CSS глазами машин

Роман Дворнов
Avito

Москва 2018

- Работаю в Avito
- Open source:
  basis.js, CSSO,
  component-inspector,
  csstree, rempl и другие

Доклад о том, как машины*
понимают CSS и обрабатывают его,
какие есть сложности и проблемы

* программы для анализа и трансформации CSS

“Машины должны страдать!”

– Андрей Ситник

Но сейчас больше

страдают разработчики

Но сейчас больше страдают разработчики

«создатели»

Сложно обрабатывать CSS,

т.к. очень много нюансов

Но сейчас больше страдают разработчики

«создатели»

Сложно обрабатывать CSS,
т.к. очень много нюансов

«пользователи»

Баги и непредсказуемость
результатов

Рассказываю, потому что подгорает :)

- CSSO (мейнтейнер) – минификатор CSS
- CSSTree (автор) – набор инструментов по работе с CSS, включающий parser, walker, lexer, generator и т.д.

Вводная

CSS

#практики #методологии #трюки #тонкости #возможности #multitarget...

Сложности

- Большое количество директив, свойств, функций и т.д.
- Исключения и особые правила
- Белые пятна в спецификациях
- CSS меняется
- Разная поддержка браузерами
- Баги браузеров
- Хаки

Проблемы

- К CSS меньше внимания чем к другим веб-технологиям
- Низкий уровень знания CSS
- Препроцессоры
- FCSS (Fantasy CSS из доклада Вадима Макеева)
- Много CSS кода в проектах (мегабайты)
- Недооценивание масштаба проблем (я оптимист!)

- Директивы (at-rule) – 40
- Медиа фичи (media feature) – 69
- Псевдо-классы – 125
- Псевдо-элементы – 146
- Свойства – 1150
- Функции – 69
- Единицы измерения (unit) – 30

- Цифры включают легаси и вендорные имена
- Собрано для словарей CSSTree в рамках проекта real-web-css

Без машин не справиться!

Программы для анализа и трансформации CSS

#каксейчас #проблемы #вызовы

А вы задумывались …

- Как машины видят CSS?
- Как понимают его значение?
- Как понимают что правильно, а что нет?
- Что нужно, чтобы объяснить им как должно быть?
- ...

Немного терминологии

Rule (правило)

```
.foo, body > .bar:hover
{
    color: red;
}
```

Rule (правило)

Prelude (прелюдия)

```
.foo, body > .bar:hover
{
    color: red;
}
```

Rule (правило)

Prelude (прелюдия)

Block (блок)

```
.foo, body > .bar:hover
{
    color: red;
}
```

Rule (правило)

Selector (селектор*)

