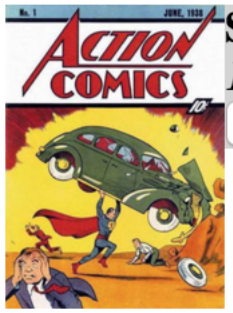
Detecting Redundant CSS Rules in HTML5 Applications: A Tree Rewriting Approach

Anthony W. Lin (with Matthew Hague and Luke Ong)

Yale-NUS College, Singapore
Royal Holloway, University of London, UK
Oxford University, UK

Introduction to Webpages

Super Heroes



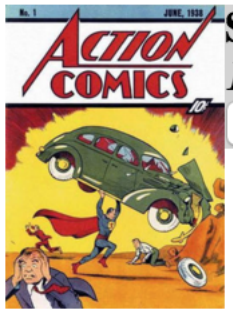
Superman
1938
Delete

```
<html>
<head>
    <title>Super Heroes</title>
    <style>
        .side-img { float: left; height: 3cm;}
        .info { background-color: lightgray; }
        .info > .name { font-weight: bold; }
        .info > .date { font-style: italic; }
    </style>
</head>
<body>
    <h1>Super Heroes</h1>

    <img class="side-img" src="superman.jpg"/>

    <div class="info">
        <div class="name">Superman</div>
        <div class="date"/>
    </div>

    <button class="del">Delete</button>

    <script src="jquery-1.9.0.min.js"></script>
    <script>
        $(document).ready(function() {
            // add date
            $(".info > .date").html("1938");

            // remove if clicked
            $('.del').click(function(e){
                $('.side-img').remove();
                $('.info').remove();
                $('.del').remove();
            });
        });
    </script>
</body>
</html>
```

A webpage has three main components...

Introduction to Webpages

Super Heroes



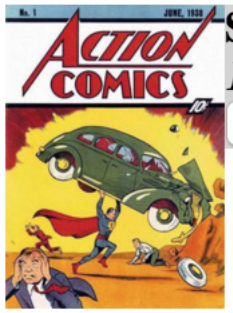
Superman
1938
Delete

```
<body>
    <h1>Super Heroes</h1>

    <img class="side-img"
         src="superman.jpg"/>

    <div class="info">
        <div class="name">Superman</div>
        <div class="date"/>
    </div>

    <button class="del">Delete</button>
</body>
```

The Document Object Model (DOM) tree.

Introduction to Webpages

Super Heroes



Superman
1938
Delete

```
<script>
    $(document).ready(function() {
        // add date
        $(".info > .date").html("1938");

        // remove if clicked
        $('.del').click(function(e){
            $('.side-img').remove();
            $('.info').remove();
            $('.del').remove();
        });
    });
</script>
```

A dynamic / scripting component.

Introduction to Webpages

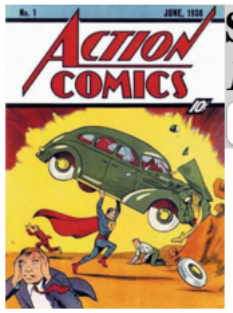
Super Heroes

```
<script>
    $(document).ready(function() {
        // add date
        $(".info > .date").html("1938");

        // remove if clicked
        $('.del').click(function(e){
            $('.side-img').remove();
            $('.info').remove();
            $('.del').remove();
        });
    });
</script>
```

A dynamic / scripting component.

