Understanding And Detecting Accessibility Issues In Web Applications

Yi Liu
Department of Computer Science and Engineering
Southern University of Science and Technology
Shenzhen, China
11610522@mail.sustech.edu.cn

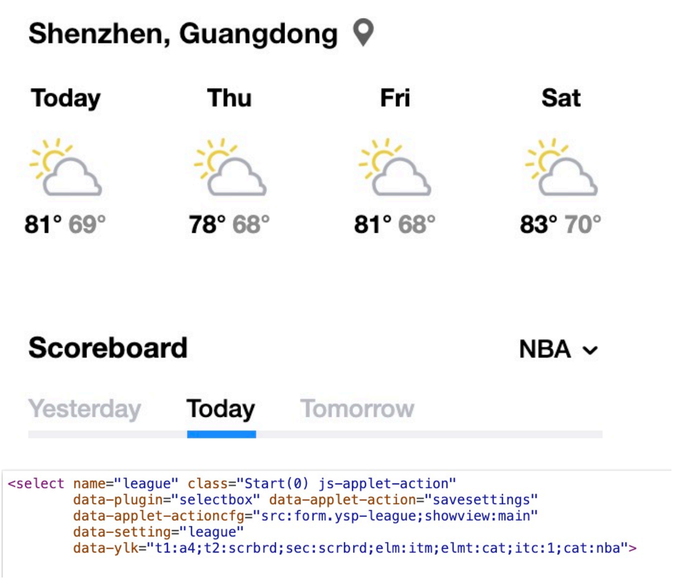
Hao Guan
Department of Computer Science and Engineering
Southern University of Science and Technology
Shenzhen, China
11612831@mail.sustech.edu.cn

***Abstract*—Mobile computing is eating the world. The quality of mobile applications affects user experience and the market share of the applications. Web applications, as a major type of applications in mobile computing era, play an important part in our daily life. Their usability is a critical concern. However, recent studies reported that many web applications have poor usability and suffer from various accessibility issues. Web applications with accessibility issues are thus not friendly to these users. Unfortunately, there is limited understanding on such issues. To bridge the gap, in this project, we will study real-world web applications to (1) investigate how serious the accessibility issues are, (2) learn the issue root cases, and (3) devise techniques to help developers improve the accessibility of their applications.**

***Index Terms*—Web Application, Accessibility , User Experience**

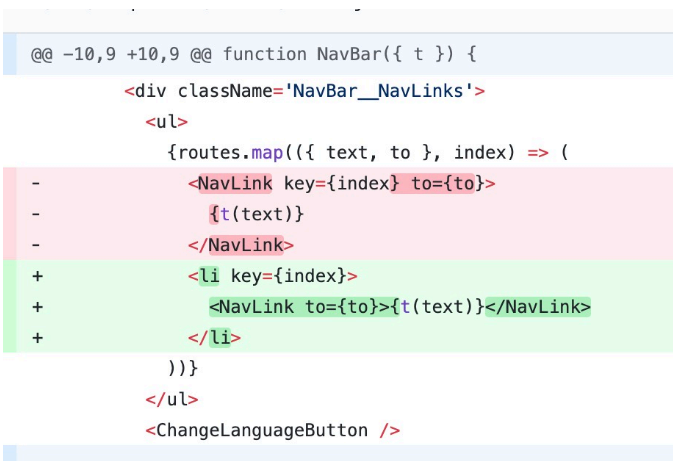
I. Introduction

Web accessibility is essential for both ethnic and legal reasons. According to the report from World Health Organization (WHO) [1], there is an estimated 180 million people worldwide who are visually disabled. Of these, between 40 and 45 million persons are blind and, by definition, cannot walk about unaided. They are usually in need of vocational and/or social support. With the blooming of the Internet, applications play an important role in bridging the user and content provider. However, there are a lot of restrictions prevent the visually disabled users from using applications fluently. Indeed, the experience of those users matters. Although, standards, like Web Content Accessibility Guidelines (WCAG) [2], have been developed purposed to give verified design patterns for web designers and best practices for web developers. Unfortunately,1) a large number of applications have been deployed and supported stable services in the real world, it will take a heavy workload to repair the web accessibility issues. 2) Due to the leak of concept for



```
<select name="league" class="Start(0) js-applet-action"
        data-plugin="selectbox" data-applet-action="savesettings"
        data-applet-actioncfg="src:form.ysp-league;showview:main"
        data-setting="league"
        data-ylk="t1:a4;t2:scrbrd;sec:scrbrd;elm:itm;elmt:cat;itc:1;cat:nba">
```

Fig. 1. The lack of form label observed from Yahoo.com



```
@@ -10,9 +10,9 @@ function NavBar({ t }) {
          <div className='NavBar__NavLinks'>
            <ul>
              {routes.map(({ text, to }, index) => (
-               <NavLink key={index} to={to}>
-                 {t(text)}
-               </NavLink>
+               <li key={index}>
+                 <NavLink to={to}>{t(text)}</NavLink>
+               </li>
              ))}
            </ul>
            <ChangeLanguageButton />
```

Fig. 2. The Accessibility Fixing in Real World

web content accessibility, developers still delivery code without accessibility enhance. 3) The web content accessibility issues vary, which will lead to a heavy workload to repair all these issues.

The screen reader [3], [4], known as a common assistant technology, can be used to support visually impaired users in most situations. However, such kinds of technologies based on the correct implementation of web applications. For example, the selector is shown in Figure 1 indicates the which scoreboard to display for the current user. Users depend on the screen readers will not able to distinguish the usage of the selector form due to the leak of form label. Since accessibility is a common problem [5]–[7], and concerned by developers shown in Figure 2, there are some tools supports the accessibility detecting. For example, a11y-checker [8] is a warn about HTML Markup code accessibility issue. Automated Accessibility Testing Tool (AATT) [9] is a browser-based accessibility testing tools and plugins require manually testing each page includes HTML CodeSniffer [10], Axe [11] and Chrome developer tool [12]. Besides, some tools are purposed for repairing the web content accessibility automatically [5], [13]. There are 32 common serious issues collected from the tools [8], [9], [11] listed in the Table I.

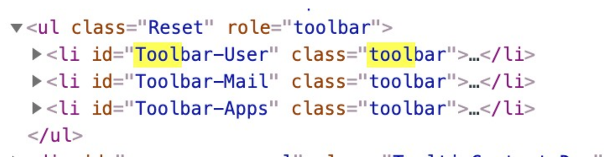
According to the issues listed in Table I, we can divide these issues into three parts.

- Attribute Losing: These patterns mainly infer that assistant information for assistant technologies have been lost. For example, the title of the page is missing, the title of the image tag is missing, etc.
- Invalid Attribute: The attributes of elements can vary but them still restricted by the standards [2]. Invalid attributes will be harmful to both browser rendering and assistant technologies recognizing.
- Unstructured Element: The structure of elements can be invalid for a DOM tree. They can lead to misunderstanding of for assistant technologies or message missing.
- Low contrast colors: The content of elements is shown in colors which are similar to the background color, which makes it hard for people with color vision deficiency to distinguish.

