Semantic Web Accessibility Testing via Hierarchical Visual Analysis

Mohammad Bajammal
University of British Columbia
Vancouver, BC, Canada
bajammal@ece.ubc.ca

Ali Mesbah
University of British Columbia
Vancouver, BC, Canada
amesbah@ece.ubc.ca

***Abstract*—Web accessibility, the design of web apps to be usable by users with disabilities, impacts millions of people around the globe. Although accessibility has traditionally been a marginal afterthought that is often ignored in many software products, it is increasingly becoming a legal requirement that must be satisfied. While some web accessibility testing tools exist, most only perform rudimentary syntactical checks that do not assess the more important high-level semantic aspects that users with disabilities rely on. Accordingly, assessing web accessibility has largely remained a laborious manual process requiring human input. In this paper, we propose an approach, called AXERAY, that infers semantic groupings of various regions of a web page and their semantic roles. We evaluate our approach on 30 real-world websites and assess the accuracy of semantic inference as well as the ability to detect accessibility failures. The results show that AXERAY achieves, on average, an F-measure of 87% for inferring semantic groupings, and is able to detect accessibility failures with 85% accuracy.**

***Index Terms*—web accessibility, web testing, accessibility testing, visual analysis**

I. INTRODUCTION

Web accessibility is the notion of implementing web apps in a fashion that allows programmatic access to software functionalities that are otherwise only perceivable through certain senses (e.g., visually). Accessibility has been sometimes dealt with as an afterthought or a nice to have optional feature, with many developers and companies ignoring it altogether [1], [2]. However, as shown in Figure 1, millions of people around the globe have software-related disabilities and are impacted by web accessibility, or the lack thereof.

Furthermore, accessibility is increasingly becoming a *legal* requirement ratified into laws in many countries. For instance, in the United States, the Americans with Disabilities Act [3] requires all government and public agencies, as well as certain businesses, to make all their software and information technology services accessible. Similar provisions are required by law under the European Union's Web Accessibility Directive [4]. Figure 2 shows the increasing number of lawsuits filed in federal US courts against businesses for failing to provide accessibility accommodations.

Despite the increasing legal, economical, and human costs due to lack of accessibility, there has been little work in the software engineering research community to automate accessibility testing. Eler et al. [5] check for missing attribute fields or incorrect attribute values related to accessibility of web pages, an approach that is also used in a few patents [6], [7]. The bulk of existing work focuses on topics such as evaluating best practices for conducting empirical accessibility evaluations, such as manual checklist walkthroughs [8] or enlisting visually-impaired users as part of user studies [9]. A number of open-source tools have been developed to conduct simple syntactic accessibility tests that only check a few attribute values in a web page. In an audit of 13 of such accessibility testing tools conducted by the United Kingdom Government's Office of Digital Services [10], these tools found only %26 of a known small set of accessibility violations present in tested web pages.

The aforementioned tools are based on conducting *syntactic* checks, which are simple markup rules (e.g., any `a` element must contain a non-empty string) to check for a minimal level of accessibility. However, none of the aforementioned tools analyze key accessibility requirements related to the *semantics* of a page, such as the high level page structure and the roles or purpose of various elements. It is these semantic aspects of a page that users with disabilities rely on the most while using web pages [11], but none of the existing tools test for. Accordingly, the high level semantic analysis of web accessibility has remained a manual and laborious time consuming process [12], [13], [14], [15].

To that end, we propose an approach that automates testing of a subset of web accessibility requirements pertaining to high level semantic checks that have not been amenable to automation. The approach is based on a visual analysis of the web page, coupled with a few natural language processing (NLP) steps, to conduct a semantic analysis of web pages. This analysis first identifies major cohesive regions of a page, then infers their *semantic role* or purpose within the page. Subsequently, a conformance check is conducted that determines whether the markup of the page correctly corresponds to the inferred semantics. If the markup contains the same semantic information perceivable by sighted users, the page is deemed accessible. Otherwise, if semantic information of the page is perceivable only visually but not conveyed in the markup, the page would be flagged as failing the accessibility test and the specific reasons are reported. In this work, we focus on vision disabilities as opposed to other forms of disability (e.g., hearing). The rationale for this is twofold. First, the web is predominantly a visual medium where most of the

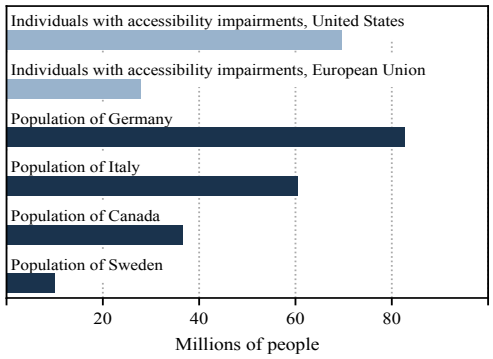


Figure 1. Number of people with software-related disabilities (e.g., vision, hand control, cognitive, or hearing) in the United States and the European Union. (Data compiled from: [16], [17])

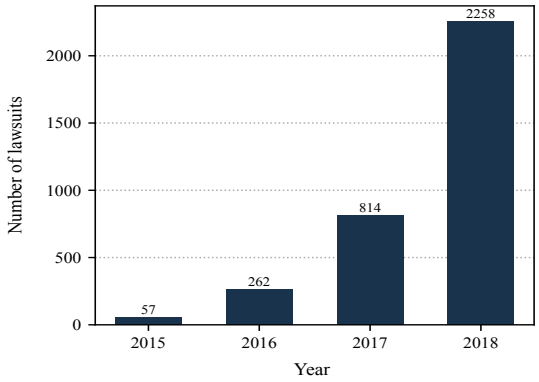


Figure 2. Number of software accessibility lawsuits filed in US federal courts per year. The number of lawsuits increased by around 3800% over the four-year period 2015-2018. (Data compiled from: [18], [19])

information is accessed visually as opposed to other senses. Second, surveys have shown that vision disabilities are the most relevant to web users with disabilities [11].

This paper makes the following main contributions:

- A novel approach for automatically testing semantic accessibility requirements, which is the first to address this issue, to the best of our knowledge.
- A technique that is based on visual analysis, coupled with a few natural language processing steps, to infer structure and semantic groupings on a web page for the purpose of accessibility testing.
- An implementation of our approach, available in a tool called AXERAY.
- A qualitative and quantitative evaluation of AXERAY in terms of its inference accuracy and ability to detect accessibility failures. The results show that it achieves an F-measure of 87% for inferring semantic groupings, and is able to detect accessibility failures with 85% accuracy.

II. BACKGROUND AND MOTIVATING EXAMPLE

Figure 3 shows an example of an inaccessible web page. In a quick glance at the rendered page in (c), a sighted user can immediately understand the structure of the page and navigate their way through the various contents of the page. For instance, the user would immediately recognize that they can navigate to other areas of the web site through the navigation menu at the top (i.e., Home, News, FAQ).

While this happens naturally and instantaneously for sighted users, that is not the case for visually impaired (i.e., non-sighted) users. The page structure (e.g., the presence of a navigation bar at the top) is *communicated exclusively* through visual design, since the HTML markup in Figure 3(a) is simply a collection of `<div>` s that do not communicate any semantic functionality. This implicit visual communication is intuitive and natural for sighted developers and users, but is unavailable for users who can not have *access* to visual information due to disabilities. Accordingly, the markup is deemed *inaccessible*, because it is expressed in a fashion that does not provide any semantic information about the page structure.

The analysis and conclusion that we just made is currently being done manually, since it requires high level semantic analysis of the page. Our goal in this work is to automate the reasoning we have just described in order to be able to automatically reach conclusions about the accessibility of web pages.