Introduction to Webpages

Super Heroes



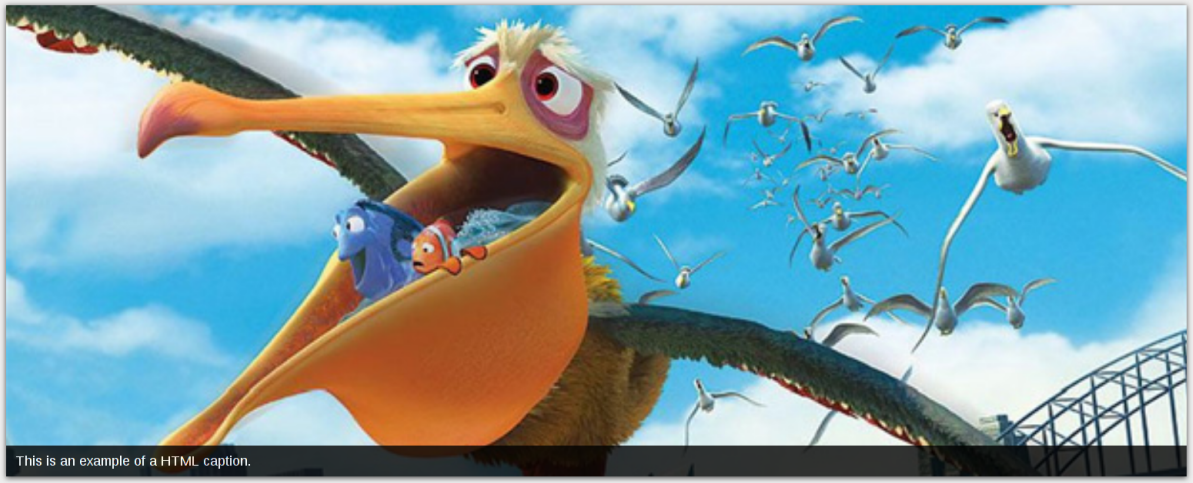
Superman
1938
Delete

```
<style>
    .side-img { float: left; }
    .info { background-color: lightgray; }
    .info > .name { font-weight: bold; }
    .info > .date { font-style: italic; }
</style>
```

The CSS component.

The CSS Redundancy Problem

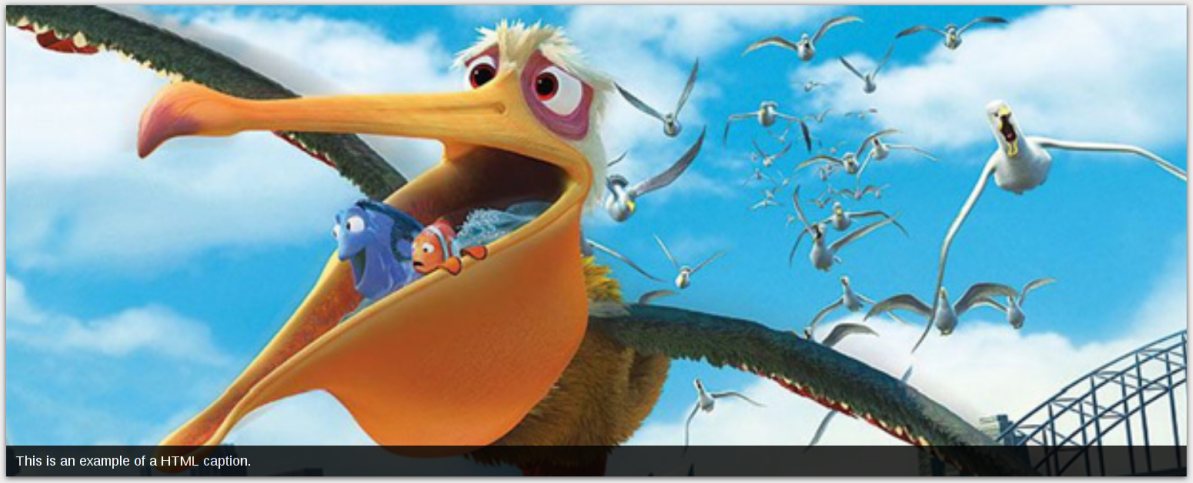
CSS Stylesheets can become very large.



- Usual development bloat (e.g. plugins with generic stylesheets).
  - E.g. Nivo-Slider has 172 selectors, 131 redundant in demo above.

The CSS Redundancy Problem

CSS Stylesheets can become very large.



- Usual development bloat (e.g. plugins with generic stylesheets).
  - E.g. Nivo-Slider has 172 selectors, 131 redundant in demo above.

60% of them are redundant... [Mesbah and Mirshokraie]

- 30% of rendering time is spent on selectors [Meyerovich and Bodik].

Existing Solutions

Existing tools for cleaning CSS are quite limited:

- Cilla [Mesbah and Mirshokraie] and UnCSS [Martino]
  - Explores as much of a page as it can.
  - Reports which selectors were not used.
  - Unsound.

