Reasoning with Style

Pierre Genevès (CNRS)

Joint work with:

Martí Bosch (Universitat Politècnica de Catalunya)

Nabil Layaïda (Inria)

IJCAI, July 30th 2015, Buenos Aires

Style

Brief History of World Wide Web:

- Initial foundations of the web (1991): HTML, HTTP, URL
- First added feature (1996): Cascading Style Sheets (CSS) for describing the visual layout (style) of HTML pages
- Since 2009, integration of HTML 5 + CSS 3 + Javascript allows advanced (Flash-less) web graphic designs

CSS Today:

- Increasingly important component of web user experience
- Visual layout consumes 40–70% of the processing time of the web browser [Meyerovich et al. WWW'10]

CSS: Very Simple at First Sight

- Everything is about setting pairs (property, value) for HTML elements
- HTML elements are identified using a selector:

$$
\begin{array}{l}
\textit{selector}\ \{ \\
\quad \textit{property}_1\ :\ \textit{value}_1\ ; \\
\quad \ldots \\
\quad \underbrace{\textit{property}_n\ :\ \textit{value}_n\ ;}_{\text{declaration}} \\
\}
\end{array}
$$

```
1 div {
2     color: red;
3     margin: 10px;
4     padding: 5px;
5     text-align: left
6 }
```

Zoom on CSS

Several features combined make the situation more complex

- Inheritance: values may be propagated recursively
- Cascade: properties may be set by rules from various origins (e.g. other stylesheets)
- Priorities: for a given property, rules have different priorities (specificity vectors)
- Expressive selector language, e.g.: `div p.contents` $\sim$ `img`
- Pseudo-classes, e.g.: `:nth-child(even)`

Consequences

- A single CSS style sheet can be written in a variety of different ways
- Real-world CSS files often contain redundant/unnecessary code declarations
- For each element of each page view, the browser must apply the most specific rule (huge source of inefficiency)

Academic Example

```
a {
   color: blue;
   text-decoration: underline
}

ul.menu a {
   color: blue;
   text-decoration: none
}

ul.menu a:nth-child(odd), ul.menu a:nth-child(even) {
   color: blue;
   text-decoration: none;
   font-weight: bold
}

ul.menu a:nth-child(odd) {
   border: 1px
}
```

Equivalent Example

```
1  a {
2     color: blue;
3     text-decoration: underline
4  }
5
6  ul.menu a {
7     text-decoration: none;
8     font-weight: bold
9  }
10
11 ul.menu a:nth-child(odd) {
12    border: 1px
13 }
```

- Totally equivalent (renders all documents exactly like previous code)
- only 3 rules and 5 declarations (instead of 4 rules and 8 declarations)
- 128 bytes instead of 228 (56%, stats on compressed representations)

→ faster rendering by the web browser (download, parsing and formatting)

Back to First Example

- Some redundancies can "easily" be removed:

```
1  a {
2    color: blue;
3    text-decoration: underline
4  }
5
6  ul.menu a {
7    color: blue;
8    text-decoration: none
9  }
10
11 ul.menu a:nth-child(odd), ul.menu a:nth-child(even) {
12   color: blue;
13   text-decoration: none;
14   font-weight: bold
15 }
16
17 ul.menu a:nth-child(odd) {
18   border: 1px
19 }
```

Back to First Example

- Some redundancies can "easily" be removed:

```
1  a {
2      color: blue;
3      text-decoration: underline
4  }
5                  S6
6  ul.menu a {
7      color: blue;
8      text-decoration: none
9  }
10                                              S11
11 ul.menu a:nth-child(odd), ul.menu a:nth-child(even) {
12     color: blue;
13     text-decoration: none;
14     font-weight: bold
15 }
16
17 ul.menu a:nth-child(odd) {
18     border: 1px
19 }
```

- Some equivalent forms are less "easy" to deduce
  → comparing selectors (e.g. $s_{11} \subseteq s_6$ and $s_6 \subseteq s_{11}$) and merging declarations

Can we generalize and compute this automatically?

Approach

1. Identify safe CSS refactoring opportunities:

   - Rules with semantically equivalent selectors
   - Inactive declarations
     (when $S_i \subseteq S_j$ and S_j gets preference, `p:v` under S_i is masked by `p:v'` under S_j)
   - Redundant declarations
     (when $S_i \subset S_j$, S_i gets preference, and `p:v` under both S_i and S_j, then `p:v` under S_i is unnecessary)
   - Empty rules (produced by refactoring)

Approach

1. Identify safe CSS refactoring opportunities:

   - Rules with semantically equivalent selectors → merge
   - Inactive declarations
     (when $S_i \subseteq S_j$ and S_j gets preference, `p:v` under S_i is masked by `p:v'` under S_j) → delete `p:v`
   - Redundant declarations
     (when $S_i \subset S_j$, S_i gets preference, and `p:v` under both S_i and S_j, then `p:v` under S_i is unnecessary) → delete `p:v` under S_i
   - Empty rules (produced by refactoring) → remove