A. ARIA roles

The aforementioned lack of semantic markup in the code makes it difficult for non-sighted users to navigate the page. This is because such users rely on *screen readers* to parse the page for them and present them with the various information or navigation options present in the page. Screen readers are tools that speak out the various options, regions, or tasks accessible from the page, and the non-sighted user would then select one of the options they heard from the screen reader. Screen readers can be thought of as web browsers for non-sighted users, with one major caveat. While a standard web browser simply renders the page as is and leaves it to the end user to understand what the various elements mean, screen readers expect that the page markup contains semantic information about various areas of the page, which the reader would then announce audibly, giving non-sighted users an understanding of the overall semantic structure of the page.

The standard that is used by screen readers during their processing of web pages is the W3C *Accessible Rich Internet Applications* (ARIA) [20] semantic markup. The ARIA standard specifies a set of markup attributes that should be included in the page's HTML to make it accessible to screen readers or other assistive technologies.

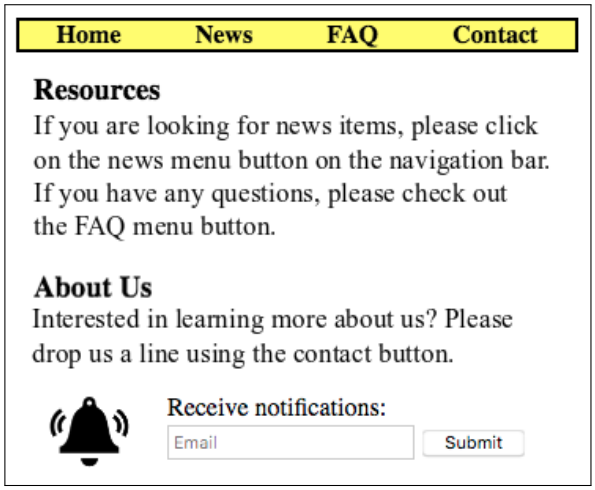
Targeted roles. ARIA defines more than 80 attributes spanning various aspects of the page, from low level syntax requirements to high level semantic structure. We therefore have to target a subset of these ARIA attributes because addressing all 80+ attributes in a single academic paper is untractable. The basis for selecting the targeted roles is that we focus on the most commonly used [11] subset of ARIA, which are *landmark roles*. Our own experimental results (Section IV-B2) also demonstrate the widespread use of these roles. The roles

```html
<div class="nv8471">
<div>Home</div>
<div>News</div>
<div>FAQ</div>
<div>Contact</div>
</div>

<div class="_hd902">
Resources
</div>
<div>
 ...
</div>

<div class="_hd902">
About Us
</div>
<div>
 ...
</div>
```

(a) HTML code

```css
.nav8471 {
background-color:
  #ffff00;
display:
  flex;
justify-content:
  space-around;
font-weight:
  bold;
border:
  2px solid #000000;
}

._hd902 {
font-size:
  5vw;
font-weight:
  900;
padding:
  8vh 0 2vh 0;
}
```

(b) CSS declaration



(c) Rendered page

Figure 3. An example of an inaccessible web page.

identify the major high-level regions of the page and specify the semantic role for each of them. They consist of a set of specific pre-defined roles that convey, though markup, the otherwise only visually perceived role of each region. For instance, the yellow rectangle at the top of Figure 3(c) is perceivable as being a navigation bar, therefore the markup of the element containing that region has to include the landmark attribute `role="navigation"`. When a non-sighted user loads the page, the screen reader would speak out the presence of a navigation bar (and any other semantic regions in the page), allowing the user to quickly perceive the high level structure of the page, and directly interact with the region they are looking for.

Accordingly, the absence of semantic markup in Figure 3(a) makes the page inaccessible, because screen readers would be unable to provide alternative, non-visual, means to access the page. That is, the information and cues about the structure and navigation of the page remain locked in the visual design of the page and can not be accessed non-visually. The task of ensuring web accessibility would therefore rely on one key principle: any information, structure, or functionality that is visually perceivable by sighted users must also be available for non-sighted users. When a web page communicates its structure and function using only visual design, without providing a programmatic means to access the same information, it is deemed inaccessible.

B. Existing tools: syntax checkers

Existing accessibility testing tools [10] are based on syntactic checking. They check the HTML against certain syntax rules. For instance, one common check is to assert that every `img` element has an `alt` text attribute (for text descriptions). Another check asserts that `input` elements must not be descendent of `a`. Another example is checking that every `form` element has a `label` attribute. In general, the syntactic checks all follow the form: if certain syntax A is present, assert that syntax B is true.

While such syntax checks can be useful simple assertions and are easy to automate, none of the existing tools perform the more important, and more challenging, high level *semantic* analysis of the page's content. For instance, consider Figure 3(c), where we can see that there is a navigation bar at the top. Now consider the HTML in (a), where we observe that the developer did not use the required ARIA attribute of `role="navigation"`. Accordingly, because we visually see a navigation region in (c) but do not find it expressed in the markup in (a), we conclude that the page has an accessibility failure, because we perceived a certain semantic structure as sighted users, but it was not expressed in the markup in order to ensure it would be equally available for non-sighted users. That is the the *central* issue in accessibility: making sure that whatever is visually perceived is also expressed in the code.

We now pause to reflect and observe that there is no syntactic check that can automate the conclusion we just reached. That is, the process that we just walked through requires our own visual perception as humans in order to reach the conclusion that there is a mismatch between the perceived page and the markup. This process can not be put in the form of a syntax check.

In the next section, we describe how we construct an approach that automates the accessibility testing procedure that we have just manually conducted.

III. Approach

In this paper, we propose an approach to automatically test for a subset of web accessibility violations that are pertinent to semantic structure. We recall that the scope of this work focuses on vision disabilities as opposed to other forms of disability, due to the web being a predominantly

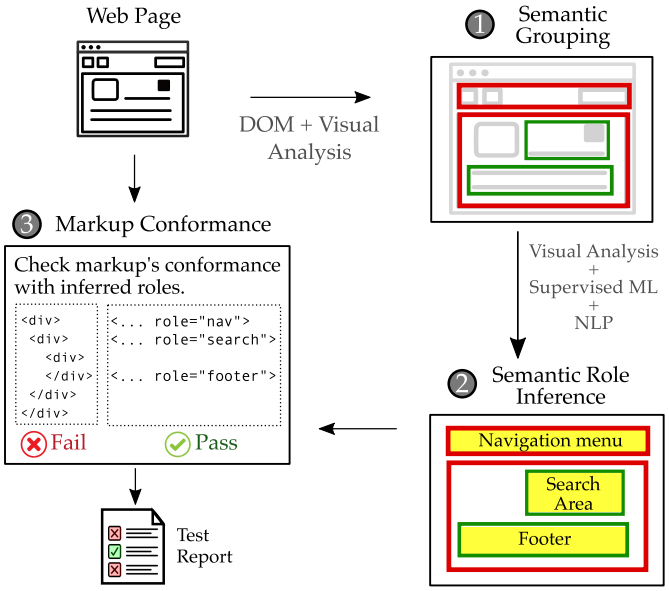


Figure 4. Overview of the proposed approach.

visual medium and the fact that vision disabilities are the most common software-related disability [11].