Existing Solutions

Existing tools for cleaning CSS are quite limited:

- Cilla [Mesbah and Mirshokraie] and UnCSS [Martino]
  - Explores as much of a page as it can.
  - Reports which selectors were not used.
  - Unsound.

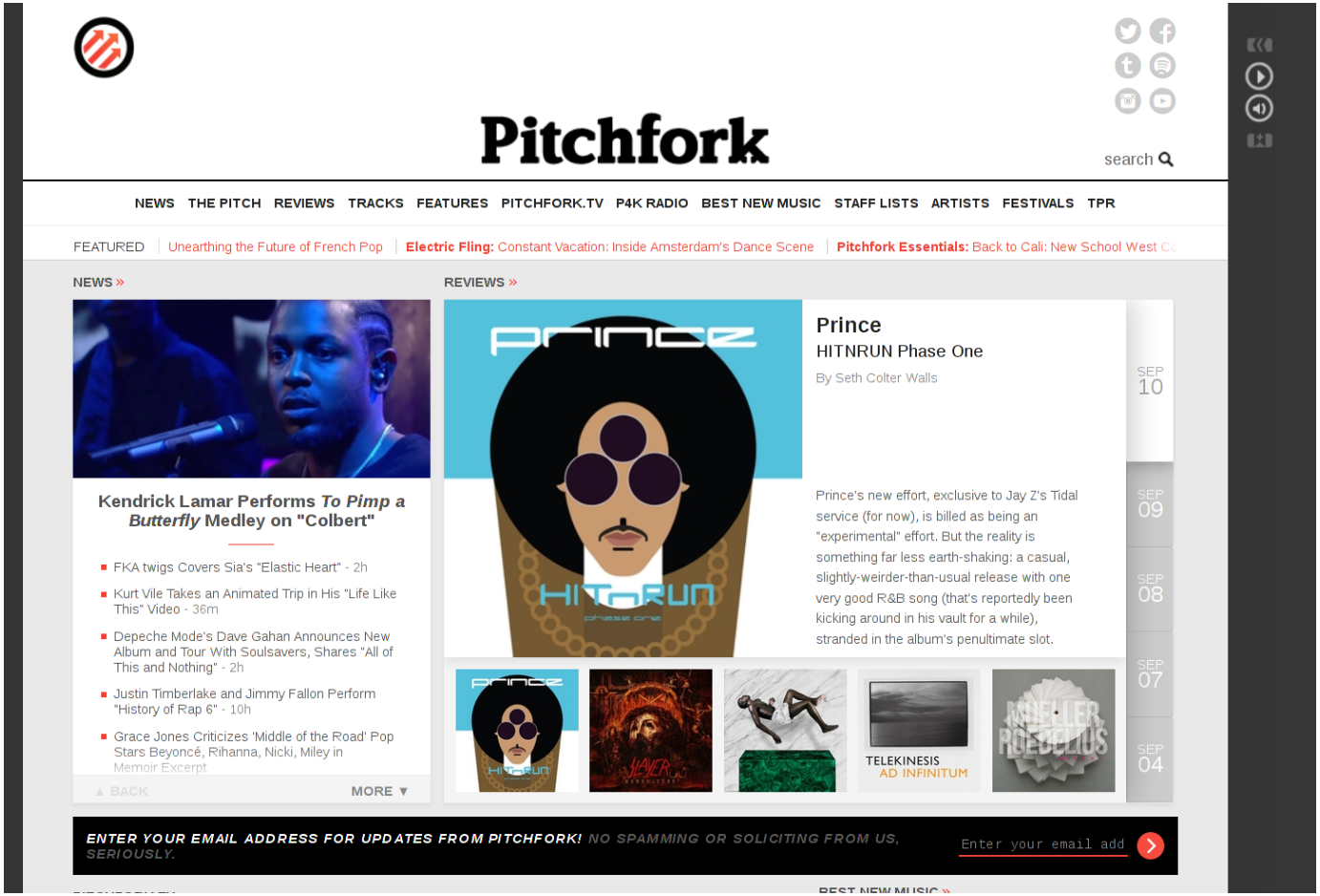
Both of these approaches can remove useful CSS rules.