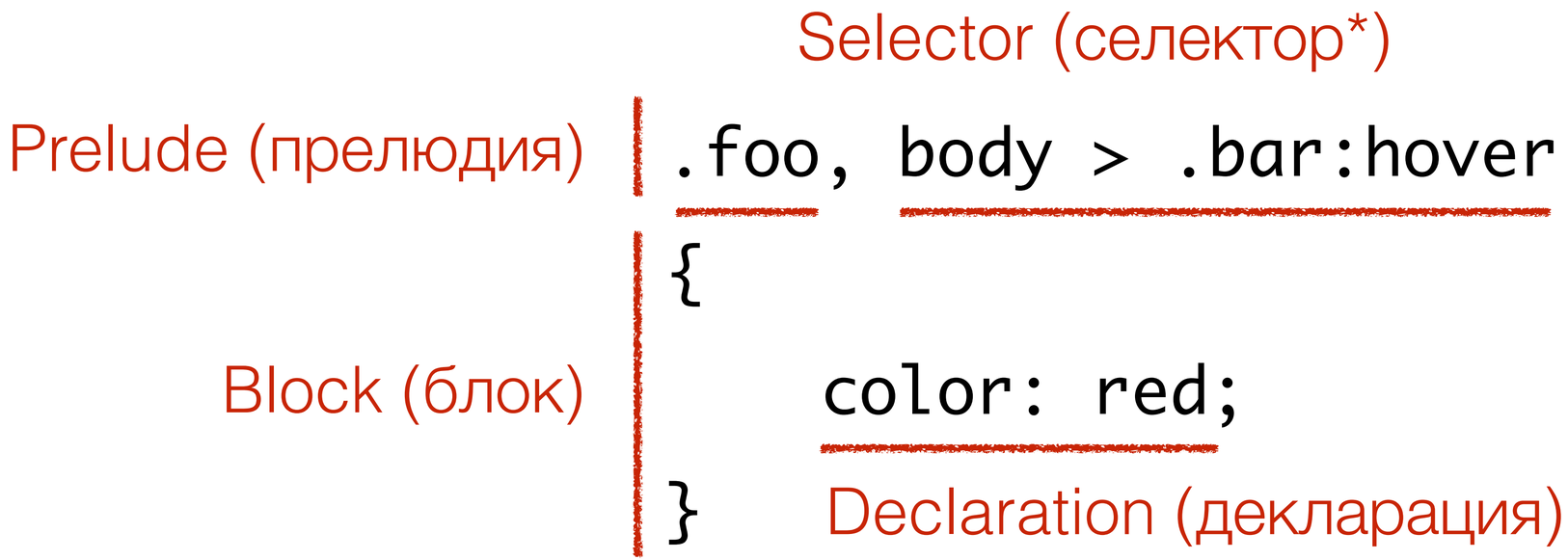
Prelude (прелюдия)

Block (блок)

```
.foo, body > .bar:hover
{
    color: red;
}
```

* в докладе термин "селектор" ссылается на complex selector

Rule (правило)



* в докладе термин "селектор" ссылается на complex selector

At-rule (директива*)

```
@media all, (max-width: 800px)
{
    ...
}
```

* перевод из словаря терминов по фронтенду

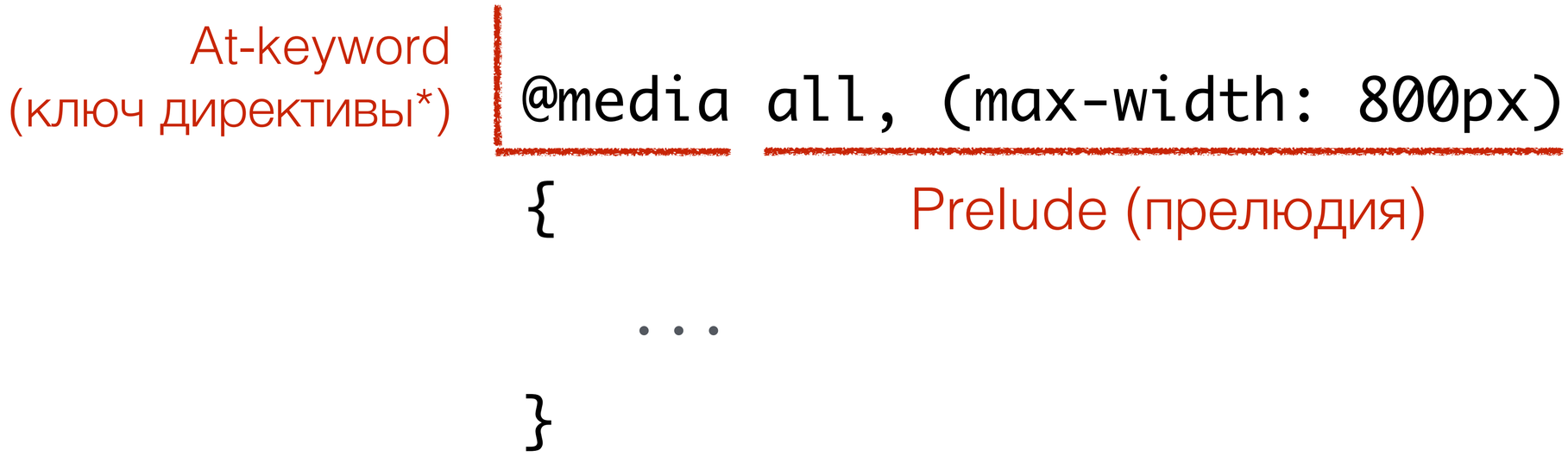
At-rule (директива*)

