cross site scripting) prevention cheat sheet. http://ha.ckers.org/xss.html.

Happs (2010). Happs. http://happs.org/.

Html5 (2010). Html5. http://dev.w3.org/html5/spec/Overview.html.

Jaxb (2010). jaxb. https://jaxb.dev.java.net/.

Kemp, A. (2010). Turbinado. http://wiki.github.com/alsonkemp/turbinado.

Kiselyov, O. (2010). Hsxml: Typed sxml. http://okmij.org/ftp/Scheme/xml.html#typed-SXML.

Marlow, S. and Bringert, B. (2010). Mohws: Modular haskell web server. http://hackage.haskell.org/cgi-bin/hackage-scripts/package/mohws.

Meier, S. and der Jeugt, J. V. (2010). Blazehtml. http://jaspervdj.be/blaze/.

Moss, T. (2010). Disability discrimination act (dda) & web accessibility. http://www.webcredible.co.uk/user-friendly-resources/web-accessibility/uk-website-legal-requirements.shtml

Nørmark, K. (2005). Web programming in scheme with laml. *J. Funct. Program.*, 15(1):53–65.

Robertson, W. and Vigna, G. (2009). Static Enforcement of Web Application Integrity Through Strong Typing. In *Proceedings of the USENIX Security Symposium*, Montreal, Canada.

Saccoccio, R. et al. (2010). Fastcgi. http://www.fastcgi.com /drupal/.

Snoyman, M. (2010). Yesod web framework. http://docs.yesodweb.com/.

Text.html (2010). Text.html. http://hackage.haskell.org/ package/html

Thiemann, P. (2001). A typed representation for html and xml documents in haskell. *Journal of Functional Programming*, 12:2002.

Thiemann, P. (2002a). Programmable type systems for domain specific languages.

Thiemann, P. (2002b). Wash/cgi: Server-side web scripting with sessions and typed, compositional forms. In *Practical Aspects of Declarative Languages: 4th International Symposium, PADL 2002, volume 2257 of LNCS*, pages 192–208. Springer-Verlag.

Thiemann, P. (2005). An embedded domain-specific language for type-safe server-side web-scripting. *ACM Transactions on Internet Technology*, 5:1533–5399.

Chisholm, W., G. V. I. J. (1999). Web content accessibility guidelines 1.0. http://www.w3.org/TR/WCAG10/.

Wallace, M. and Runciman, C. (1999). Haskell and xml: Generic combinators or type-based translation? pages 148–159. ACM Press.

Wittersheim, A. (2006). Why comply? the movement to w3c compliance. http://ezinearticles.com/?Why-Comply?-The-Movement-to-W3C-Compliance&id=162596.

Xmlbeans (2010). Xmlbeans. http://xmlbeans.apache.org/.

Xmlc (2008). Xmlc. http://xmlc.enhydra.org.