- This can ruin the look of your website.

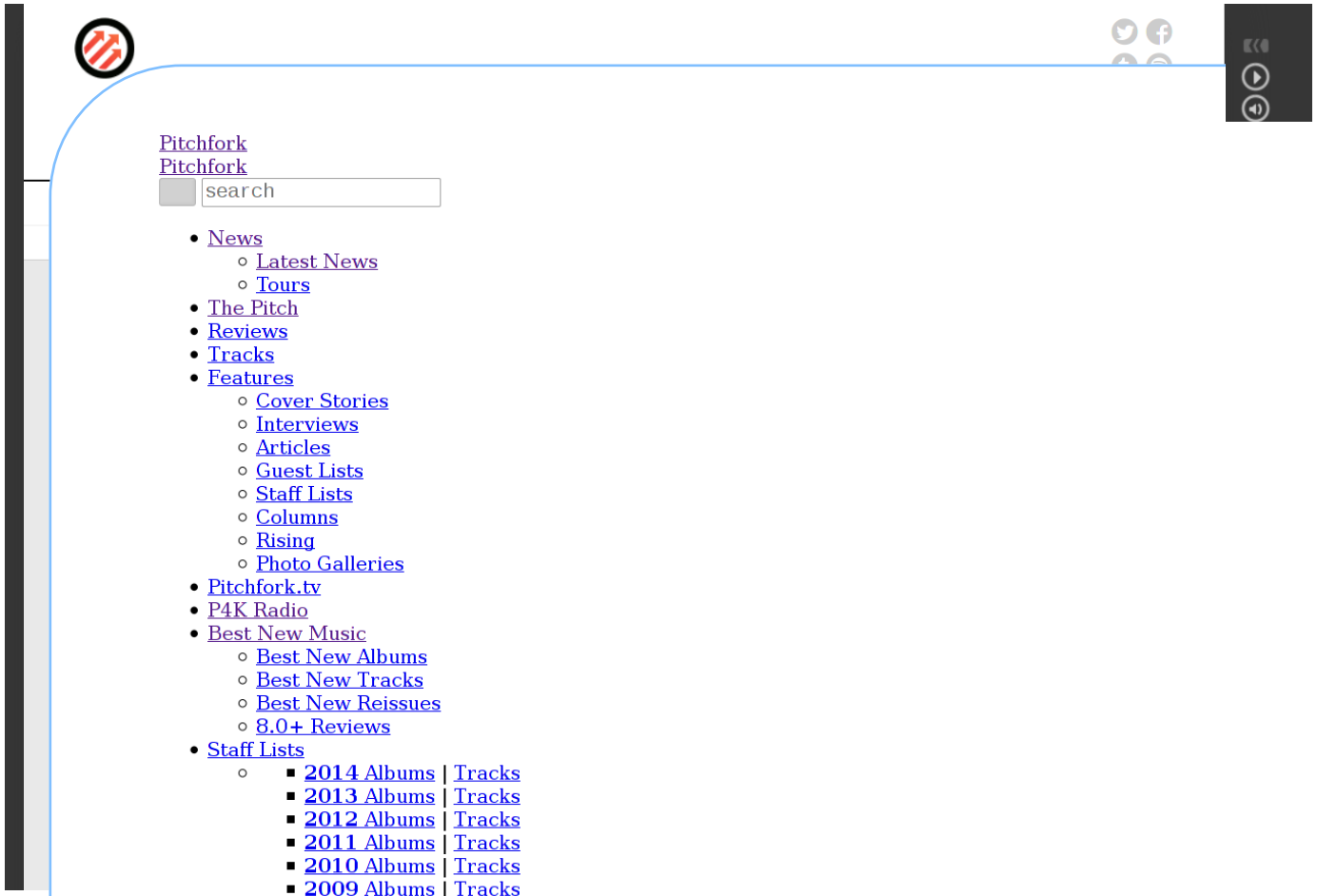
Existing Solutions

Ex

Bo



Existing Solutions



Pitchfork
Pitchfork
search

- News
  - Latest News
  - Tours
- The Pitch
- Reviews
- Tracks
- Features
  - Cover Stories
  - Interviews
  - Articles
  - Guest Lists
  - Staff Lists
  - Columns
  - Rising
  - Photo Galleries
- Pitchfork.tv
- P4K Radio
- Best New Music
  - Best New Albums
  - Best New Tracks
  - Best New Reissues
  - 8.0+ Reviews
- Staff Lists
  - **2014** Albums | Tracks
  - **2013** Albums | Tracks
  - **2012** Albums | Tracks
  - **2011** Albums | Tracks
  - **2010** Albums | Tracks
  - **2009** Albums | Tracks
  - **2008** Albums | Tracks
  - **2007** Albums | Tracks
  - **2006** Albums | Tracks
  - **2005** Albums | Singles
  - **2004** Albums | Singles
  - **2003** Albums | Singles
  - **2002** Albums

Existing Solutions

Existing tools for cleaning CSS are quite limited:

- Cilla [Mesbah and Mirshokraie] and UnCSS [Martino]
  - Explores as much of a page as it can.
  - Reports which selectors were not used.
  - Unsound.

Both of these approaches can remove useful CSS rules.

- This can ruin the look of your website.

Existing Solutions

Existing tools for cleaning CSS are quite limited:

- Cilla [Mesbah and Mirshokraie] and UnCSS [Martino]
  - Explores as much of a page as it can.
  - Reports which selectors were not used.
  - Unsound.

Both of these approaches can remove useful CSS rules.

- This can ruin the look of your website.
- And break functionality (UnCSS breaks Nivo-Slider).

Static Analysis?

Static Analysis?

- Leading tools: WALA and TAJS

Static Analysis?