At-keyword
(ключ директивы*)

```
@media all, (max-width: 800px)
{
    ...
}
```

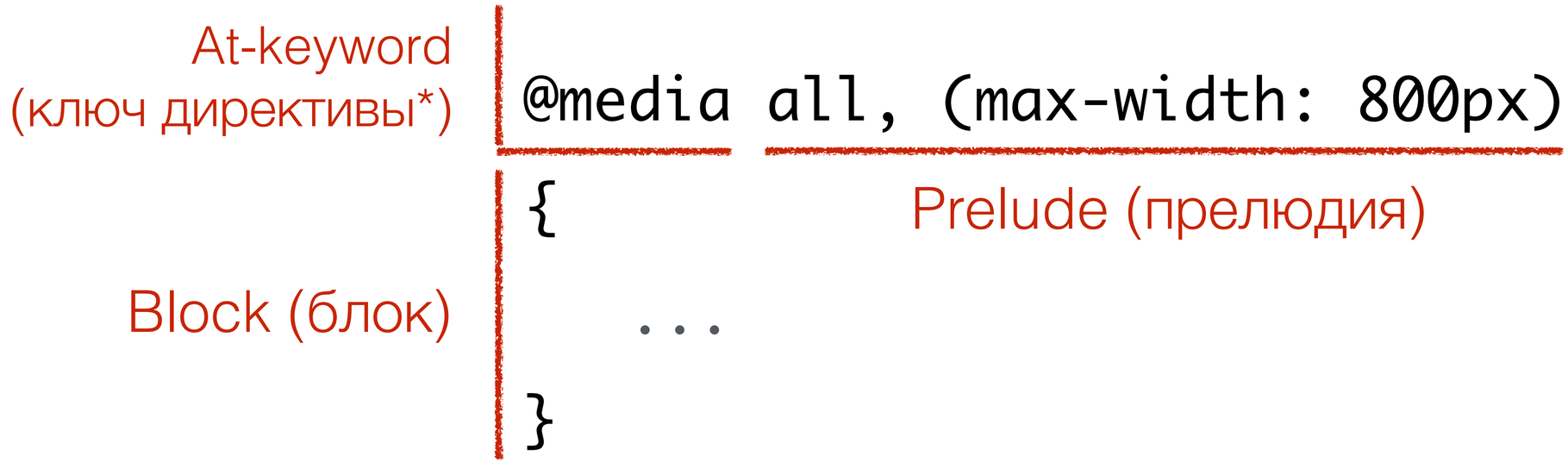
* перевод из словаря терминов по фронтенду

At-rule (директива*)