Detecting all types of patterns listed above is not an easy way. Even for the tools developed by google [14] still arise some false positive issues after

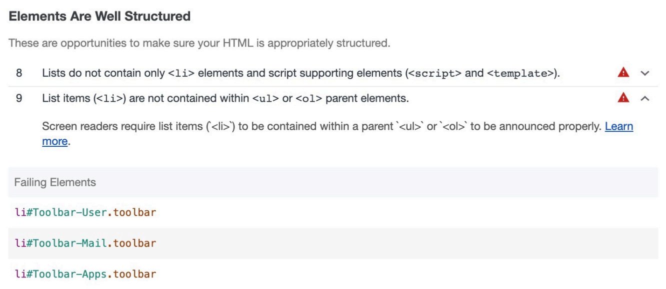
TABLE I
COMMON SERIOUS ISSUE PATTERNS IN ACCESSIBILITY DETECTING TOOLS

Rule ID	Impact
accesskeys	Serious
aria-hidden-focus	Serious
aria-roles	Serious
autocomplete-valid	Serious
blink	Serious
button-name	Serious
color-contrast	Serious
css-orientation-lock	Serious
definition-list	Serious
dlitem	Serious
document-title	Serious
duplicate-id-active	Serious
frame-title-unique	Serious
frame-title	Serious
html-has-lang	Serious
html-lang-valid	Serious
label-content-name-mismatch	Serious
label-title-only	Serious
layout-table	Serious
link-in-text-block	Serious
link-name	Serious
list	Serious
listitem	Serious
marquee	Serious
object-alt	Serious
p-as-heading	Serious
tabindex	Serious
table-fake-caption	Serious
td-headers-attr	Serious
th-has-data-cells	Serious
valid-lang	Serious



```
▼<ul class="Reset" role="toolbar">
  ▶<li id="Toolbar-User" class="toolbar">…</li>
  ▶<li id="Toolbar-Mail" class="toolbar">…</li>
  ▶<li id="Toolbar-Apps" class="toolbar">…</li>
 </ul>
```

Fig. 3. The element altered by Lighthouse in search.yahoo.com



Elements Are Well Structured

These are opportunities to make sure your HTML is appropriately structured.

8 Lists do not contain only `<li>` elements and script supporting elements (`<script>` and `<template>`).

9 List items (`<li>`) are not contained within `<ul>` or `<ol>` parent elements.

Screen readers require list items (`<li>`) to be contained within a parent `<ul>` or `<ol>` to be announced properly. Learn more.

Failing Elements

```
li#Toolbar-User.toolbar
li#Toolbar-Mail.toolbar
li#Toolbar-Apps.toolbar
```

Fig. 4. The issues altered by Lighthouse in search.yahoo.com

manually checking in Figure 4, 3. Besides, repairing the issues especially for Attribute Losing, Invalid Attribute is not trivial because most of them are related to the content, which can not understand by the program, of the website. Even though there are some tools [5] could fix the aria attributes based on trivial fixing. These will not so helpful for developers which possibly conducts misunderstanding for the assistant technologies.

There also lacks a standard to judge whether the colors are in high contrast or low contrast, which need to invite people with color vision deficiency to test the web page. In this paper, we refer to the CUD (Color Universal Design) introduced by CUDO, an organization from Japan committed to improve the experience of the people with cataract or color vision deficiency.

We introduce the *WARepair*, which combines the detecting and repairing for unstructured elements. To our best knowledge, we are the first one purposed the unstructured elements repairing tools for web content accessibility issues. We conduct the invariant element repair, which will fix the web content accessibility issues meanwhile not affect the appearance of the original display.

The technical challenges can be listed as the following. 1) The formalization of constriants is very difficult. 2) The detecting of unstructured elements in current tools is still not accurate.

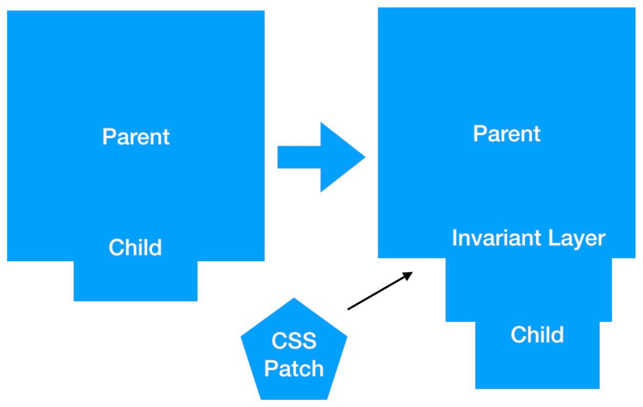


Fig. 5. The concept of Invariant Element Repair

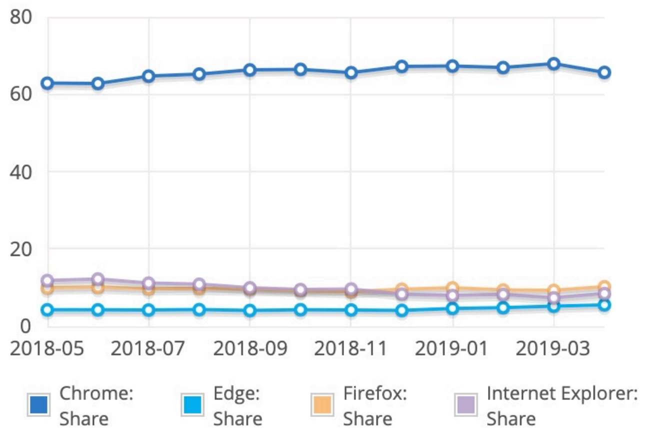


Fig. 6. The browser market share

II. Approaches

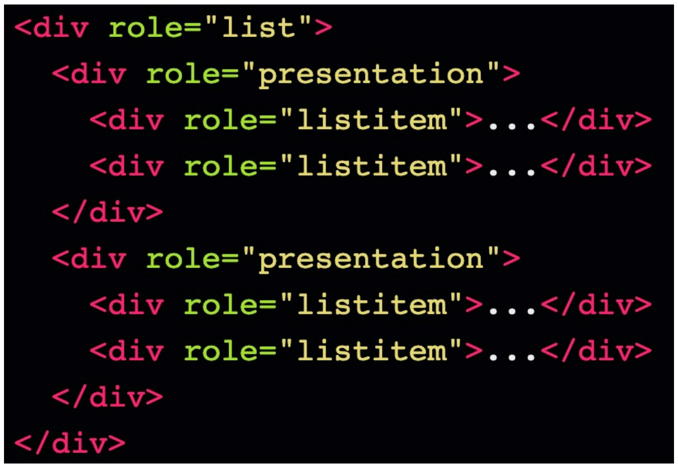
A. Naive Fixing Approaches

We study from the Github commits manually, and summarize the fix approaches into three.

*1) **unset Property**:*

a) Definition: The unset CSS keyword resets a property to its inherited value if it inherits from its parent, and to its initial value if not. In other words, it behaves like the inherit keyword in the first case, and like the initial keyword in the second case. It can be applied to any CSS property, including the CSS shorthand all [15].

b) Drawback: Unset property indeed can be applied to layout problems repairing and it performs well in most platform. However, the browser



```html
<div role="list">
  <div role="presentation">
    <div role="listitem">...</div>
    <div role="listitem">...</div>
  </div>
  <div role="presentation">
    <div role="listitem">...</div>
    <div role="listitem">...</div>
  </div>
</div>
```

Fig. 7. The listitem role in implementation

still has fragmentation problems. As the Figure 6 shown, Edge and Internet Explorer occupy Top 3 and Top 4 position in desktop browser market. Simply apply the *unset* property will still introduce layout problems even we take actions for the accessibility issues.