Figure 4 shows an overview of our proposed approach, which is based on the strategy of visually analyzing the web page to infer semantic groupings and their roles, and then checking that the HTML markup matches the inferred semantic roles. The approach begins by obtaining the Document Object Model (DOM) and screenshot of the web page rendered in a web browser. Next, the set of visibly perceivable objects is identified. This is then used to perform a semantic grouping of the page into a set of semantically coherent regions. Subsequently, this information is used in an inference stage where the specific semantic role of each region is detected. Finally, the inferred semantics are checked against the markup used in the page, and a report is generated as to which parts of the page are inaccessible. The rationale behind this strategy is to check whether the semantics visually perceivable by sighted users are reflected in the semantics of the HTML markup, thereby ensuring accessibility.

A. Visual Objects Identification

In this first stage, the goal is to identify objects that are perceivable by sighted users, which we refer to as *visual objects*. For instance, in Figure 3(c), each item in the top navigation menu would be a visual object. This step of visual objects identification is the foundation of our overall approach, since the identification of visual objects enables checking whether the information and elements that are perceivable by sighted users are also accessible to non-sighted users.

1) Objects Extraction: We begin by taking as input the DOM of the page after it is loaded and rendered in a browser. We then extract from the DOM a set of nodes that represent visual content of the page, and we refer to each of these as *Visual Objects*. We define three types of Visual Objects: textual, image, and interactive.

Textual Objects. The extraction of text content is achieved by traversing text nodes of the DOM. More specifically:

$$\Theta_t := \{E \mid \nu(E) \wedge \tau(E)\} \quad (1)$$

; where Θ_t is the set of all visual objects that represent text in the page, $E \in DOM$ is a leaf element iterator of the rendered DOM in the browser, $\nu(E)$ is a heuristic predicate that runs a series of checks to detect visually perceivable elements (as will be described in section III-A2), and $\tau(E)$ is a predicate that examines whether there is a text associated with E. More specifically, it returns non-empty nodes of DOM type `#TEXT`, which represent string literals. An example of extracted textual objects would be the "Resources" section in Figure 3(c). We note that the predicate is based on a node type, rather than an element (i.e., tag) type. This allows more robust abstraction because the predicate captures any text and does not make assumptions about how developers choose to place their text. In other words, regardless of the tag used for text data (e.g., `<span>, <div>`), text would still be stored in nodes of type `#TEXT`, even for custom HTML elements. This helps in making the approach more robust by reducing assumptions about tags and how they are used in the page.

Image Objects. Subsequently, we perform another extraction for image objects. We define this as follows:

$$\Theta_m := \{E \mid \nu(E) \wedge \mu(E)\} \quad (2)$$

where Θ_m is the set of all visual objects that represent images. As in the previous case, the predicate $\mu(E)$ examines whether there is any relevant image content associated with E. This has two possibilities: a) nodes of `<img>`, `<svg>`, and `<canvas>` elements, and b) non-image nodes with a non-null background image. An example of extracted image objects would be the bell icon in Figure 3(c). We note that this predicate makes the proposed approach more robust by eliminating assumptions about how developers markup images. If images are contained in standard tags (e.g., `<img>`, `<svg>`), then the predicate readily captures them. However, we make no assumptions that this is the only way an image can be included. For this reason, we also capture elements of any type when a non-null background image is detected.

Interaction Objects. Finally, we extract the interaction elements as follows:

$$\Theta_i := \{E \mid \nu(E) \wedge \eta(E)\} \quad (3)$$

where Θ_i is the set of all visual objects that represent form elements or similar interactive elements. These are determined by the predicate $\eta(E)$, which collects elements such as input fields and drop down menus. An example of extracted interaction objects would be the Email input field in Figure 3(c).

2) Visual Assertion: After the preceding extraction of an initial set of visual objects, this stage proceeds by conducting a visual analysis of the objects. This analysis detects if an object is visually perceivable. We conduct the visual analysis

as follows. First, we obtain the box model of each object. We use the *computed* box model in order to faithfully capture the location as finally rendered on screen. Next, we obtain a screenshot of the region defined by the box model. We then analyze the screenshot using the Prewitt operator [21] used in computer vision. This operator applies a set of derivatives or differentiation operations on the image, and then typically used to detect salient visual features in the image (e.g., shapes, textures). We therefore use this operator to extract any visual features present in the image, regardless of the form of these features. Depending on the presence or absence of visual features, the perceptibility state of the object is determined. If no visual features are detected, the object is deemed to be non-perceivable, and vice versa. For example, consider Figure 3(c). The navigation region in the top, and the main content that follows it, are perceivable by sighted users. However, web pages also have spacing elements that do affect the layout but are not individually perceivable themselves. For instance, there can be an element between the navigation bar and the "Resources" section such that a certain vertical distance is maintained below the navigation bar. While such a spacing element certainly affects the layout and occupies screen space, it does not constitute a visual object due to it's imperceptibility.

B. Semantic Grouping

After the visual objects identification is completed, we proceed by grouping visual objects into groups representing potential semantically relevant regions on the page. For instance, in Figure 3(c), one semantic grouping would be the navigation region at the top of the page. Another semantic grouping would be the "Resources" and "About Us" sections representing the main content of the page.

The rationale for this step of the approach is as follows. We recall that screen readers expect the markup to indicate the major semantic regions of a page. Accordingly, in order to automatically assert that any visually perceivable semantic region has been also expressed in the markup, we first need a mechanism by which we can detect the semantic regions in the first place. This is what we aim to achieve in this stage. Here we are only concerned with creating potential semantic groupings, while the next stage (Section III-C) infers what exactly is the semantic role (if any) of each potential grouping.

The grouping uses both structural (DOM) information as well as visual analysis. The DOM is used to generate a large number of potential seed groupings, and the visual analysis performs filtering and further analysis to produce a final set of groups. We adopted this strategy for the following reasons. We observed that the DOM can be used as a source of seed groupings due to its inherently hierarchical nature that also tend to capture the developer's or designer's own intended semantic grouping. That is, the children of a node constitute a rudimentary form of a group, which can then be further analyzed, merged, or divided, to create a refined grouping. Subsequently, visual analysis filters these initial seed groups and process them to construct semantic groupings. Visual analysis is used because, while the DOM may provide seed groupings, it does not faithfully represent what the end user is actually observing on the screen.

Grouping process. We now describe the mechanism of the grouping process. First, we obtain one flat non-hierarchical set of all DOM elements. For instance, in Figure 3(a), this would be all the `div` elements in one flat set. The elements are collected regardless of visibility, due to the complex nature of DOM and CSS rendering where non-visible nodes can contain visible children. For this same reason, the initial set of elements is flat and non-hierarchical, because visible children can often be inside non-visible nodes, and therefore relying on DOM hierarchy would yield many false positives and negatives. Instead, we build the hierarchy by visually analyzing the collected flat set of elements. We do this by first collecting the computed box model of each element in the set. For instance, in Figure 3(a), this would result in a set containing the computed box model of each `div` element regardless of hierarchy. Next, we remove box models that are visually located outside the page boundaries, since they are not perceivable to sighted users. For boxes that are only partially outside the page, we trim them to page boundaries. Subsequently, we filter *equivalent* boxes, which is when a pair of boxes visually contain the same set of visual objects. We do this by removing the smaller box (in terms of visible area) in a pair of equivalent boxes. Next, we filter boxes based on how many visual objects are visually contained (i.e., located) within them. We remove each box that visually contains the entire set of visual objects on the page. For instance, in Figure 3(a), any `div` that visually contains the entire set of all `div`s is removed. This is because such a set does not represent any semantically useful grouping, since the entire set of objects is in one group only. Finally, we iterate over the set of visual objects. For each object, we find the largest box that visually contains the object. Once this is completed for all visual objects, the final result is a set of boxes representing the potential semantic groupings on the page.