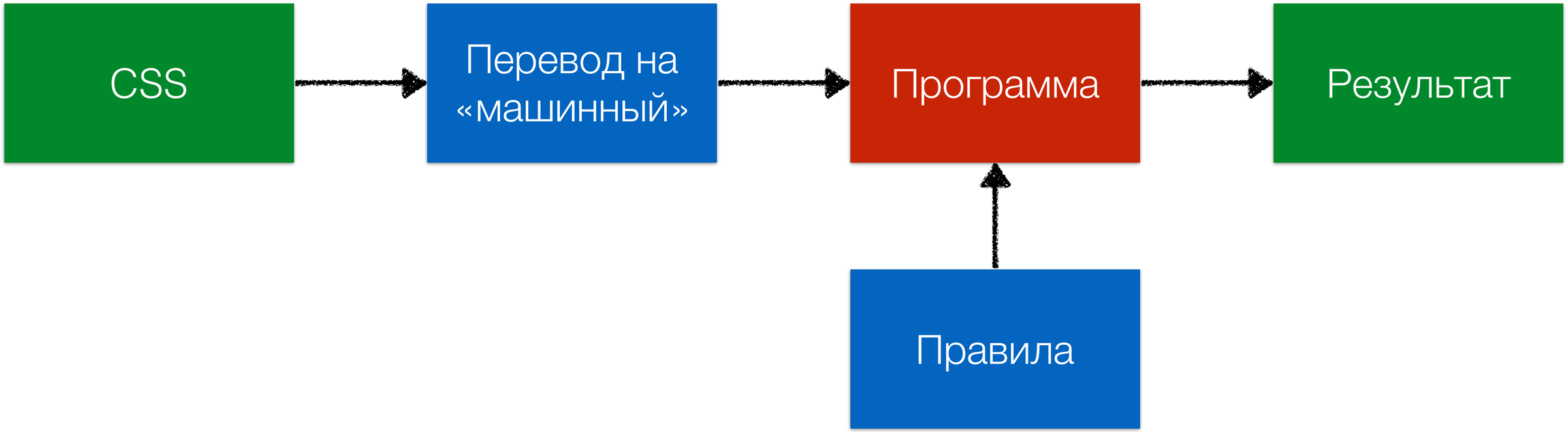
* перевод из словаря терминов по фронтенду

At-rule (директива*)



* перевод из словаря терминов по фронтенду

Как машины видят CSS


CSS
Перевод на «машинный»
Программа
Результат
Правила

AST*

структура данных

для представления исходного кода

* Abstract Syntax Tree, тот самый «машинный»