*2) **list-style-type Property**:*

a) Definition: The list-style-type CSS property sets the marker (such as a disc, character, or custom counter style) of a list item element [16].

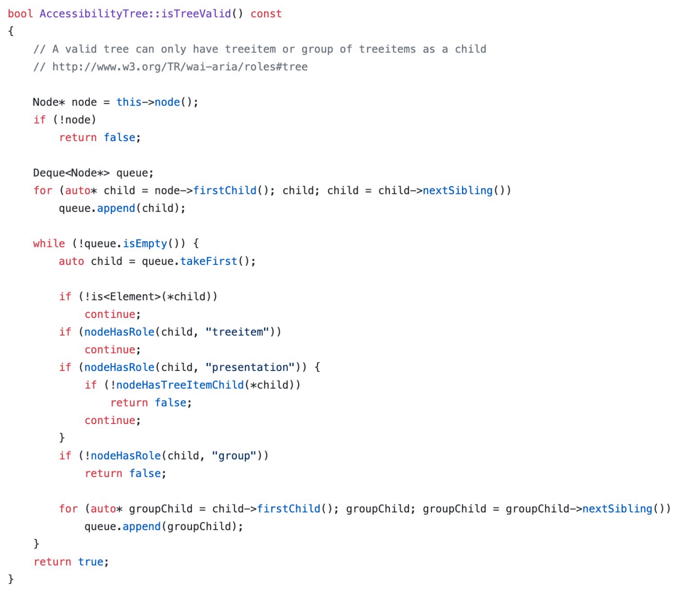
b) Drawback: By applying the property to inserted list item, it can simply refine the layout. However, we need additional CSS tricks to make it works for most screen reader. Even worse, due to maintain issues, the major assistant technology in Android does not support such kind of fixing [17].

*3) **list-item Role**:*

a) Definition: As shown in the Figure 7, any content that consists of an outer container with a list of elements inside it can be identified to assistive technologies using the list and listitem containers respectively [18].

b) Drawback: There are multiple existing studies and standards have warned developers to avoid using such kind of repairing technology [18], [19]. Due to the anti-patterns of best practices and maintain issues of delivered projects, we'd better keep away from such kind of repairing.

B. Unstructured Elements Category



```cpp
bool AccessibilityTree::isTreeValid() const
{
    // A valid tree can only have treeitem or group of treeitems as a child
    // http://www.w3.org/TR/wai-aria/roles#tree

    Node* node = this->node();
    if (!node)
        return false;

    Deque<Node*> queue;
    for (auto* child = node->firstChild(); child; child = child->nextSibling())
        queue.append(child);

    while (!queue.isEmpty()) {
        auto child = queue.takeFirst();

        if (!is<Element>(*child))
            continue;
        if (nodeHasRole(child, "treeitem"))
            continue;
        if (nodeHasRole(child, "presentation")) {
            if (!nodeHasTreeItemChild(*child))
                return false;
            continue;
        }
        if (!nodeHasRole(child, "group"))
            return false;

        for (auto* groupChild = child->firstChild(); groupChild; groupChild = groupChild->nextSibling())
            queue.append(groupChild);
    }
    return true;
}
```

Fig. 8. The partial code Accessibility Tree in Webkit

As shown in Figure 8, we analysis the source code Webkit [20], it generates the Accessibility Tree to enhance the assistant technology (like screen reader). However, the unstructured elements will bring information leaking and unexcepted logical expressions.

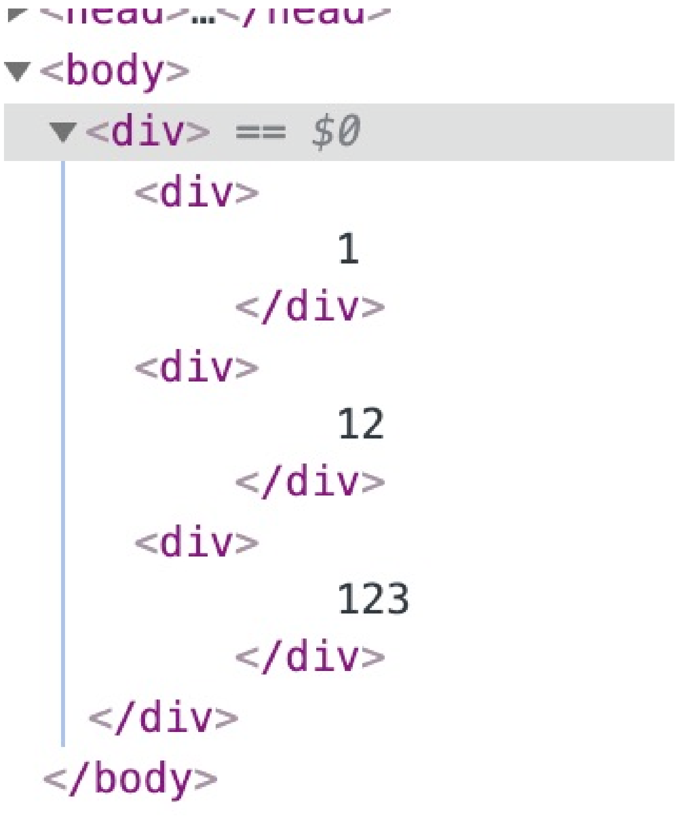
We divide unstructured elements anti-patterns we have found so far into two types : **Absence of elements**, **Misuse of elements**. All these anti-issues of unstructured elements shown above can be repaired by constraint solving.

1) Absence of elements: This issue happens when the leaking of proper structure elements. This kind of issue will introduce misunderstanding of page expression.

2) Misuse of elements: It is difficult to detect such kind of issues. We design two experiments to show the effects of such kind of issue. We implement a naive list shown in Figure 9, hence we get the Accessibility Tree of naive list in Figure 10. However, in the normal implementation (shown in Figure 9), we can simply get the logic structure of list.

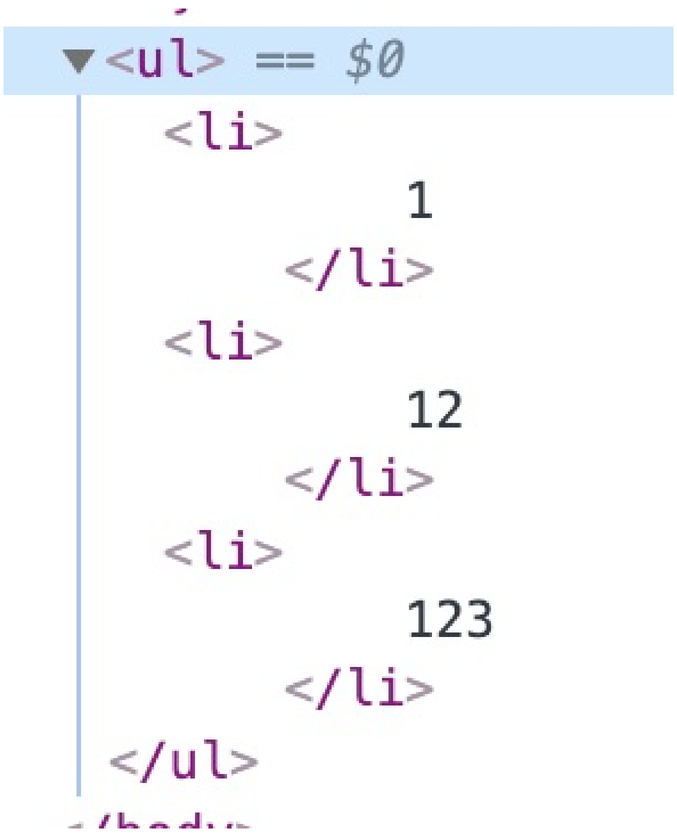
C. Constraint Solving

The Figure 5 refers to the way how can we fix the unstructured elements. Actually, most unstructured elements issues in the web content accessibility issues are the absence of proper elements. For example, the list elements should not contain other elements except the li and script elements. The