2. Apply corresponding rewrite rules

Approach

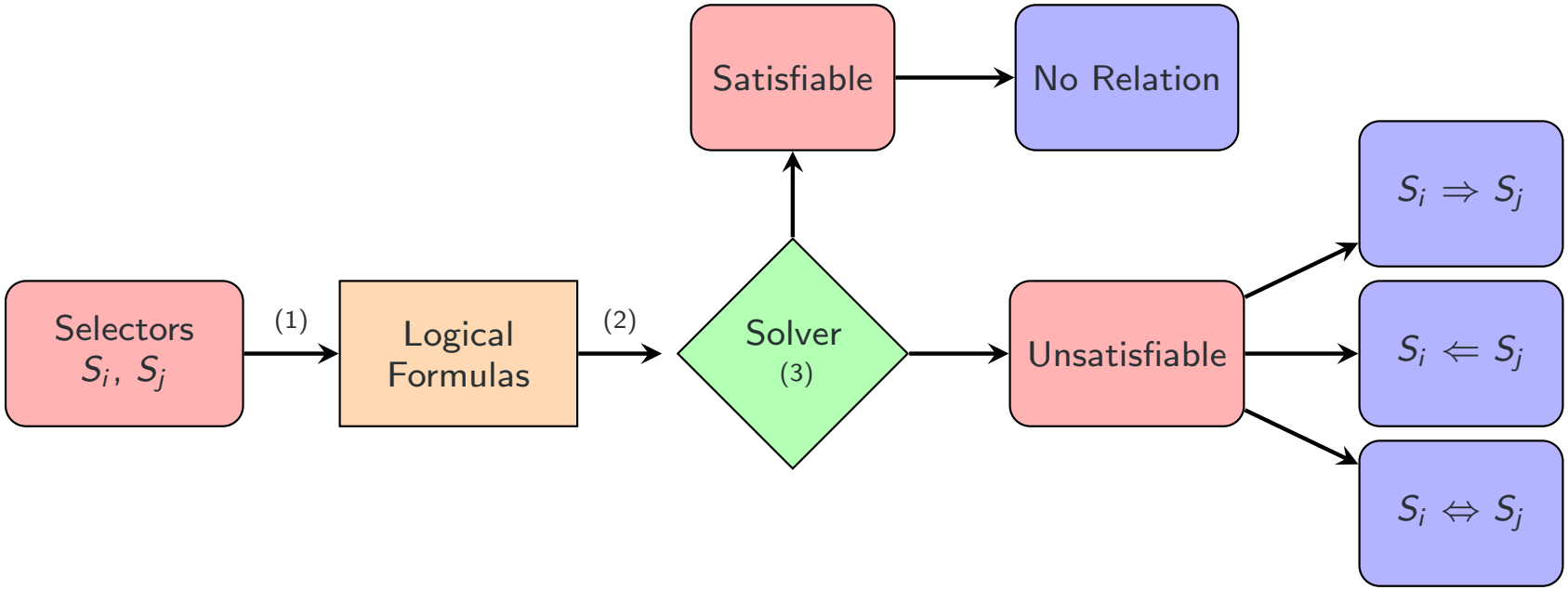
1. Identify safe CSS refactoring opportunities:

   - Rules with semantically equivalent selectors ($S_i \equiv S_j$) → merge
   - Inactive declarations
     (when $S_i \subseteq S_j$ and S_j gets preference, `p:v` under S_i is masked by `p:v'` under S_j) → delete `p:v`
   - Redundant declarations
     (when $S_i \subset S_j$, S_i gets preference, and `p:v` under both S_i and S_j, then `p:v` under S_i is unnecessary) → delete `p:v` under S_i
   - Empty rules (produced by refactoring) → remove

2. Apply corresponding rewrite rules

Common requirement: detecting relations between selectors

Detecting Relations between CSS Selectors



1. Selectors are translated into tree logic [Genevès-WWW'12]: $S_i \rightarrow \phi_i$
2. Logical implications are checked for validity ($\phi_i \implies \phi_j$ and $\phi_j \implies \phi_i$)
3. Validity is tested with satisfiability solver from [Genevès-PLDI'07]: $\text{valid}(\phi) = \text{unsat}(\neg\phi)$
4. Relations detected between selectors hold for all HTML pages (formal proof)

Contribution

A Tool that:

1. parses CSS files
2. performs a semantic analysis of selectors (calls to logical solver)
3. rearranges and cleans declarations logically
4. generates an equivalent* and smaller file for faster rendering in the browser