Исходный код

Строка (набор символов),
сложно определить где какая часть

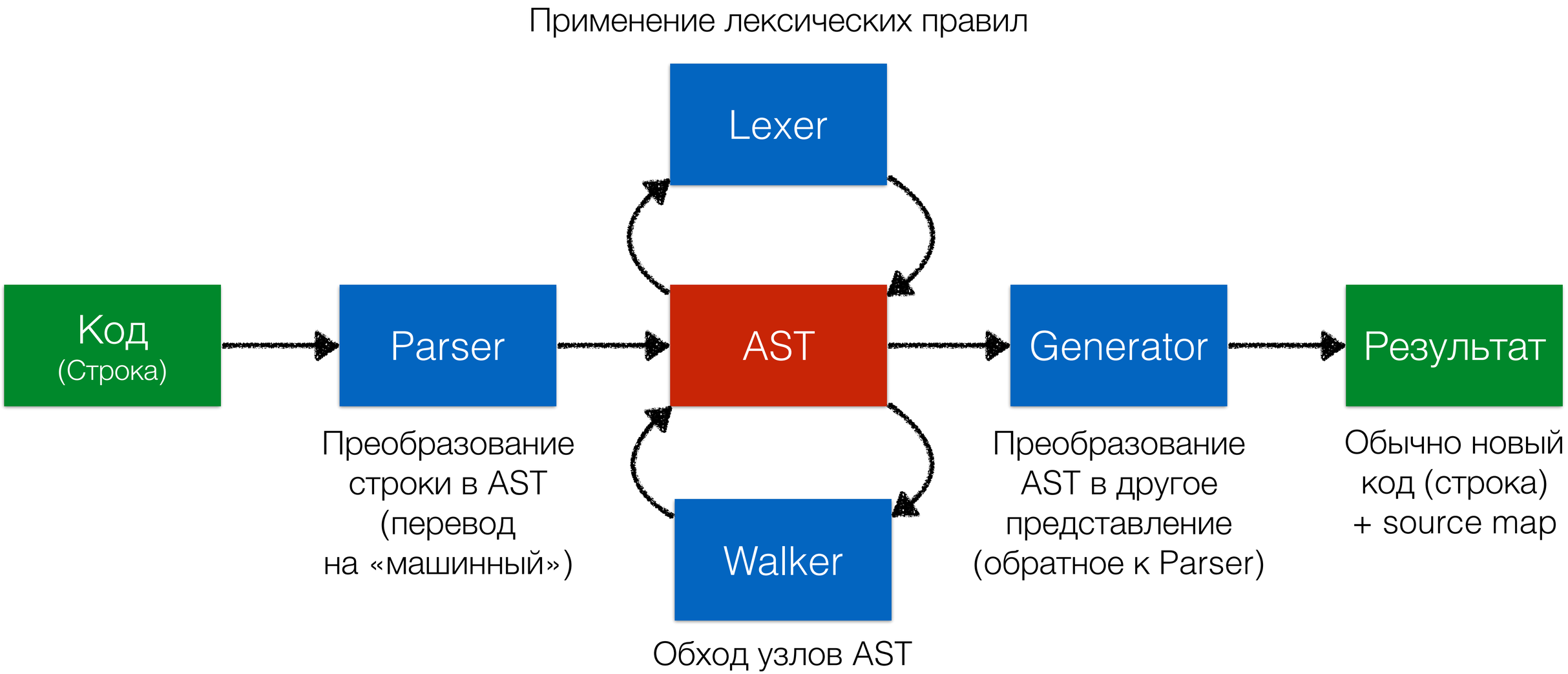
```
.foo {
  color: red;
}
```

AST

Структура (дерево), где все
разложено по полочкам

```
{
  type: "Rule",
  prelude: ".foo",
  declarations: [
    {
      type: "Declaration",
      property: "color",
      value: "red"
    }
  ]
}
```

Общая схема по работе с кодом



Так работают ...

- Линтеры (проверяют ошибки)
- Минификаторы
- Подсказки в IDE/редакторах
- Программы форматирования кода
- Пре- и пост-процессоры
- Autoprefixer и т.д.

Формат AST

Современные JavaScript парсеры
(esprima, acorn, babylon, ...)
используют спецификацию ESTree
для формата AST

Современные CSS парсеры

(PostCSS, CSSTree, Gonzales PE...)

используют каждый свой формат AST,

общей спецификации нет

Какие сложности это вызывает

- Нет совместимости между инструментами с разными парсерами
- Разработчики парсеров решают одни и те же проблемы
- Сложно с именованием типов узлов и их свойств
- Открытые вопросы как представлять разные части CSS
- Конфликт интересов в формате для разных задач

Спецификации CSS

не рассматривают формат AST,

так как они нацелены на браузеры

Что получается у меня:

Формат AST в CSSTree

По прежнему есть открытые вопросы :(

Уровень деталей

Пример

```
width: 100px
```

Пример

```
width: 100px
```

PostCSS

не разбирает значения

```
"100px"
```

Пример

```
width: 100px
```

PostCSS

не разбирает значения

```
"100px"
```

CSSTree

1 узел

```
{
  type: "Dimension",
  value: "100",
  unit: "px"
}
```

Пример

```
width: 100px
```

PostCSS

не разбирает значения

```
"100px"
```

CSSTree

1 узел

```
{
  type: "Dimension",
  value: "100",
  unit: "px"
}
```

Gonzales-pe

3 узла

```
{
  type: "dimension",
  content: [{
    type: "number",
    content: "100"
  }, {
    type: "ident",
    content: "px"
  }]
}
```

Детальность

Меньше

- Необходим дополнительный разбор
- Так как "доразбор" на стороне пользователя: больше зависимостей и шансов сломаться

Больше

- Больше типов узлов
- Больше памяти на хранение узлов
- Больше времени на обход дерева