C. Semantic Role Inference

Once semantic grouping is completed, we proceed to infer the semantic role of each group. This step infers one of the pre-defined landmark roles (Section II-A). An example can be seen in the top navigation bar in Figure 3(c), indicating the pre-defined role of `navigation`. However, not all roles are relevant to our scope of automated semantic analysis. For instance, `region` is a generic catch-all label that does not convey any specific semantic role, and its use is generally discouraged and typically not used by screen readers. Another example is `form`, a label that indicates form regions. The label is directly associated with HTML `<form>` elements, and therefore no semantic analysis or inference is needed for its detection. Accordingly, we focus our semantic analysis on the more relevant roles of `main`, `navigation`, `contentinfo`, and `search`, which will be described in the following sections.

Main Role. The `main` ARIA role indicates a region that contains the main output or results in a web page. For example,

on the search results page of a search engine, the region containing the list of retrieved search results would be the main region, which is then surrounded by other regions such as the navigation bar or footer.

The process by which we infer the role of a group to be `main` is as follows. First, we compute a score for each detected group in the page. The score uses both visual geometrical attributes as well as natural language processing (NLP) measurements. More specifically:

$$\psi_{main}(r) = A(r)\rho(r) \tag{4}$$

where r is a semantic grouping of the page, ψ_{main} is the score, $A(r)$ is the visual geometric area for r, and $\rho(r)$ is an NLP metric we define to measure linguistic aspects of the contents of r. More specifically, $\rho(r)$ first performs a part-of-speech (POS) tagging, which is a common NLP analysis than assigns POS labels (e.g., verb, noun, adjective) to each word. $\rho(r)$ then measures the variance of the linguistic POS tag frequencies of all textual objects contained in r. We give an example to clarify the various measured values. Consider the rendered page in Figure 3-c. r would represent, for instance, the region containing the body of the page (e.g., the Resources and About Us sections). $A(r)$ would be the geometric area of that region as visible on the screen. The rationale is to capture how much would a region occupy the visible space for sighted users. As for $\rho(r)$, it first collects all textual objects (as explained in section III-A1) within r, which would collect all text elements such as "Resources", "About Us", as well as the paragraphs on the page. For each text object, POS tags are collected, and then their frequencies (i.e., count of each tag type) are computed. ρ then measures the variance of these POS tag frequencies. For instance, a navigation region r that has, say, the textual objects "Images", "News", and "Settings" has no variance since they all have identical POS tags. Contrast this with the main body of text in a page, which contains elements such as such as paragraphs, section headings, links, and much more. The likelihood of all such content to be linguistically monotonous (i.e., all tags are nouns) is practically negligible. This is why eq. (4) includes the $\rho(r)$ factor. The $A(r)$ in the equation accounts for the fact that it is unlikely that the main region of the page would be the visually smallest area on the page. Once the score in eq. (4) is computed for all detected regions, we sort the regions by score and select the region with the highest score, which is finally reported to be the region having the main role.

Navigation Role. The `navigation` ARIA role indicates a region in a webpage that allows users to navigate between various pages or views. The process by which we infer the role of a group to be `navigation` is as follows. We first compute a score for each group, using the following equation:

$$\psi_{nav}(r) = \frac{C(r)}{1 - \rho_h(r)} \tag{5}$$

where r is a semantic group of the page, ψ_{nav} is the score, and $C(r)$ is a metric that measures the *clickables ratio* inside the group r. This computes the ratio of visual objects that appear to be clickable to sighted users, which we define as any visual object whose onscreen cursor is a hand or a pointer, indicating to sighted users that it can be clicked on. Accordingly, a group that has high $C(r)$ is mostly composed of objects that a sighted user can click on, which is typically the case for navigation regions. For example, in Figure 3-c, the yellow navigation region at the top contains elements that all appear as clickables to sighted users. In contrast, a group that does not contain any clickables (e.g., only static texts and images) would have a $C(r)$ equal to zero and therefore is not a navigation region. This can be seen, for instance, in Figure 3-c in the body of the page below the navigation bar, where the body contains only static text paragraphs or images. $\rho_h(r)$ is a measure of the homogeneity of the contents of r. For semantic groups containing only textual elements, $\rho_h(r)$ is the same NLP linguistic variance metric we defined in eq. (4). For all other elements, $\rho_h(r)$ represents the dimensional variance of the objects in r. Finally, a given group is inferred to have a navigation role when ψ_{nav} greater than or equal unity, which was determined empirically.

ContentInfo/Footer Role. The `contentinfo` role (also known as the footer role) indicates regions of the page that represent complementary content to the parent document. That is, instead of containing the main output of the page or the main navigation elements, footer regions serve as complementary content or information that comes after the main content. In a similar fashion to previous roles, we compute a score for each detected grouping, using the following equation:

$$\psi_{footer}(r) = \frac{C(r)D(r)}{A(r)} \tag{6}$$

where, as in the previous roles, r is a semantic group of the page, ψ_{footer} is the score, $A(r)$ is the visual pixel count for r, $D(r)$ is the visual distance from the geometric center of r to the origin of the screen, and $C(r)$ is the clickables ratio in r as defined in eq. (5). As can be observed from the equation, the score is mostly concerned with the visual geometric aspects of the region, since this ARIA role is, by definition, spatial in nature since it refers to a specific spatial visual placement on the page. Accordingly, we compute and sort the score for all groups, and select the group with the highest score. If the group is located in the lower half of the page, it is reported as a footer. Otherwise no footer regions are reported.

Search Role. The `search` ARIA role indicates regions in a page that allow users to enter a search query and retrieve items on the page or site. To infer this role, we use a combination of visual analysis, a supervised machine learning model, as well as linguistic (i.e., keyword) techniques.

First, we train a Convolutional Neural Network (CNN) to visually recognize search icons. We collected and labeled 500 data points representing icon images (50% positive examples) and used the Inception CNN architecture [22], which has been shown to produce very effective classifications for computer vision machine learning problems [22]. Subsequently, we use this model to find search icons on a page. Next, we perform a nearest neighbor search to look for text input fields in the

spatial vicinity of detected search icons. If a text input field is found, we mark the region containing the search icon and the input field as having a search semantic role. Furthermore, we also check for cases where the search input text field has no associated search icons. In this scenario, we extract all text input fields on the page. We then perform a nearest neighbor search to find any visible label texts in the visual spatial vicinity around the input field. We then conduct NLP *stemming* on the label text and find those that include key linguistically significant *stem words*, such as "find", "search", and "locate." Any detected group that matches any of the above cases is marked as having a search semantic role.

Finally, we note that, due to the non-hierarchical nature of our semantic groupings, all inferred roles are agnostic to hierarchies and the proposed approach is therefore able to detect hierarchical combinations of the inferred roles (e.g., a navigation region within a footer region).

D. Markup Conformance

This final stage asserts that the source of the page contains markup indicating the presence of the inferred semantic regions and their semantic roles. For instance, in Figure 3(c), the approach so far would infer that the group of elements at the top of the page represent a coherent semantic grouping, and that their semantic role is navigation. If the HTML markup corresponding to that area does not contain the ARIA landmark role of `navigation`, then screen readers will not be able to provide this semantic information to users, and we recall that this semantic information is among the most important and widely used of ARIA roles by users with disabilities [11]. Therefore, in such cases where the markup does not conform to the inferred semantic roles, we report an accessibility failure and indicate the expected semantic markup and where it should have been expressed in the page.