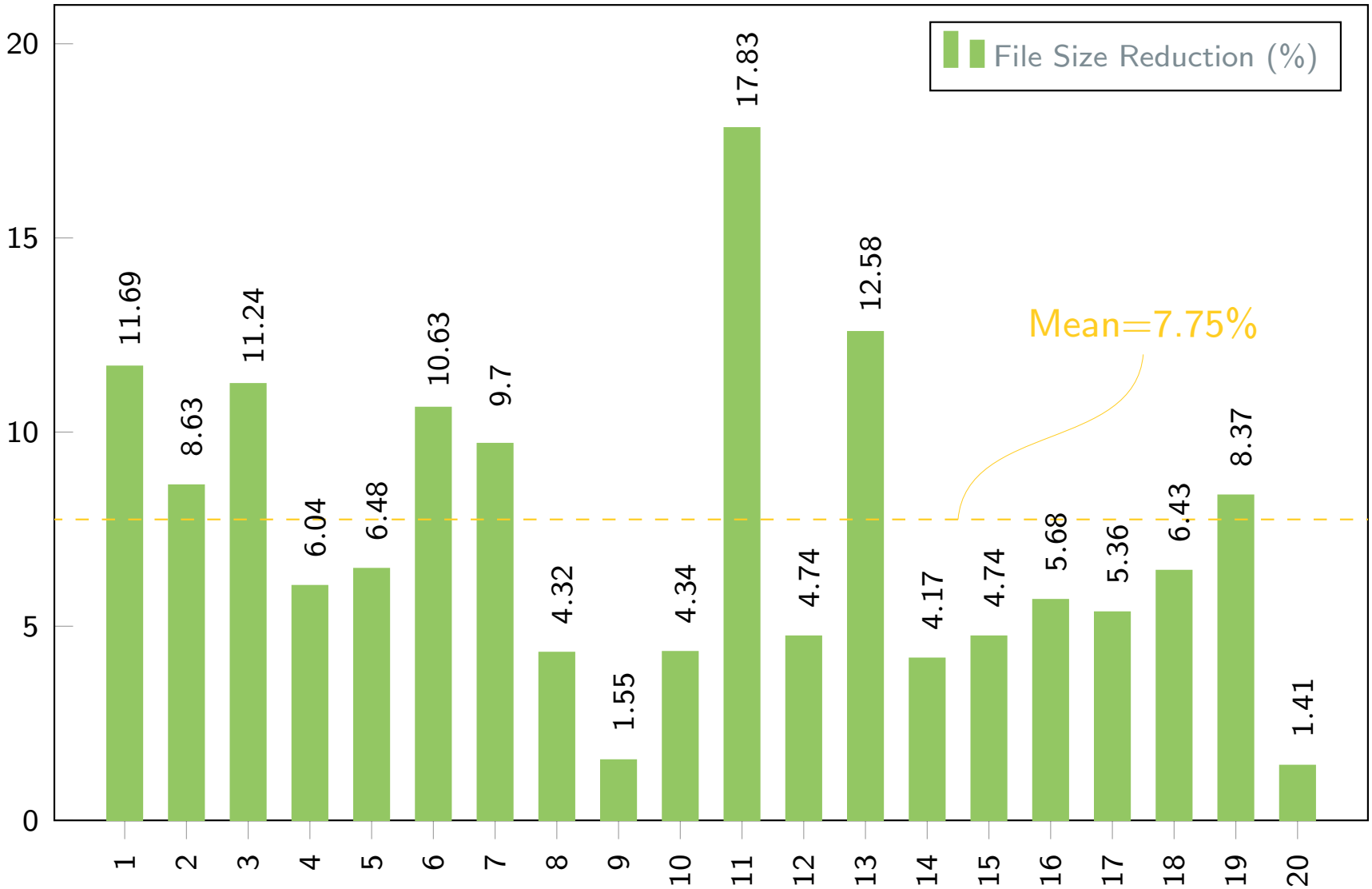
(*) "equivalent": preservation of the rendering semantics for all pages
(no page with different layout after optimization)

Real-World CSS Examples

ID	Website	# CSS Rules	ID	Website	# CSS Rules
1	ACM Digital Library	102	11	Google Sites	1676
2	Aerolineas Argentinas	1284	12	IJCAI-15	384
3	Apple	784	13	Lamborghini	1472
4	Argentina Travel	651	14	Microsoft	702
5	Clarin	1188	15	Opera	708
6	CNN	2738	16	PayPal	1089
7	Coursera	3690	17	Salesforce	1158
8	Ebay	1573	18	Shell	1111
9	Facebook	2757	19	Univ. of Cambridge	845
10	Foundation	800	20	YouTube	1841

File sizes range from 10 Kb to 320 Kb.