Пример: перевод px в rem

(недетальное AST)

```
// !singlequotes|!doublequotes|!url()|pixelunit
var pxRegex = /"[^"]+"|'[^']+'|url\([^\)]+\)|(\d*\.?\d+)px/ig;

ast.walkDecls(function (decl, i) {
    ...
    decl.value = decl.value.replace(pxRegex, function (m, $1) {
        // 16px -> 1rem
    });
});
```

postcss-pxtorem (PostCSS плагин)

Та же задача с детальным AST

(как мог бы выглядеть плагин на CSSTree)

```
csstree.walk(ast, function (node) {
    ...
    if (node.type === 'Dimension' && node.unit === 'px') {
        // ... вычисление нового значения (newValue)
        node.value = newValue;
        node.unit = 'rem';
    }
});
```

Пример: подсчет specificity

(не детальное AST)

```
...
attributeRegex = /(\[[^\]]+\])/g,
idRegex = /(#[^\#\s\+>~\.\[:]+)/g,
classRegex = /(\.[^\s\+>~\.\[:]+)/g,
pseudoElementRegex = /(::[^\s\+>~\.\[:]+|:first-line|:first-letter|:before|:after)/gi,
pseudoClassWithBracketsRegex = /(:[\w-]+\([^\)]*\))/gi,
pseudoClassRegex = /(:[^\s\+>~\.\[:]+)/g,
elementRegex = /([^\s\+>~\.\[:]+)/g;
...
```

specificity (1.2M downloads/month, 32 dependants)

Пример: подсчет specificity

(детальное AST)

```
...
var A = 0, B = 0, C = 0;
simpleSelector.children.each(function walk(node) {
    switch (node.type) {
        case 'SelectorList':
        case 'Selector':
            node.children.each(walk); break;

        case 'IdSelector':
            A++; break;

        case 'ClassSelector':
        case 'AttributeSelector':
            B++; break;
...
```

модуль подсчета specificity в CSSO (57 SLOC)

Детальность AST

- Нужный уровень детальности зависит от задачи
  (например, если обрабатываются только селекторы,
  не нужно разбирать значения деклараций)
- Уровень детализации зависит от парсера
- В некоторых парсерах можно управлять уровнем детализации

Несколько примеров

- PostCSS

  Не разбирает (оставляет строкой): прелюдии у директив и правил (selector list), значения деклараций. Детализация не настраивается

- CSSTree

  Не разбирает значения custom properties, но можно включить настройкой парсера. Можно отключить разбор прелюдий и значений деклараций

- Gonzales PE

  Самой большой уровень детализации. Детализация не настраивается

Детальность в CSSTree

```
csstree.parse('@example 1 2;');
```

```
{
    "type": "Atrule",
    "prelude": {
        "type": "AtrulePrelude",
        "children": [
            { "type": "Number", "value": "1" },
            { "type": "WhiteSpace", "value": " " },
            { "type": "Number", "value": "2" }
        ]
    },
    "block": null
}
```

Детальность в CSSTree

```
csstree.parse('@example 1 2;', {
    parseAtrulePrelude: false
});
```

```
{
    "type": "Atrule",
    "prelude": {
        "type": "Raw",
        "value": "1 2"
    },
    "block": null
}
```

Паритет CSSTree с PostCSS

```
csstree.parse(css, {
    positions: true,
    parseAtrulePrelude: false,
    parseRulePrelude: false,
    parseValue: false
});
```

Разбор

CSS is hard

- at-rules @media, @supports, @font-face, …
- необязательные кавычки url(…), [name=value], …
- сложные функции calc(), var(), attr(), …
- экранирование
- bad-url, bad-string…
- …

Экранирование

```
#\31 \ \+\ 2 {
  content: "hello\
world";
  background: url(p\(4\).jpg);
}
```