- Leading tools: WALA and TAJS
- Using them to identify redundant CSS is challenging.

Static Analysis?

- Leading tools: WALA and TAJS
- Using them to identify redundant CSS is challenging.
- Problem 1: Call graph of JS+jQuery is hard to construct

Static Analysis?

- Leading tools: WALA and TAJS
- Using them to identify redundant CSS is challenging.
- Problem 1: Call graph of JS+jQuery is hard to construct
- Problem 2: DOM tree is not precisely tracked

Static Analysis?

- Leading tools: WALA and TAJS
- Using them to identify redundant CSS is challenging.
- Problem 1: Call graph of JS+jQuery is hard to construct
- Problem 2: DOM tree is not precisely tracked

See *Andreasen and Moller'14* for an up-to-date survey on static analysis of JavaScript+jQuery.

Our Contributions

Clean tree-rewriting model of DOM dynamics due to JS.

- Precise
- Automatic analysis (via reduction to symbolic pushdown systems)

Our Contributions

Clean tree-rewriting model of DOM dynamics due to JS.

- Precise
- Automatic analysis (via reduction to symbolic pushdown systems)

Proof-of-concept translation from HTML5 to our model.

- Models jQuery functions in how they modify the DOM.
- Does not yet support many features of JavaScript.

Our Contributions

Clean tree-rewriting model of DOM dynamics due to JS.

- Precise
- Automatic analysis (via reduction to symbolic pushdown systems)

Proof-of-concept translation from HTML5 to our model.

- Models jQuery functions in how they modify the DOM.
- Does not yet support many features of JavaScript.

Promising experimental results.

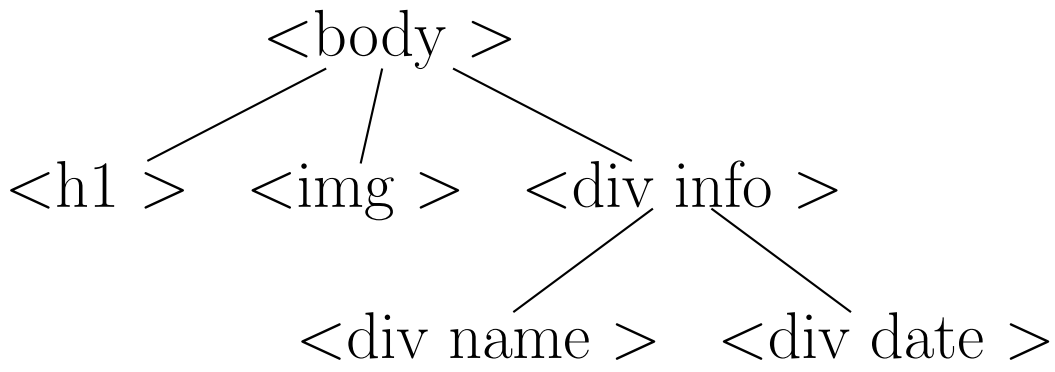
Our approach (in a nutshell)