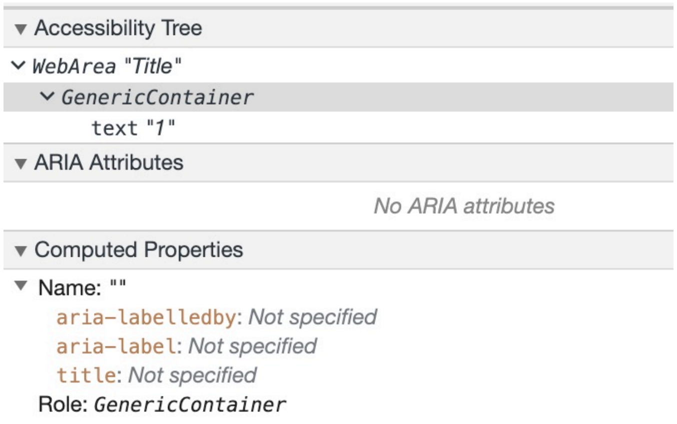
```
▶ <head>…</head>
▼ <body>
  ▼ <div> == $0
      <div>
              1
          </div>
      <div>
              12
          </div>
      <div>
              123
          </div>
    </div>
  </body>
```

Fig. 9. The naive list implementation



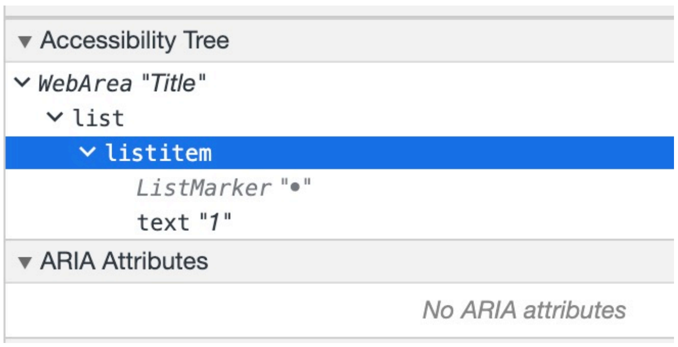
```
  ▼ <ul> == $0
        <li>
                  1
              </li>
        <li>
                  12
              </li>
        <li>
                  123
              </li>
      </ul>
```

Fig. 11. The normal list implementation



```
▼ Accessibility Tree
⌄ WebArea "Title"
  ⌄ GenericContainer
      text "1"
▼ ARIA Attributes
                  No ARIA attributes
▼ Computed Properties
▼ Name: ""
    aria-labelledby: Not specified
    aria-label: Not specified
    title: Not specified
  Role: GenericContainer
```

Fig. 10. The Accessibility Tree of Naive List in Figure 9



```
▼ Accessibility Tree
⌄ WebArea "Title"
  ⌄ list
    ⌄ listitem
        ListMarker "•"
        text "1"
▼ ARIA Attributes
                  No ARIA attributes
```

Fig. 12. The Accessibility Tree of Normal List in Figure 11

possible fix solution can be wrapped the invalid elements. The problem can be converted into the absence of the element. For situations like missing form label, it can be treated as one special case of element absence.

As the Figure 5 shown, we 1) obtains the properties of original parent and child elements. 2) Insert the invariant layer to obtain the new properties of child and invariant layer. 3) Minify the properties needed to be treated as constraints. 4) Solve the constraints and apply the CSS patch. Here we generate constraints by the CSS standard [21]. We transfer the problem into SAT problems.

There are numerous patterns of improper color combination. One of the most common situations is the foreground and background. Both colors can be too similar and hard to decide the content. To analyze the color quantitatively, we convert the RGB (red, green, blue) color model to HSL (hue, saturation, lightness) color model, which can make it more intuitive to describe the feelings of human's eye.

HSL is cylindrical geometry with hue, angular dimension, starting at the red primary at $0°$, passing through the green primary at $120°$ and the blue primary at $240°$, and then wrapping back to red at $360°$. In the geometry, the central vertical axis comprises the neutral, achromatic, or gray colors, ranging from black at lightness 0, the bottom, to white at lightness 1, the top.

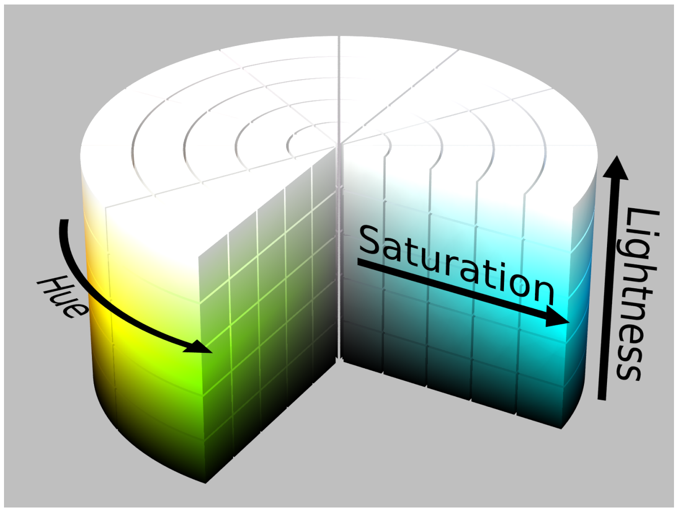


Fig. 13. HSL cylinder

To convertion from RGB to HSL is as follows:

- The R,G,B values are divided by 255 to change the range from $0\cdots255$ to $0\cdots1$:

$$\begin{aligned} R' &= R/255 \\ G' &= G/255 \\ B' &= B/255 \\ C_{\max} &= \max(R', G', B') \\ C_{\min} &= \min(R', G', B') \\ \Delta &= C_{\max} - C_{\min} \end{aligned} \tag{1}$$

- Hue calculation:

$$H = \begin{cases} 0° & , \Delta = 0 \\ 60° \times \left(\dfrac{G' - B'}{\Delta} \mod 6\right) & , C_{\max} = R' \\ 60° \times \left(\dfrac{B' - R'}{\Delta} + 2\right) & , C_{\max} = G' \\ 60° \times \left(\dfrac{R' - G'}{\Delta} + 4\right) & , C_{\max} = B' \end{cases} \tag{2}$$

- Saturation calculation:

$$S = \begin{cases} 0 & , \Delta = 0 \\ \dfrac{\Delta}{1 - |2L - 1|} & , \Delta \neq 0 \end{cases} \tag{3}$$

- Lightness calculation:

$$L = (C_{\max} - C_{\min})/2 \tag{4}$$

For two colors, if the hue and saturation values are the same and the difference of lightness is small, we can tell that the two colors are similar.