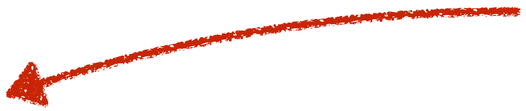
В идентификаторах можно использовать любые символы, если экранировать; в браузерах есть метод для этого – CSS.escape()

```
CSS.escape('1 + 2') === '\31 \ \+\ 2'
```

drafts.csswg.org/cssom/#the-css.escape()-method

Экранирование

```
#\31 \ \+\ 2 {
  content: "hello\
world";
  background: url(p\(4\).jpg);
}
```



Можно экранировать перевод строки в строках, как в JavaScript

www.w3.org/TR/CSS22/grammar.html#grammar

Экранирование

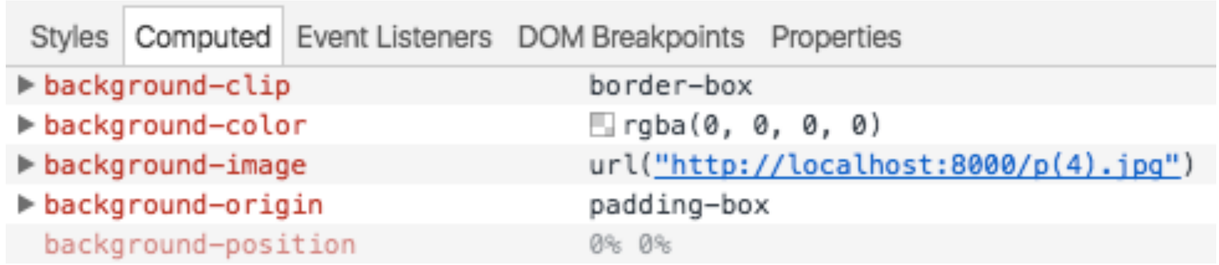
```
#\31 \+2 {
  content: "hello\
world";
  background: url(p\(4\).jpg);
}
```

Можно экранировать специальные символы в url()

www.w3.org/TR/CSS22/grammar.html#grammar

Экранирование

Браузер поймет правильно



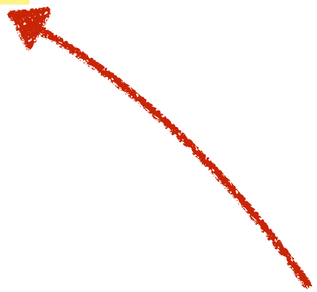
Когда парсер плох (Chrome DevTools)

@supports

```
@supports (display: flex) {
  .simple {
    display: flex;
  }
}
```

@supports

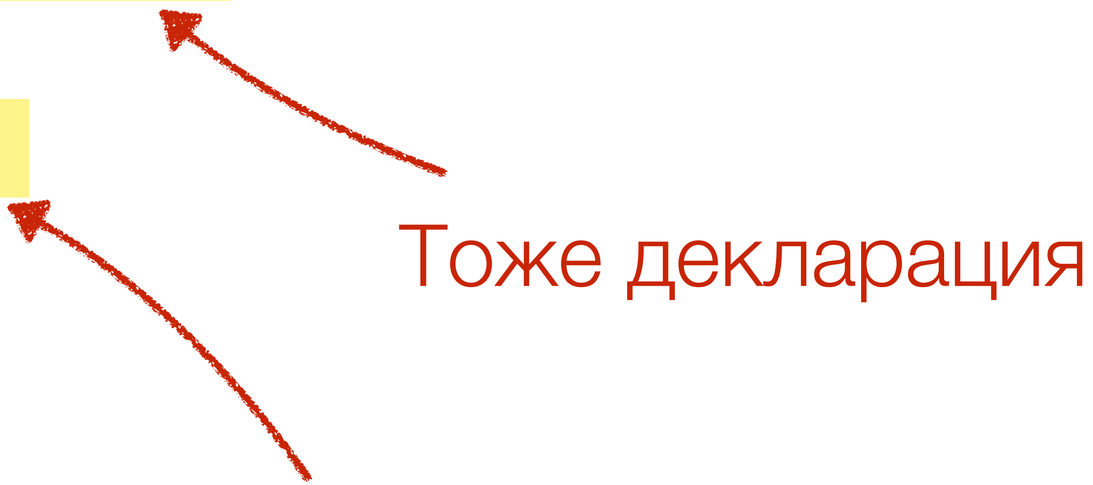
```
@supports (display: flex) {
   .simple {
      display: flex;
   }
}
```