- Aim to overapproximate DOM dynamics with tree-rewriting model $\mathcal{R}$

Our approach (in a nutshell)

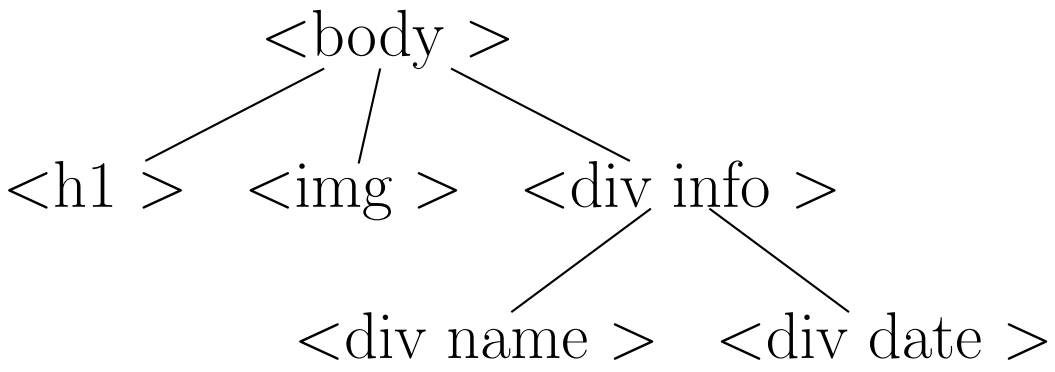
- Aim to overapproximate DOM dynamics with tree-rewriting model $\mathcal{R}$
- Redundant node selectors in $\mathcal{R} \Rightarrow$ redundant in HTML5

Our Tree-Rewriting Model: Domain



DOM is abstracted as an (unordered) unranked tree with class labels.

Our Tree-Rewriting Model: Domain



DOM is abstracted as an (unordered) unranked tree with class labels.

HTML elements and node IDs are treated as constant classes

Our Tree-Rewriting Model: Rewrite Rules

A rewrite rule is a pair (g, χ), where:

- g is a "guard" (a.k.a. "node selector"): modal logic formula with modalities $\langle\uparrow\rangle, \langle\uparrow^+\rangle, \langle\downarrow\rangle, \langle\downarrow^+\rangle$,
- χ is a rewrite operation: AddClass, AddChild, RemoveClass, RemoveNode.

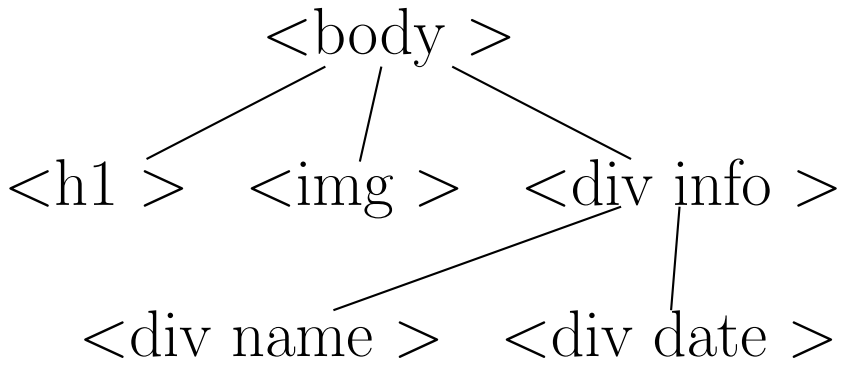
Our Tree-Rewriting Model: Rewrite Rules

A rewrite rule is a pair (g, χ), where:

- g is a "guard" (a.k.a. "node selector"): modal logic formula with modalities $\langle\uparrow\rangle, \langle\uparrow^+\rangle, \langle\downarrow\rangle, \langle\downarrow^+\rangle$,
- χ is a rewrite operation: AddClass, AddChild, RemoveClass, RemoveNode.