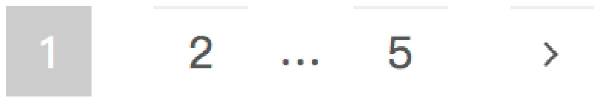


Fig. 14. An example of similar colors from Webpage

D. *Alternative Color Table*

Based on the work of Universal Color Design, we have a set of colors that can be used as a standard. When we find two colors are too similar, we can refer to the table and find the hue with the same offset.

E. *XPath*

XPath is a general way to express an element in DOM. In the browser we can find out the style of a specific XPath. And then recursively copy the whole style.

As shown in Figure 14, the '1' is $(255, 255, 255)$ in RGB, while the background is $(204, 204, 204)$. After convertion to HSL, they becomes $(0°, 0\%, 100\%)$ and $(0°, 0\%, 80\%)$, which only have slightly different in lightness.

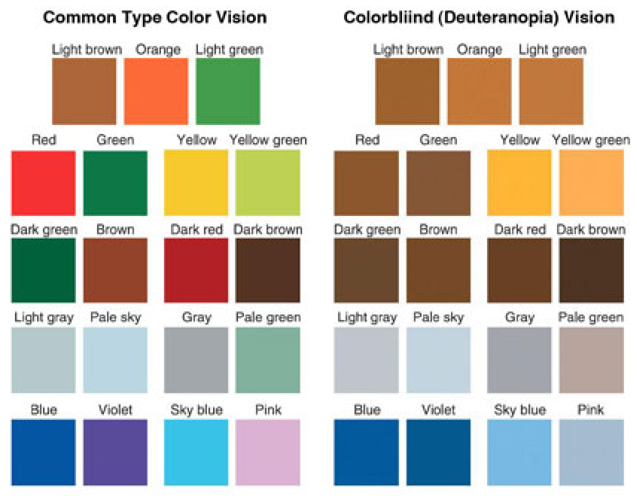
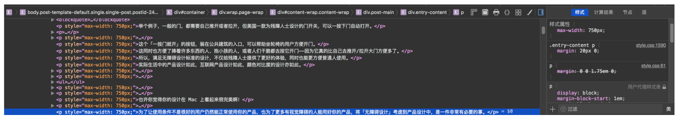


Fig. 15. Color vision table


E body.post-template-default.single.single-post.postid-24... 〉 E div#container 〉 E div.wrap.page-wrap 〉 E div#content-wrap.content-wrap 〉 E div.post-main 〉 E div.entry-content 〉 E p
样式 计算结果 节点 层
过滤
类

```
▶ <blockquote>…</blockquote>
  <p style="max-width: 750px;">举个例子，一般的门，都需要自己推开或者拉开，但美国一款为残障人士设计的门开关，可以一按下门自动打开。</p>
▶ <p>…</p>
▶ <p style="max-width: 750px;">…</p>
  <p style="max-width: 750px;">这个「一按门就开」的按钮，装在公共建筑的入口，可以帮助坐轮椅的用户方便开门。</p>
  <p style="max-width: 750px;">这同时也方便了捧着许多东西的人、抱小孩的人，或者人们干脆都去按它开门—因为它真的比自己去推开/拉开大门方便多了。</p>
  <p style="max-width: 750px;">所以，满足无障碍设计标准的设计，不仅给残障人士提供了更好的体验，同时也能更方便普通人使用。</p>
  <p style="max-width: 750px;">实际生活中的产品设计如此，互联网产品设计如此，颜色对比度的设计亦如此。</p>
▶ <p style="max-width: 750px;">…</p>
▶ <p style="max-width: 750px;">…</p>
▶ <ul>…</ul>
▶ <p style="max-width: 750px;">…</p>
▶ <p style="max-width: 750px;">…</p>
  <p style="max-width: 750px;">也许你觉得你的设计在 Mac 上看起来很完美啊！</p>
▶ <p style="max-width: 750px;">…</p>
  <p style="max-width: 750px;">为了让使用条件不是很好的用户仍然能正常使用你的产品，也为了更多有视觉障碍的人能用好你的产品，将「无障碍设计」考虑到产品设计中，是一件非常有必要的事。</p> = $0
```

```
样式属性 {
  max-width: 750px;
}
.entry-content p {                style.css:1590
  margin: 20px 0;
}
p {                               style.css:61
  ~~margin: 0 0 1.75em 0;~~
}
p {                               用户代理样式表
  display: block;
  margin-block-start: 1em;
```

Fig. 16. XPath example

F. *Compatible Constraints*

Due to the fragmentation of browser, we introduce the compatible constrains to ensure the compatibility of our patch. We make a spider to crawler data from MDN [22]. And then we build the knowledge base to guard the compatibility of patch.

III. Approach

We formalize the constraint rules according to the the rendering standard of browsers [21]. We divide the constraints into four categories shown as the following:

- **Inherited Constraint**: some attributes are directly inherited from the parent wrapper.
- **Static Constraint**: some attributes must be assigned directly.
- **Solvable Constraint**: some attributes have dependency on other
- **Compatible Constraint**: contextual information related to certain CSS attributes.

A. *Inherited Constraint*

There are multiple constraints directly inherits from the parent wrapper. When we calculate the inherited attributes. We can easily do intersection operation to get the inherited properties (Figure 17). We select some possible subset of *Inherited Constraint* shown in the Table III.

TABLE II
A subset of Inherited Constraint

Property
display
backgroud
color
font
float

TABLE III
A subset of Inherit Constraint

Property
border

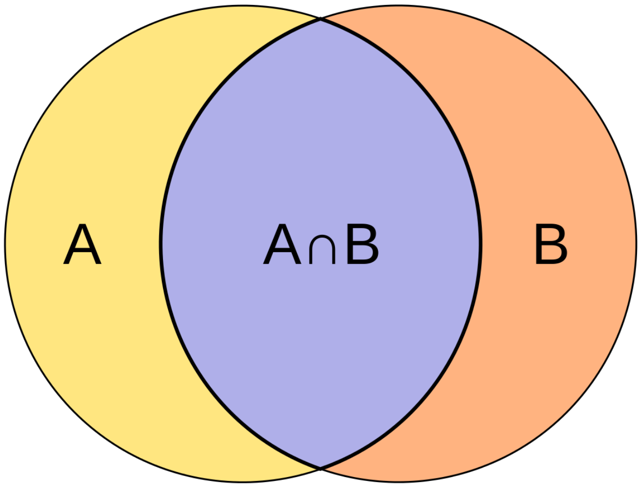


Fig. 17. Set Intersection Operation

B. *Static Constraint*

Due to the invariant of inserted wrapper, we can know some attributes will be assigned as a default value from domain knowledge learnt from browser rendering rules [21]. Here we simply give a subset of *Static Constraint* shown as Table.