The mechanism of checking markup conformance is as follows. First, we obtain the semantic groupings and any inferred roles, as described in the previous sections. For each semantic group, we identify all DOM elements that satisfy two criteria: 1) all visual objects of the group are located inside the element's box model, and 2) the element's box model is located inside the group's box model. This process captures all possible DOM elements that would qualify as a root for the region, without including objects from other regions. Any of these DOM elements would therefore have to contain markup indicating the presence of a region and its role.

We then check whether any element in the set meets both of the following requirements: 1) the element has a `role` attribute whose value matches the inferred semantic role of the group. 2) the element's computed box model visually overlaps the box model of the inferred semantic group. The rationale for adopting this approach is as follows. As we noted in Section III-B, the complex nature of DOM and CSS rendering easily allows cases where non-visible/non-rendered nodes can contain visible rendered children. A DOM-based approach (e.g., checking containment by XPath) would therefore yield many false positives and negatives. Accordingly, we use the visual check above for a more robust analysis.

If an element satisfying these requirements is found, we log it and move on to the next inferred semantic role and check that it has been correctly expressed in the markup. The process is repeated for all semantic groupings for which a role has been inferred. Any semantic grouping for which no role has been inferred is discarded. A report is finally generated indicating all roles that have been correctly expressed in markup, and all roles that should have been in the markup but are missing.

E. Implementation

We implemented the proposed approach in a tool called AXERAY (short for Accessibility Ray). It is implemented in Java. We use Selenium WebDriver to instrument browsers and extract DOM information and computed attributes. We use OpenCV [23] for computer vision computations, DeepLearning4J [24] for machine learning operations, and the Stanford CoreNLP library [25] for linguistic analysis. To make the study replicable, we made available online a link to our AXERAY tool and the anonymized participants' responses [26].

IV. EVALUATION

To evaluate AXERAY, we conducted qualitative and quantitative studies to answer the following research questions:

RQ1 How accurate is AXERAY in inferring semantic groupings and semantic roles?

RQ2 To what extent can AXERAY detect accessibility failures in real-world web pages?

In the following subsections, we discuss the details of the experiments that we designed to answer each research question, together with the results.

A. RQ1: Semantic Grouping and Roles Inference

In this question, the objective is to assess how accurate is the semantic grouping and semantic role inference processes. The rationale for evaluating this aspect is that the approach first performs the grouping and semantics inference, and then uses this inference to test for accessibility. Accordingly, we first need to assess the inference process itself.

We evaluated this question as follows. First, we collected 10 random subjects from the Moz Top 500 [1] most popular websites. We then ran AXERAY on each test subject's URL and obtained the output groupings and semantic roles. Figure 5 shows an example of the output. Each rectangle represents an inferred grouping, together with it's semantic role. Subsequently, we recruited human evaluators. 10 evaluators were recruited from the MTurk [2] crowdsourcing platform. The qualifications of participants are to be working in the software industry and to have maintained the highest level of accuracy on the MTurk platform, which is referred to as Masters level.

Subsequently, each human evaluator was presented with the output of AXERAY for all test subjects, and asked to assess

[1] https://moz.com/top500

[2] https://www.mturk.com

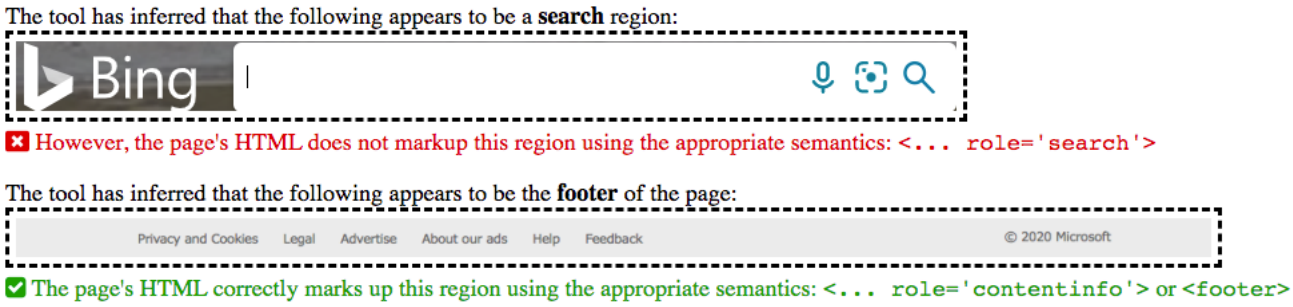


Figure 5. Sample of the generated accessibility report.

the accuracy of groupings and roles. More specifically, we asked them to identify any output groups that do not represent a meaningful semantic grouping. This represents the *false positives* of groupings, while the remainder are *true positives*. We also asked them to identify any meaningful semantic groupings on the page that were not included in the output. These are the *false negatives*. The same process is repeated for the semantic roles.

1) Results and Discussion: Table I shows the results of evaluating the accuracy of semantic grouping and role inference. The columns show the precision, recall, and F-1 measure averaged across evaluators. The two groups of columns, labeled "Grouping inference" and "Role inference", show the accuracy of the proposed approach in inferring semantic groupings and semantic roles, respectively. The highlighted cells show the minimum and maximum values in each column.

The key outcome of this evaluation is the F-1 measures, which are at 87% and 90% for grouping and role inference, respectively. These values indicate a rather effective inference process. The lowest precision was 71%. This often happens due to a somewhat unusual DOM structure, where elements in the same region were placed at large tree depth separations from one another. This resulted in mistakenly grouping a number of elements that should have not been grouped. As for the recall, the lowest performance was at 60%. Such low values often happen in corner cases where elements are falsely excluded from groups due to having an empty box model stemming from complex nested CSS rules, despite being present in the group.

Table I
PRECISION AND RECALL OF GROUPING AND ROLE INFERENCE. HIGHLIGHTED NUMBERS ARE THE MINIMUM AND MAXIMUM VALUES IN EACH COLUMN.

	Grouping inference			Role inference		
Subject	**Prec.**	**Recall**	**F-1**	**Prec.**	**Recall**	**F-1**
wikipedia.org	94%	89%	91%	91%	91%	91%
google.com	88%	85%	87%	86%	81%	83%
amazon.com	92%	87%	89%	**83%**	**79%**	**81%**
stackoverflow.com	91%	94%	92%	91%	81%	86%
medium.com	96%	94%	**95%**	94%	**99%**	96%
khanacademy.org	92%	**98%**	95%	96%	89%	93%
imdb.com	**71%**	86%	78%	92%	96%	94%
cnn.com	**97%**	92%	95%	**97%**	97%	**97%**
rt.com	92%	**60%**	**72%**	94%	90%	92%
booking.com	92%	73%	81%	93%	89%	91%
Average	**90%**	**85%**	**87%**	**91%**	**89%**	**90%**

B. RQ2: Accessibility Failures Detection

While the previous question examined how accurate the inferred semantic groupings and roles are, this RQ evaluates to what extent the inferred information can be used to reliably detect accessibility failures.

Due to the absence of ground truth data, we evaluate this RQ in two complementary ways: 1) using a *fault injection* experiment, and 2) evaluating the output on a large number of real-world subjects in the wild. The rationale for using these two complementary ways is as follows. In the fault injection experiment (explained section IV-B1), existing semantic markups (if any) are removed (thereby simulating a fault) and a check is made whether the tool is able to detect this removal. The benefit of this experiment is that it allows automated evaluation without a subjective assessment. The drawback, however, is that it is only a *lower bound* of the actual accuracy, since it says nothing about other possible role faults that have not been injected (i.e., due to the absence of some roles in the original markup itself). For this reason, we supplement the fault injection experiment with a manual item-by-item evaluation of the tool's output in order to evaluate all true/false positive and negative results. We describe each approach in the following subsections.