A tree-rewrite system is a finite set of rewrite rules.

Examples



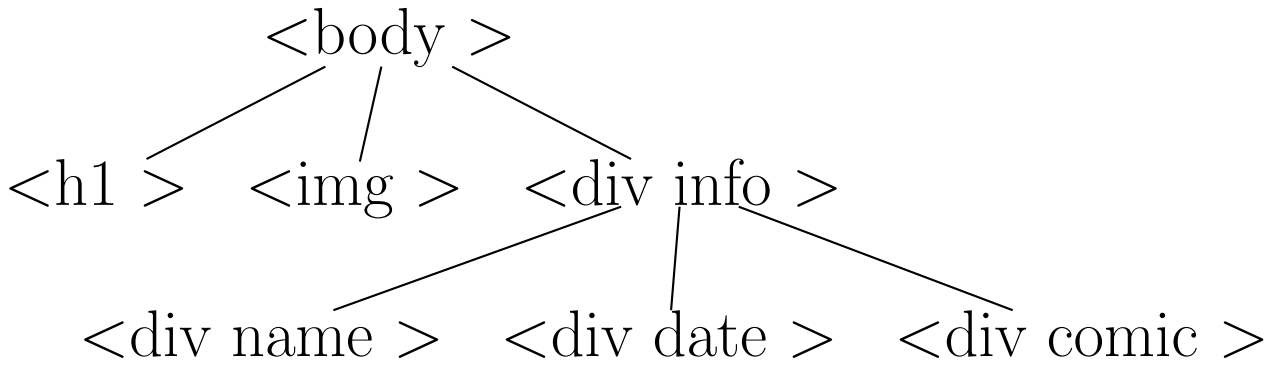
A jQuery line

```
$(".info").appendChild("<div class='comic'>DC Comics</div>")
```

is represented by

$$(\texttt{info}, \text{AddChild}(\texttt{div comic}))$$

Examples



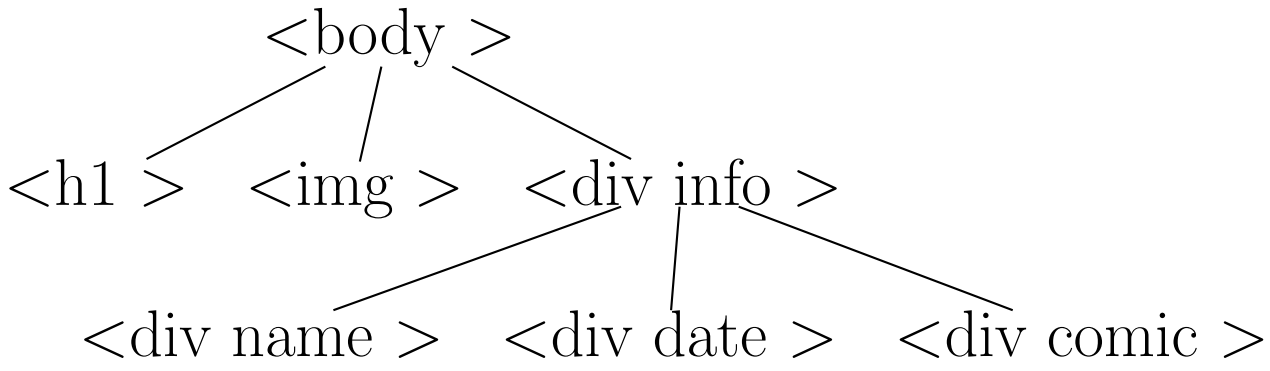
A jQuery line

```
$(".info").appendChild("<div class='comic'>DC Comics</div>")
```

is represented by

$$(\texttt{info}, \text{AddChild}(\texttt{div comic}))$$

Examples



A CSS Selector

```
.info > .date { font-style: italic; }
```

can be represented by

$$(\texttt{date} \wedge \langle\uparrow\rangle \texttt{info}, \mathrm{AddClass}(\texttt{cssrule1}))$$

Operational Semantics of TRS

$T_1 \rightarrow_{\mathcal{R}} T_2$ if T_1 can be rewritten into T_2

i.e. $\exists$ a node v in T_1 where some rule (g, χ) in $\mathcal{R}$ can be "fired".