C. *Solvable Constraint*

With the purpose of keep the appearance of original web page, we have to make sure that the inserted element has no side effect on the appearance. However, multiple attributes might introduce a coupling scene which affects each other together. So we have to formalize them into unique rules and make them solve by the constraint solver. Here we use the example of box model to calculate the primitive attributes of inserted element.

$$(margin-left)_{insected} + (margin-right)_{insected} \\ +width_{insected} + (padding-left)_{parent} \\ +(padding-right)_{parent} = originalwidth$$

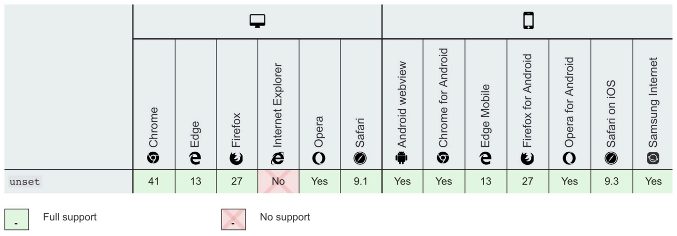
$$(margin-top)_{insected} + (margin-bottom)_{insected} \\ +height_{insected} + (padding-top)_{parent} \\ +(padding-bottom)_{parent} = originalheight$$

$$(padding-top)_{insected} + (padding-bottom)_{insected} \\ +height_{child} + (margin-top)_{child} \\ +(margin-bottom)_{child} = height_{insect}$$

$$(padding-left)_{insected} + (padding-right)_{insected} \\ +width_{child} + (margin-left)_{child} \\ +(margin-right)_{child} = width_{insect}$$

D. Compatible Constraint

With the fragmentation problem in browsers, we have to introduce *Compatible Constraint* to meet the gap of various browsers. We build knowledge base for attributes supporting status in different browsers from MDN [21]. One example shown in the Figure 18.



	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Android webview	Chrome for Android	Edge Mobile	Firefox for Android	Opera for Android	Safari on iOS	Samsung Internet
unset	41	13	27	No	Yes	9.1	Yes	Yes	13	27	Yes	9.3	Yes

🟩 Full support 🟥 No support

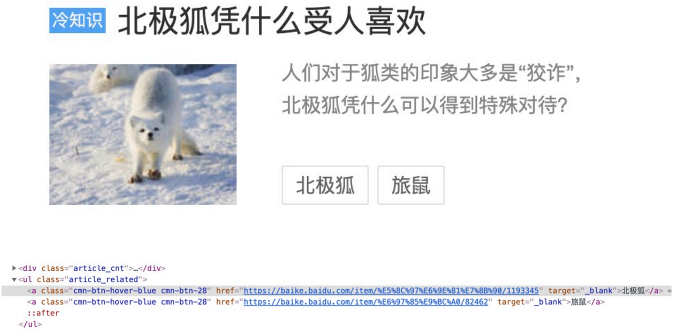
Fig. 18. Unset Attribute Compatible Status

IV. Evaluation and Case Study

A. Data Collection

We choose two rules of the unstructured elements issues in Table I. We randomly pick up two to five words from the Webster dictionary and using Bing search API to obtain the top ten results as the data set. We have evaluated 117308 pages and found 352699 issues.

B. Case Study on Baidu


冷知识 北极狐凭什么受人喜欢
人们对于狐类的印象大多是“狡诈”，
北极狐凭什么可以得到特殊对待?
北极狐
旅鼠

```
▸<div class="article_cnt">…</div>
▾<ul class="article_related">
   <a class="cmn-btn-hover-blue cmn-btn-28" href="https://baike.baidu.com/item/%E5%8C%97%E6%9E%81%E7%8B%90/1193345" target="_blank">北极狐</a> =
   <a class="cmn-btn-hover-blue cmn-btn-28" href="https://baike.baidu.com/item/%E6%97%85%E9%BC%A0/82462" target="_blank">旅鼠</a>
   ::after
 </ul>
```

Fig. 19. The Original Display and Dom of Baidu Baike

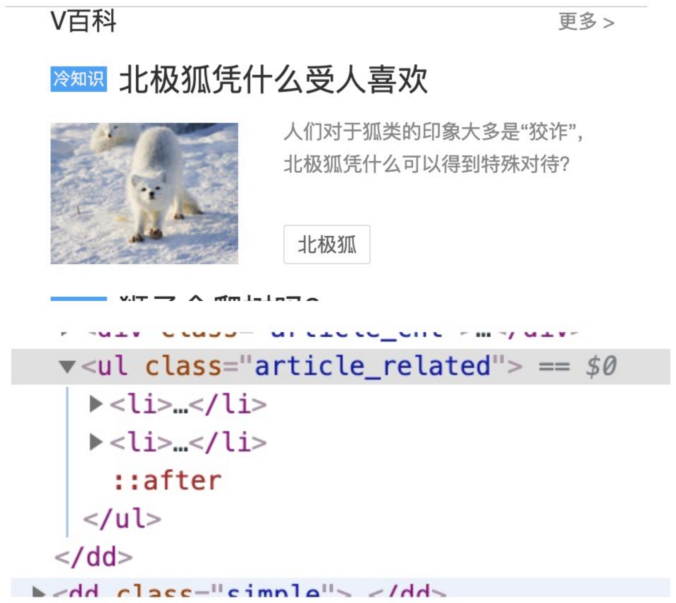
The Figure 19 shows the original display and content accessibility issues. The $< a >$ tag is invalid directly contained in the $< ul >$ tag.

The Figure 20 shows the display after naive fixing by simply wrapping the $< a >$ tag with $< li >$ tag. This fixing arises no web content accessibility issues but breaks the original appearance.

The Figure 21 shows the display after real fixing by simply wrapping the $< a >$ tag with $< li >$ tag by adding attribute $display : inline - block$. This fixing arises no web content accessibility issues also keeps the original appearance.

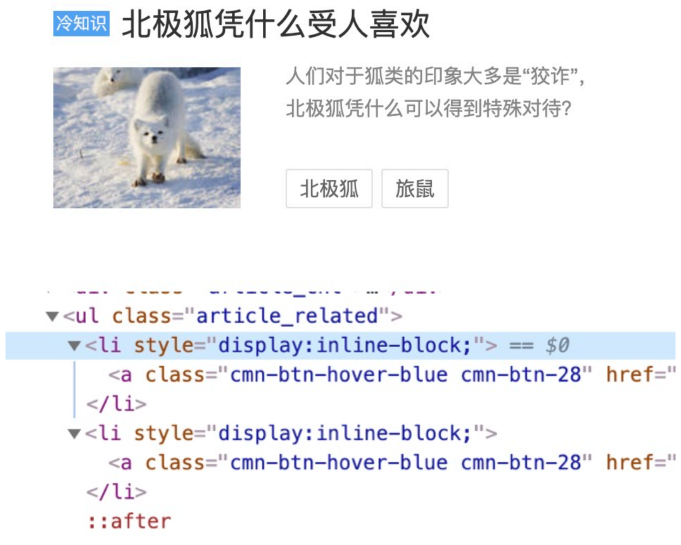
C. Case Study on Hexo Theme NeXT

Hexo is a popular framework to generate blog web pages, and NeXT is the most commonly used theme (color style, layout, etc.) in China. Figure 14 which is previously mentioned is the page selector style in NeXT. Figure 22 - 24 show the repairing processure.