Experimental Results (Isolated CSS Files)

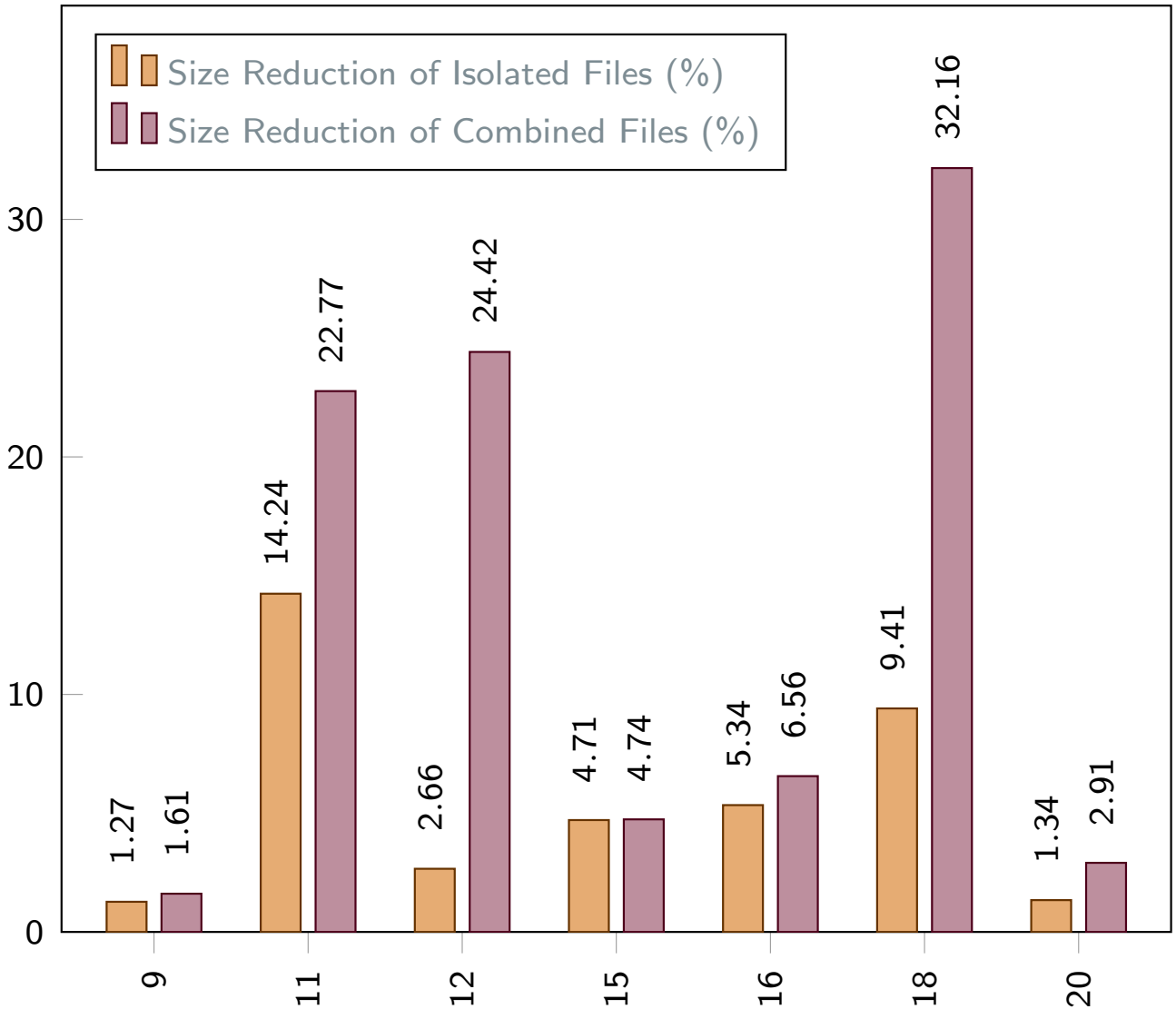


Combination of CSS Files

Real-world websites very often use several CSS files:

ID	Website	DocType	# CSS Files	Default Media	# CSS Rules
9	Facebook	HTML5	5	`all`	3543
11	Google Sites	XHTML 1	6	`screen`	2046
12	IJCAI-15	XHTML 1	14	`screen`	1170
15	Opera	HTML5	2	`all`	890
16	PayPal	HTML5	3	`all`	1186
18	Shell	HTML5	6	`all`	1551
20	YouTube	HTML5	4	`all`	3615

More Experimental Results (Combined CSS Files)



Conclusion

Transforming stylesheets automatically into equivalent but smaller versions

- An example of static analysis: "Analyze once, benefit everytime"
- Semantic analyses of CSS code made possible thanks to advances in logical solvers [Genevès-TOCL15]

Perspectives:

- Extending initial prototype by supporting more CSS features
- Considering schemas (profiles): more refactoring opportunities

`http://tyrex.inria.fr/websolver`