Redundancy Problem

INPUT : a TRS $\mathcal{R}$, an initial DOM tree T,
and a S set of node selectors

QUESTION : Identify selectors in S that cannot be matched
(in all reachable trees)

Solving the Redundancy Problem

Q: given a tree, rules and classes, which classes are redundant?

Solving the Redundancy Problem

Q: given a tree, rules and classes, which classes are redundant?

CSS redundancy is undecidable in general

Solving the Redundancy Problem

Q: given a tree, rules and classes, which classes are redundant?

CSS redundancy is undecidable in general

In practice, suffices to restrict to positive guards

Solving the Redundancy Problem

Q: given a tree, rules and classes, which classes are redundant?

CSS redundancy is undecidable in general

In practice, suffices to restrict to positive guards

e.g. $\texttt{date} \wedge \langle\uparrow\rangle \texttt{info}$

Solving the Redundancy Problem

Q: given a tree, rules and classes, which classes are redundant?

CSS redundancy is undecidable in general

In practice, suffices to restrict to positive guards

e.g. $\texttt{date} \wedge \langle\uparrow\rangle \texttt{info}$

Theorem: The redundancy problem for positive TRS is efficiently reducible to analysis of symbolic PDS, which is EXP-complete, but for which fast solvers exist.

The Reduction

Theorem: The redundancy problem for positive TRS is efficiently reducible to analysis of symbolic PDS, which is EXP-complete, but for which fast solvers exist.

- One boolean variable for each class

The Reduction

Theorem: The redundancy problem for positive TRS is efficiently reducible to analysis of symbolic PDS, which is EXP-complete, but for which fast solvers exist.

- One boolean variable for each class
- Stack used in a "depth-first search" of the tree.

The Reduction

Theorem: The redundancy problem for positive TRS is efficiently reducible to analysis of symbolic PDS, which is EXP-complete, but for which fast solvers exist.

- One boolean variable for each class
- Stack used in a "depth-first search" of the tree.
- New nodes can be created by pushing onto the stack.

The Reduction

Theorem: The redundancy problem for positive TRS is efficiently reducible to analysis of symbolic PDS, which is EXP-complete, but for which fast solvers exist.

- One boolean variable for each class
- Stack used in a "depth-first search" of the tree.
- New nodes can be created by pushing onto the stack.
- Backtrack by popping.

The Reduction

Theorem: The redundancy problem for positive TRS is efficiently reducible to analysis of symbolic PDS, which is EXP-complete, but for which fast solvers exist.

- One boolean variable for each class
- Stack used in a "depth-first search" of the tree.
- New nodes can be created by pushing onto the stack.
- Backtrack by popping.
- (Since guards are positive, we can always recreate nodes.)

Implementation

We implemented a tool TreePed to test the approach.

- A rough tool for extracting rules from jQuery script.
- jMoped used as a pushdown backend.

Tested on a number of examples (next slide).

- Reasonable run times.
- Identified all and only redundant rules.
- Two real-world examples, five made up examples.

Results Table

Case Study	Ns	Ss	Ls	Rs	Time
`bikes.html`	22	18 (0)	97	37	3.6s
`comments.html`	5	13 (1)	43	26	2.9s
`example.html`	11	1 (0)	28	4	.6s
`example-up.html`	8	1 (1)	15	3	.6s
`igloo/`		261 (89)			3.4s
`index.html`	145		24	1	
`engineering.html`	236		24	1	
`Nivo-Slider/`					
`demo.html`	15	172 (131)	501	21	6.3s
`transactions.html`	19	9 (0)	37	6	1.6s

Ns — # of HTML elements in the initial tree

Ss — # of CSS rules (redundant CSS rules)

Ls — # of lines of JavaScript (cloc)

Rs — # of rules extracted from JS

Summary and Future Work

Web pages are dynamic programs:

- Manipulate a tree data structure (DOM).
- Can be modelled by tree rewrite systems.
- Model-checking can be used for optimising CSS rules.

Future work:

- Systematically extracting rewrite rules from JavaScript.
  - JavaScript analysis is hard! (c.f. Moller et al)