V百科
更多 >
冷知识 北极狐凭什么受人喜欢
人们对于狐类的印象大多是“狡诈”，
北极狐凭什么可以得到特殊对待?
北极狐

```
▼<ul class="article_related"> == $0
  ▶<li>…</li>
  ▶<li>…</li>
    ::after
  </ul>
</dd>
```

Fig. 20. The Display and Dom of Baidu Baike After Naive Fixing


冷知识 北极狐凭什么受人喜欢
人们对于狐类的印象大多是“狡诈”，
北极狐凭什么可以得到特殊对待?
北极狐
旅鼠

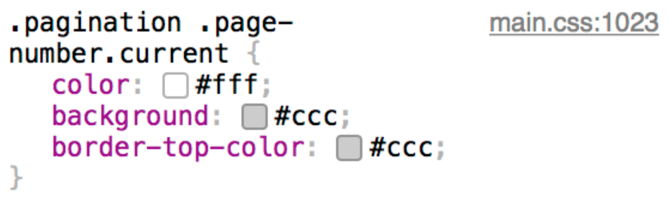
```
▼<ul class="article_related">
  ▼<li style="display:inline-block;"> == $0
      <a class="cmn-btn-hover-blue cmn-btn-28" href="
    </li>
  ▼<li style="display:inline-block;">
      <a class="cmn-btn-hover-blue cmn-btn-28" href="
    </li>
    ::after
```

Fig. 21. The Display and Dom of Baidu Baike After Real Fixing

V. Practical case study

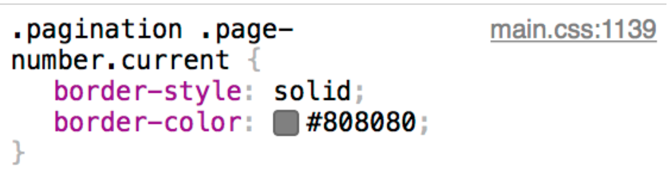
A. Voice Captcha

1) Introduction: Captcha is a common component in web pages, to ensure that the user is a real person. While most captchas are in form of figures, which is not friendly to vision inpaired people. Alternatively, there are some websites supporting voice captcha, such as Apple and Google.



```
.pagination .page-                     main.css:1023
number.current {
   color: ⬜#fff;
   background: ⬜#ccc;
   border-top-color: ⬜#ccc;
}
```

Fig. 22. The original CSS of low contrast color



```
.pagination .page-                     main.css:1139
number.current {
   border-style: solid;
   border-color: ⬛#808080;
}
```

Fig. 23. CSS with higher contrast to override

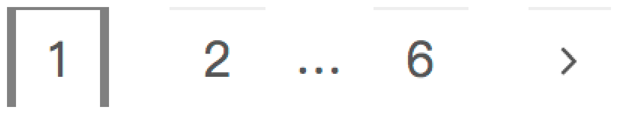


Fig. 24. Effect after repairing

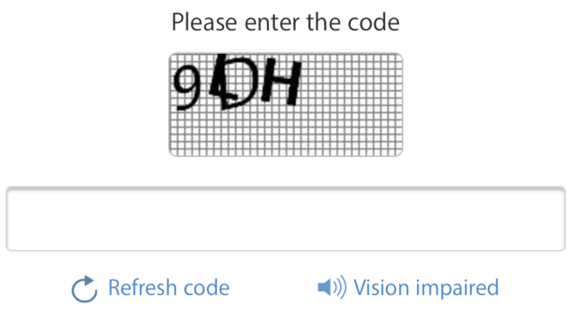


Fig. 25. Apple's Captcha

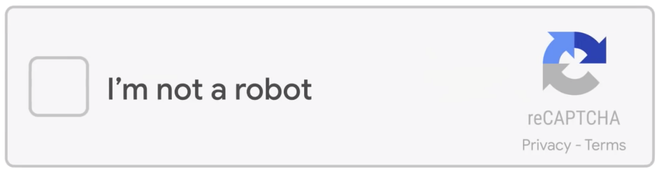


Fig. 26. Google's reCAPTCHA

However most websites still uses figure-only captchas.

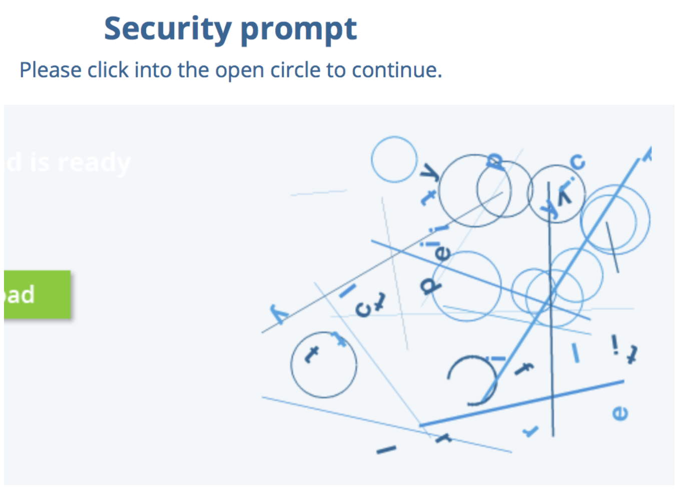
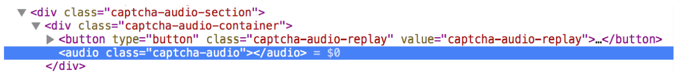


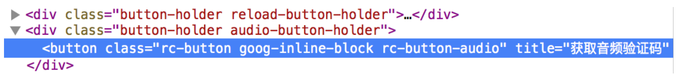
Fig. 27. Figure-only Captcha

2) Detection - Using Lighthouse: Lighthouse is a tool set for detecting web page problems. We can add a rule to judge whether the website supports voice captcha.



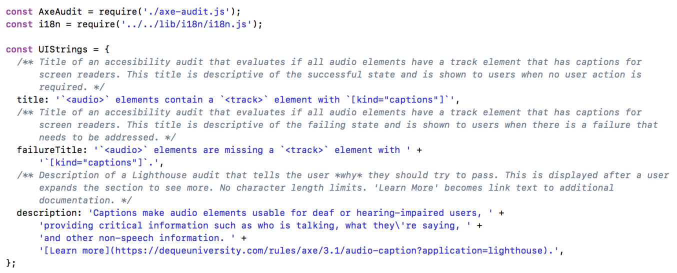
```
▼ <div class="captcha-audio-section">
  ▼ <div class="captcha-audio-container">
    ▶ <button type="button" class="captcha-audio-replay" value="captcha-audio-replay">…</button>
      <audio class="captcha-audio"></audio> = $0
    </div>
```

Fig. 28. Apple's code



```
▶ <div class="button-holder reload-button-holder">…</div>
▼ <div class="button-holder audio-button-holder">
    <button class="rc-button goog-inline-block rc-button-audio" title="获取音频验证码"
  </div>
```

Fig. 29. Google's code



```
const AxeAudit = require('./axe-audit.js');
const i18n = require('../../lib/i18n/i18n.js');

const UIStrings = {
  /** Title of an accesibility audit that evaluates if all audio elements have a track element that has captions for
      screen readers. This title is descriptive of the successful state and is shown to users when no user action is
      required. */
  title: '`<audio>` elements contain a `<track>` element with `[kind="captions"]`',
  /** Title of an accesibility audit that evaluates if all audio elements have a track element that has captions for
      screen readers. This title is descriptive of the failing state and is shown to users when there is a failure that
      needs to be addressed. */
  failureTitle: '`<audio>` elements are missing a `<track>` element with ' +
      '`[kind="captions"]`.',
  /** Description of a Lighthouse audit that tells the user *why* they should try to pass. This is displayed after a user
      expands the section to see more. No character length limits. 'Learn More' becomes link text to additional
      documentation. */
  description: 'Captions make audio elements usable for deaf or hearing-impaired users, ' +
      'providing critical information such as who is talking, what they\'re saying, ' +
      'and other non-speech information. ' +
      '[Learn more](https://dequeuniversity.com/rules/axe/3.1/audio-caption?application=lighthouse).',
};
```