Subjects. We conducted the experiments in this RQ on a total of 30 real-world subjects. The subjects were collected in two ways. The first half of subjects were randomly selected from the Moz Top 500 most popular websites as in RQ1. The second half of subjects were obtained from Discuvver.com, which is a service that returns a random website from the internet.

1) Fault injection: For each subject, we inject a random fault in its markup, and assess if AXERAY was able to detect the fault. We recall from section II-A that a web page is deemed inaccessible if there is an *absence* of semantic roles. That is, the developer did not add the necessary semantic markup to the page. Our goal is thus to simulate this behavior by *removing* all existing semantic markups on the page, and therefore create a new page as if the developer had not included the necessary semantic markup, and then check whether our approach can re-detect them. Such markup omission, by definition, is what makes web pages inaccessible,

Table II
RESULTS OF DETECTING FAULT INJECTIONS.

Subject	Total # injections	# Detected faults	
		proposed approach	baseline
bing.com	1	1	0
youtube.com	3	3	0
google.com	4	3	0
microsoft.com	7	5	0
bbc.com	5	3	0
amazon.com	4	3	0
medium.com	N/A		
yahoo.com	3	1	0
live.com	4	4	0
paypal.com	2	1	0
blogger.com	2	2	1
netflix.com	N/A		
stackoverflow.com	3	3	0
imdb.com	4	3	0
walmart.com	3	2	0
fuelly.com	1	1	0
typing.com	3	2	0
iconpacks.net	2	2	0
expatistan.com	N/A		
memrise.com	N/A		
retrevo.com	3	3	0
startupstash.com	4	3	0
eatthismuch.com	1	0	0
kdl.org	3	2	0
getpocket.com	2	2	0
retailmenot.com	3	3	0
mailinator.com	N/A		
myfridgefood.com	1	0	0
joinhoney.com	2	2	0
bannereasy.com	1	1	0
Total	71	**77.5%**	**1.4%**

and is therefore the only meaningful fault type. In other words, any mutation of the markup is effectively a markup omission, since the exact expected semantic role would become absent from the markup. Misspelled attribute values are also effectively markup omissions. For instance, suppose that a region should have been marked as a navigation region. Whether the `navigation` role was completely absent from the markup, or was misspelled (i.e., `nagviton`), both cases are still effectively markup omissions.

We now describe the injection process. First, we load the subject in an instrumented browser (i.e., via Selenium). We then remove all semantic markups on the page (i.e., landmark `role`s). We then apply AXERAY on the subject and collect the output. If after the fault injection (i.e., removal of `role`s) AXERAY was able to indicate that there should be a semantic `role`, we conclude that the injected fault has been caught. Otherwise, the fault was not detected.

Baseline. In order to have a more thorough evaluation, we included a baseline in our experiments. However, since there are no existing tools that perform semantic checking, our baseline consists of a simple random selection process. In this process, random regions from the page are selected. Next, a random semantic role is assigned to each randomly selected region. This set of semantic regions and their semantic roles is then taken to be the baseline.

Results and Discussion. Table II shows the results of evaluating the fault injection experiment. The first column shows the total number of fault injections performed on each subject. We recall that this first column is not a number we chose; it is rather the total number of injections that were *possible*, since the faults are removals of existing semantic markup. The second column shows the number of injected faults that were successfully detected by, whereas the third column shows the number of injected faults that the tool has failed to detect. The last row sums up the results across all subjects.

The main result of the evaluation is that AXERAY has detected, on average, around 77.5% of the injected faults. While this performance is relatively good, given that this sort of analysis hasn't been automated so far, we do note this is only a lower bound of the actual accessibility failure detection ability, since it says nothing about other possible failures that have not been injected (e.g., where the original markup itself did not include certain semantic roles).

In certain subjects in Table II, marked with "N/A", the fault injection process was not possible. This was because the subject's markup did not contain any of the landmark semantic roles, and therefore it was not possible to remove them them and check if our tool was able to detect them back. Despite some of these subjects being top 100 websites, the lack of such markup in the subject is an example that illustrates the need for effective and automated accessibility testing.

2) Direct output evaluation: In this step, we manually evaluate the output on a large number of real-world subjects in the wild. First, we loaded each test subject in an instrumented web browser (i.e., via Selenium). Page popups or notifications, if any, were closed. Next, we applied AXERAY on the subject and collected the generated output (as shown in Figure 5). We then categorized each item in the report into one of the following: **True positive:** This represents a true accessibility failure. This category holds whenever the tool has reported the absence of a correct semantic role that is indeed missing from markup, and therefore the reported failure is true. **False positive:** This is a false accessibility failure. In this case the tool has either reported an incorrect semantic role or the role is already in the markup but was falsely flagged as missing, and therefore the reported failure is false. **False negative:** This case is a false accessibility *pass* (not failure, as in the two previous cases). This represents cases where a semantic role should have been included the markup, but the tool did not report an accessibility failure. **True negative:** This corresponds to true accessibility pass. This represents cases where a role is actually semantically not present on the page, and the tool did not report a failure. This also corresponds to cases where a semantic role is present in the markup, and the tool has reported that the markup is conforming to the inferred semantic role.

Results and Discussion. Table III shows the results of the evaluation. Each row lists the subject, the true positive/negative and the false positive/negative for the subject. The last row shows the accuracy, precision, recall, and the F-1 measure.

Table III
DIRECT EVALUATION OF ACCESSIBILITY FAILURE DETECTION ON 30 REAL-WORLD SUBJECTS.

	Proposed approach				SortSite		Baseline			
Subject	**TP**	**FP**	**TN**	**FN**	**semantic issues**	**syntactic issues**	**TP**	**FP**	**TN**	**FN**
bing.com	4	0	1	0	0	11	0	3	0	4
youtube.com	0	0	4	0	0	10	0	2	0	0
google.com	2	2	3	0	0	6	0	2	0	2
microsoft.com	2	1	10	0	0	10	0	1	0	2
bbc.com	2	2	2	2	0	3	0	4	0	4
amazon.com	4	0	7	0	0	13	0	1	0	4
medium.com	4	0	1	0	0	6	0	2	0	4
yahoo.com	1	0	1	3	0	9	0	3	0	4
live.com	2	0	4	0	0	2	0	3	0	2
paypal.com	3	0	2	0	0	13	0	2	0	3
blogger.com	1	0	3	1	0	5	1	2	0	2
netflix.com	2	0	1	1	N/A	N/A	0	4	0	3
stackoverflow.com	3	0	6	0	0	18	0	2	0	3
imdb.com	2	2	3	0	N/A	N/A	0	3	0	2
walmart.com	6	0	2	1	0	7	0	4	0	7
fuelly.com	3	0	2	0	0	12	0	1	0	3
typing.com	2	1	3	2	0	10	0	2	0	4
iconpacks.net	3	0	1	1	0	8	0	4	0	4
expatistan.com	2	3	1	0	0	9	0	1	0	2
memrise.com	4	1	2	0	0	5	0	1	0	4
retrevo.com	2	0	3	0	N/A	N/A	0	2	0	2
startupstash.com	1	0	4	0	0	12	0	4	0	1
eatthismuch.com	6	0	1	0	0	15	0	3	0	6
kdl.org	3	1	2	1	0	9	0	3	0	4
getpocket.com	2	0	3	0	0	7	0	1	0	2
retailmenot.com	5	0	3	0	0	2	0	2	0	5
mailinator.com	2	1	1	0	0	6	0	4	0	2
myfridgefood.com	2	1	1	1	0	8	0	2	0	3
joinhoney.com	3	0	3	0	0	7	0	3	0	3
bannereasy.com	4	0	2	0	0	5	0	1	0	4
	Acc.	**Prec.**	**Rec.**	**F1**			**Acc.**	**Prec.**	**Rec.**	**F1**
	85.4%	84.5%	86.3%	85.4%			0.6%	1.3%	1.0%	1.2%

The values of the accuracy, precision, and recall are between 84% and 86%, indicating a relatively good performance. The true positive column represents the true accessibility failures that are indeed present in the subjects. This certainly does not cover all possible accessibility failures, but rather the subset of accessibility issues that we focus on in this work (i.e., the semantic roles). The true negative column can be thought of as the number of cases where the markup of the subject is "in agreement" with the tool's output. In the median, two true accessibility failures were detected per subject, and two inferred semantic roles per subject were in agreement with what has been expressed in the markup. For the second half of subjects in Table III (i.e., the random sites), 60% have used semantic roles. In contrast, for the subset of top websites (i.e., the first half of subjects), 74% have used semantic roles. 40% of the random websites did not use any semantic roles, compared to 26% of the top websites. This observation is expected since top sites are more likely to have more resources to create better products. The average execution runtime was 17 seconds.

We then investigated the reasons behind the false positives and negatives. One common reason is erroneous inference of navigation roles. This occurred, for instance, for a region that consisted of weather forecast for the next few days. From the perspective of our inference procedure, this looked like a navigation area since it had a group of links that were coherent in content and presentation (in the sense described in Section III-C). Accordingly, it was falsely indicated as a navigation, and therefore resulted in a false accessibility failure. Another reason involves missing footer roles that should have been reported. This occurred, for instance, for a region that was not recognized by the semantic grouping stage, and therefore no role was able to be inferred for it.

For comparison, Table III also includes an evaluation of SortSite [3], which is the best performing state-of-the-art accessibility testing tool [10]. As mentioned in the introduction, SortSite and other state-of-the-art tools only perform syntactic checks and is therefore unable to detect the semantic issues that are the focus of this work. Accordingly, we run an evaluation that verifies this empirically. Each subject is fed to SortSite, and the output report is saved. Each reported failure is then categorized as either a syntactic issue or a semantic issue. We recall that, as discussed in Section II-B, a syntactic issue is any failure that only checks the syntax of the HTML. Examples include checks like ("each `a` element must contain non-empty text"), or ("`input` must not appear as a descendant of `a`"). These are direct checks that are only concerned with syntax. Compare this, for instance, with the reported failure shown in Figure 5. Here, the failure is *semantic* in nature. That is, the failure is not an application of

[3]https://www.powermapper.com/products/sortsite/

some syntactic rule. Rather, the failure is reported because the markup does not conform to how the page *is semantically perceived* from a visual perspective, not because it didn't conform to a predetermined syntactic rule.

From Table III, we observe that SortSite was able to find many syntactic issues (rows with N/A are cases were the tool was unable to load the subject). However, it did not detect any of the semantic issues. This is expected, as the rationale for this work was the observation that the state-of-the-art only conduct syntactic checks, which can not detect the more important and widely used semantic information.

3) Future work: In this work, we focused on an important subset of accessibility requirements, which are the semantic roles as discussed in II-A. Therefore, as expected, our approach can not cover all possible accessibility requirements. This leaves open a number of avenues for future work to address other accessibility requirements, each of which would require a novel technique to address. The variations in the semantics of various accessibility requirements and the lack of approaches to address them, mainly due to the difficulty of performing high-level semantic analysis, presents a rich and fertile ground to conduct research, which has received little attention from the software engineering research community as discussed in the introduction. For future work, we believe it will be a fruitful and interesting pursuit for the research community to explore some of these other accessibility requirements.

4) Threats to validity: We chose test subjects (i.e., web sites) randomly from the Internet with the mentioned criteria in Section IV, to avoid any selection bias. Plus, the participants were selected to be highly qualified evaluators at the crowdsourcing platform, mitigating the threats to the internal validity of the study. The subjects are diverse and complex enough to be representative of real-world scenarios, mitigating the external validity of the study by making the results generalizable. To make the study replicable, we made available online a link to our AXERAY tool and the anonymized participants' responses [26].

V. RELATED WORK

Accessibility attribute checkers. Existing approaches related to accessibility testing focus on checking syntactical attributes. Eler et al. [5] and Patil et al. [27] check for missing or wrong UI attributes in Android apps, such as missing alternative text attributes in images, or color attribute values below a certain threshold. Similar checks are also used in other tools such as Google's Accessibility Scanner [28], WAVE [29], and ASLint [30]. The aforementioned papers focus on syntactic checks, as opposed to the proposed approach in this work which checks for high-level aspects such as page structure and semantic landmarks, which are the most important ARIA roles that users with disabilities rely on [11].

Accessibility guidelines. The majority of existing work lies within the accessibility research community rather than software engineering. This research area involves studying certain categories of websites (e.g., airline websites [31], [32], education portals [33], other categories [34], [35], [36], [37]) or certain platforms (e.g., Android [38], [39], [40]), and then focusing on *manually observing* how non-sighted users would use those apps or websites in order to identify any patterns or trends in accessibility, with the purpose of publishing improved accessibility guidelines. Another line of work focuses on researching software development *best practices* and how do they impact the accessibility of the end product. For instance, Sanchez et al. [41] and Bai et al. [42], [43] examine development practices in agile teams working on accessible software, with the goal of proposing a guideline for better agile practices. Krainz et al. [44] investigates the impact of a model-driven approach to development on the accessibility of the created product. None of the aforementioned works, however, is concerned with developing an automated approach to test accessibility. Instead, their focus is researching best practices or guidelines for developers and designers.

Visual analysis. There exist a few techniques that analyze web applications from a visual perspective. Choudhary et al. [45] propose an approach that detects cross-browser compatibility by examining visual differences between the same app running in multiple browsers. Burg et al. [46] present a tool that helps developers understand the behavior of front-end apps. It allows developers to specify the element they are interested in, then tracks that element for any visual changes to understand code behavior. Bajammal et al. [47] propose an approach to generate reusable web components by analyzing design mockups. In contrast to our work, none of these works are related to accessibility.

VI. CONCLUSION

Software accessibility is the notion of building software that is usable by users with disabilities. Traditionally, software accessibility has often been an afterthought or a nice to have optional feature. However, software accessibility is increasingly becoming a legal requirement that must be satisfied. While some tools exist to perform basic forms of accessibility checks, they focus on syntactic checks, as opposed to checking the more critical high level semantic accessibility features that users with disabilities rely on. In this paper, we proposed an approach that automates web accessibility testing from a semantic perspective. It analyzes web pages using a combination of visual analysis, supervised machine learning, and natural language processing, and infers the semantic groupings present in the page and their semantic roles. It then asserts whether the page's markup matches the inferred semantics. We evaluated our approach on 30 real-world websites and assessed the accuracy of semantic inference as well as its ability to detect accessibility failures. The results show, on average, an F-measure of 87% for inferring semantic groupings, and an accessibility failures detection accuracy of 85%.

REFERENCES

[1] C. Vendome, D. Solano, S. Liñán, and M. Linares-Vásquez, "Can everyone use my app? an empirical study on accessibility in android

apps," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 41–52.

[2] S. Harper and A. Q. Chen, "Web accessibility guidelines," *World Wide Web*, vol. 15, no. 1, pp. 61–88, 2012.

[3] Title 42 of the United States Code, Section 12101, "The americans with disabilities act," https://www.law.cornell.edu/uscode/text/42/12101, 2020.