Fig. 30. Custom rule for Lighthouse

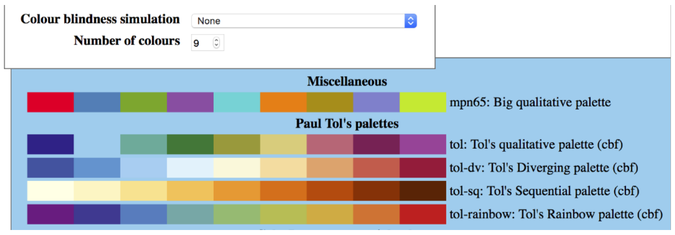
3) Continuous integration: We can also add the Lighthouse evaluation into test cases when doing continuous integration:

```
$ lighthouse --only-categories=
  ↪ accessibility  --output=json
  ↪ --output-path=./report.json
  ↪ --save-assets <url>
```

B. Color palette generator

C. Steps

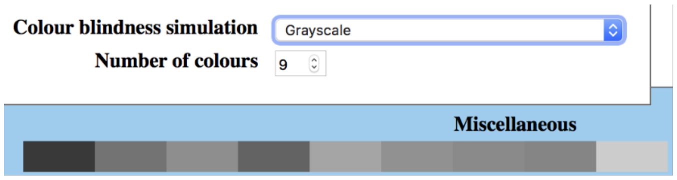
- Detect color problems with Lighthouse
- Classify
  - Low contrast for text: change text color
  - Color match problem: generate a new set
- New palette: google/palette.js
- Give options



Colour blindness simulation: None
Number of colours: 9

Miscellaneous
mpn65: Big qualitative palette
Paul Tol's palettes
tol: Tol's qualitative palette (cbf)
tol-dv: Tol's Diverging palette (cbf)
tol-sq: Tol's Sequential palette (cbf)
tol-rainbow: Tol's Rainbow palette (cbf)

Fig. 31. Results from palette.js



Colour blindness simulation: Grayscale
Number of colours: 9

Miscellaneous

Fig. 32. Simulating grey palette

D. Export API

```
var sol_base = ['002b36', '073642',
  ↪ '586e75', '657b83', '839496'
  ↪ , '93a1a1', 'eee8d5', 'fdf6e3
  ↪ '];
```

```
var seq = palette('sequential', 10)
  ↪ ;
var cbf = palette('sequential-cbf',
  ↪ 10);
```

```
var pal_for_queries = palette(['
  ↪ sequential'], 10, 0);
var pal_for_errors = palette(['
  ↪ sequential'], 10, 1);
var pal_for_latency = palette(['
  ↪ sequential'], 10, 2);
```

VI. Related Work

There has been some work in the literature trying to detecting and repairing the web accessibility issues. But most of them are focused on the trivial repairing part. To our best knowledge, we are the first one to conduct the unstructured element repairing. The work was done by Nádia Fernandes [5] attempts to fix invalid RIA attributes. The repairing of this is trivial by simply adding a un-closed tag or missing attributes. This might lead to a misunderstanding of assistant technologies like a screen reader. The Pavel Panchekha and Adam T. Geller's work gives an extensible framework to detect layout problems with different display parameters. Rules can be defined by domain-specific language in their framework. But they just providing assertions for the layout displayed rather the fix patch.

VII. Conclusion

In this work, we first implement an extensible detecting and repairing framework *WARepair*for unstructured element repairing. The detection part in our framework outperform the google lighthouse [14] in some specific cases by the false positive rate. To our best knowledge, we are the first purposed to formalize the invariant repair constraints and transform it into SAT problems. Our work might reduce the heavy workload for developers in fixing such accessibility issues and benefit millions of visually impaired users.

References

[1] "Blindness: Vision 2020 - the global initiative for the elimination of avoidable blindness." https://www.who.int/mediacentre/factsheets/fs213/en/.

[4] "Get started on android with talkback - android accessibility help." https://support.google.com/accessibility/android/answer/6283677?hl=en.

[2] "Web content accessibility guidelines (wcag) overview." https://www.w3.org/WAI/standards-guidelines/wcag/.

[3] "Vision accessibility - mac - apple (ae)." https://www.apple.com/ae/accessibility/mac/vision/.

[5] N. Fernandes, "Towards web accessibility repair," in *International Cross-Disciplinary Conference on Web Accessibility, W4A '13, Rio de Janeiro, Brazil, May 13-15, 2013*, pp. 9:1–9:2, 2013.

[6] X. Zhang, A. S. Ross, A. Caspi, J. Fogarty, and J. O. Wobbrock, "Interaction proxies for runtime repair and enhancement of mobile application accessibility," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Denver, CO, USA, May 06-11, 2017.*, pp. 6024–6037, 2017.

[7] S. Mirri, L. A. Muratori, and P. Salomoni, "Monitoring accessibility: large scale evaluations at a geo political level," in *The 13th International ACM SIGACCESS Conference on Computers and Accessibility, ASSETS '11, Dundee, Scotland, UK, October 24-26, 2011*, pp. 163–170, 2011.

[8] "A11y checker." https://github.com/Muhnad/a11y-checker.

[9] "Automated accessibility testing tool (aatt)." https://github.com/paypal/AATT.

[10] "Html_codesniffer." http://squizlabs.github.io/HTML_CodeSniffer/.

[11] "Axe." https://github.com/dequelabs/axe-core.

[12] "Accessibility developer tools." https://github.com/GoogleChrome/accessibility-developer-tools.

[13] A. Jasselette, M. Keita, M. Noirhomme-Fraiture, F. Randolet, J. Vanderdonckt, C. V. Brussel, and D. Grolaux, "Automated repair tool for usability and accessibility of web sites," in *Computer-Aided Design Of User Interfaces V, Proceedings of the Sixth International Conference on Computer-Aided Design of User Interfaces, CADUI 2006 6-8 June 2006, Bucharest, Romania*, pp. 261–272, 2006.

[14] "Googlechrome lighthouse." https://github.com/GoogleChrome/lighthouse.

[15] "Unset." https://developer.mozilla.org/en-US/docs/Web/CSS/unset.

[16] "list-style-type." https://developer.mozilla.org/en-US/docs/Web/CSS/list-style-type.

[17] "Aria lists." https://developer.mozilla.org/en-US/docs/Web/CSS/Reference.

[18] "List item." https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Roles/Listitem_role.

[19] "Aria lists." https://www.scottohara.me/blog/2018/05/26/aria-lists.html.

[20] "Webkit." https://github.com/WebKit/webkit.

[21] "Cascading style sheets level 2 revision 1 (css 2.1) specification." https://www.w3.org/TR/2011/REC-CSS2-20110607/.

[22] "Css reference." https://github.com/WebKit/webkit.