[4] Publications Office of the European Union, "Directive (eu) 2016/2102 of the european parliament on the accessibility of the websites and mobile applications of public sector bodies." https://eur-lex.europa.eu/eli/dir/2016/2102/oj, 2016.

[5] M. M. Eler, J. M. Rojas, Y. Ge, and G. Fraser, "Automated accessibility testing of mobile apps," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2018, pp. 116–126.

[6] SAP S.E., "Accessibility testing software automation tool," 2019, uS Patent 10,275,339.

[7] R. J. Breeds and J. S. Taylor, "Software accessibility testing," Mar. 4 2014, uS Patent 8,667,468.

[8] H. Braga, L. S. Pereira, S. B. L. Ferreira, and D. S. Da Silveira, "Applying the barrier walkthrough method: Going beyond the automatic evaluation of accessibility," *Procedia Computer Science*, vol. 27, pp. 471–480, 2014.

[9] N. L. Bayer and L. Pappas, "Accessibility testing: Case history of blind testers of enterprise software," *Technical Communication*, vol. 53, no. 1, pp. 32–38, 2006.

[10] United Kingdom Government's Office of Digital Services, "Accessibility tools audit." https://alphagov.github.io/accessibility-tool-audit/.

[11] WebAIM Non-profit Organization, "Screen reader user survey results." 2019. [Online]. Available: https://webaim.org/projects/screenreadersurvey8/

[12] A. Bai, H. C. Mork, T. Schulz, and K. S. Fuglerud, "Evaluation of accessibility testing methods. which methods uncover what type of problems?" *Studies in health technology and informatics*, vol. 229, pp. 506–516, 2016.

[13] P. Acosta-Vargas, S. Luján-Mora, T. Acosta, and L. Salvador-Ullauri, "Toward a combined method for evaluation of web accessibility," in *International Conference on Information Theoretic Security*. Springer, 2018, pp. 602–613.

[14] G. Brajnik, "A comparative test of web accessibility evaluation methods," in *Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility*. ACM, 2008, pp. 113–120.

[15] S. Abou-Zahra, "Web accessibility evaluation," in *Web accessibility*. Springer, 2008, pp. 79–106.

[16] Centers for Disease Control and Prevention, National Center on Birth Defects and Developmental Disabilities, Division of Human Development and Disability., "Disability and health data system (dhds) data." https://dhds.cdc.gov, 2016.

[17] European Union's Eurostat, "European union labour force survey (eu-lfs). prevalence of disability," http://appsso.eurostat.ec.europa.eu, 2014.

[18] Seyfarth Shaw Law Firm, LLP., "Americans with disabilities (ada) act, title iii report." https://www.seyfarth.com/news-publications, 2018.

[19] United States Courts, Court Records., "Public access to court electronic records (pacer)." https://www.uscourts.gov/court-records, 2020.

[20] "World wide web consortium. accessible rich internet applications 1.2. 2019."

[21] M. Nixon and A. Aguado, *Feature extraction and image processing for computer vision*. Academic Press, 2019.

[22] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," 2015.

[23] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[24] Eclipse Deeplearning4j Development Team, "Deeplearning4j: Open-source distributed deep learning for the jvm, apache software foundation license 2.0." [Online]. Available: http://deeplearning4j.org

[25] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Association for Computational Linguistics (ACL) System Demonstrations*, 2014, pp. 55–60. [Online]. Available: http://www.aclweb.org/anthology/P/P14/P14-5010

[26] https://github.com/msbajammal/icse2021. (2020)

[27] N. Patil, D. Bhole, and P. Shete, "Enhanced ui automator viewer with improved android accessibility evaluation features," in *2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT)*. IEEE, 2016, pp. 977–983.

[28] Google Inc., "Accessibility scanner." 2020. [Online]. Available: https://support.google.com/accessibility/android/answer/6376570

[29] WebAIM Non-profit Organization, "Web accessibility evaluation tool." [Online]. Available: https://wave.webaim.org/

[30] ASLint, "Accessibility linter." 2020. [Online]. Available: https://www.aslint.org/

[31] G. Agrawal, D. Kumar, M. Singh, and D. Dani, "Evaluating accessibility and usability of airline websites," in *International Conference on Advances in Computing and Data Sciences*. Springer, 2019, pp. 392–402.

[32] T. Domínguez Vila, E. Alén González, and S. Darcy, "Website accessibility in the tourism industry: an analysis of official national tourism organization websites around the world," *Disability and rehabilitation*, vol. 40, no. 24, pp. 2895–2906, 2018.

[33] R. Kimmons, "Open to all? nationwide evaluation of high-priority web accessibility considerations among higher education websites," *Journal of Computing in Higher Education*, vol. 29, no. 3, pp. 434–450, 2017.

[34] S. Bhagat and P. Joshi, "Evaluation of accessibility and accessibility audit methods for e-governance portals," in *Proceedings of the 12th International Conference on Theory and Practice of Electronic Governance*, 2019, pp. 220–226.

[35] A. S. Ross, X. Zhang, J. Fogarty, and J. O. Wobbrock, "Examining image-based button labeling for accessibility in android apps through large-scale analysis," in *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility*, 2018, pp. 119–130.

[36] S. Snider, W. L. Scott II, and S. Trewin, "Accessibility information needs in the enterprise," *ACM Transactions on Accessible Computing (TACCESS)*, vol. 12, no. 4, pp. 1–23, 2020.

[37] L. C. Serra, L. P. Carvalho, L. P. Ferreira, J. B. S. Vaz, and A. P. Freire, "Accessibility evaluation of e-government mobile applications in brazil," *Procedia Computer Science*, vol. 67, pp. 348–357, 2015.

[38] A. Alshayban, I. Ahmed, and S. Malek, "Accessibility issues in android apps: state of affairs, sentiments, and ways forward," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 1323–1334.

[39] K. Park, T. Goh, and H.-J. So, "Toward accessible mobile application design: developing mobile application accessibility guidelines for people with visual impairment," *Proceedings of HCI Korea*, pp. 31–38, 2014.

[40] S. Yan and P. Ramachandran, "The current status of accessibility in mobile apps," *ACM Transactions on Accessible Computing (TACCESS)*, vol. 12, no. 1, pp. 1–31, 2019.

[41] S. Sanchez-Gordon and S. Luján-Mora, "A method for accessibility testing of web applications in agile environments," in *Proceedings of the 7th World Congress for Software Quality (WCSQ). En proceso de publicación.(citado en la página 13, 15, 85)*, 2017.

[42] A. Bai, K. S. Fuglerud, R. A. Skjerve, and T. Halbach, "Categorization and comparison of accessibility testing methods for software development," 2018.

[43] A. Bai, V. Stray, and H. Mork, "What methods software teams prefer when testing web accessibility," *Advances in Human-Computer Interaction*, vol. 2019, 2019.

[44] E. Krainz, K. Miesenberger, and J. Feiner, "Can we improve app accessibility with advanced development methods?" in *International Conference on Computers Helping People with Special Needs*. Springer, 2018, pp. 64–70.

[45] S. R. Choudhary, M. R. Prasad, and A. Orso, "Crosscheck: Combining crawling and differencing to better detect cross-browser incompatibilities in web applications," in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. IEEE, 2012, pp. 171–180.

[46] B. Burg, A. J. Ko, and M. D. Ernst, "Explaining visual changes in web interfaces," in *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. ACM, 2015, pp. 259–268.

[47] M. Bajammal, D. Mazinanian, and A. Mesbah, "Generating reusable web components from mockups," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2018, pp. 601–611.