Декларация

@supports

```
@supports (display: flex) {
   .simple {
     display: flex;
   }
}
```



@supports

```
@supports (background: top calc(25% + 2px) !important) {
   …
}
```

С точки зрения спецификаций разрешается все,
что можно в декларации, даже !important

www.w3.org/TR/css3-conditional/#at-supports

§ 2.2. Error Handling

When errors occur in CSS, the parser attempts to recover gracefully, throwing away only the minimum amount of content before returning to parsing as normal. This is because errors aren't always mistakes—new syntax looks like an error to an old parser, and it's useful to be able to add new syntax to the language ...

CSS Syntax Module Level 3

Парсеры CSS толерантные к ошибкам

- CSSTree – толерантный к ошибкам согласно спецификации
- PostCSS – толерантен к ошибкам, если используется postcss-safe-parser вместо стандартного парсера. Соответствие спецификации не известно
- Это все варианты, что я знаю...

* парсеры написанные на JavaScript

Итоги

Ключевое

- Нет стандарта формата AST для CSS
- Уровень детализации AST может быть разный
- Детальное AST может требовать больше ресурсов, но удобнее для анализа и трансформации
- Парсинг CSS в AST сложная задача, много нюансов
- CSS парсер должен быть толерантным к ошибкам

Формат AST и его свойства,
определяют что и как
машины могут делать с CSS

Обход и поиск

Обход осуществляет walker, обходит все узлы дерева и вызывает функцию обработчик

Обход

Сложности: декларация

```
@supports (display: flex) {
    .foo {
        font-family: "Open Sans";
    }
}

@font-face {
    font-family: "Open Sans";
}
```

Сложности: декларация

```
@supports (display: flex) {
    .foo {
        font-family: "Open Sans";
    }
}

@font-face {
    font-family: "Open Sans";
}
```

В CSSTree это все декларации (тип узла Declaration):

- Декларации внутри @supports обычно обходить не нужно, их приходится игнорировать (а если нужны?)
- Внутри @font-face на самом деле не декларация, а дескриптор (decscriptor); для них, например, нельзя применять !important (будет считать ошибкой и браузер проигнорирует дескриптор)

Сложности: селекторы

```
@page :first {
    ...
}

@keyframes name {
    from { ... }
    25%, 75% { ... }
    to { ... }
}

.foo {
    ...
}
```

Сложности: селекторы

```
@page :first {
    ...
}

@keyframes name {
    from { ... }
    25%, 75% { ... }
    to { ... }
}

.foo {
    ...
}
```

В CSSTree это все селекторы (тип узла SelectorList):

- Но обходить селекторы в @page обычно не нужно; да и по спеке это <page-selector-list>
- Внутри @keyframes это не обычные селекторы – <keyframe-selector>; а блок <keyframe-block>, хотя внутри все те же декларации
- Все это подталкивает к новым типам узлов, усложнению парсера и волкера

Поиск «правильного» формата AST
и правил обхода –
сложный процесс проб и ошибок

Поиск

Давайте найдем

все значения цвета?

Цвет это ...

- HexColor
- rgb(), rgba(), hls(), hlsa()
- named colors
- deprecated system colors
- vendor named colors
- ...
- CSS Color Module Level 4
    - изменения HexColor, rgb(a)/hls(a)
    - новое: hwb(), lab(), lch(), color(), color-mod(), device-cmyk()

В целом, можно победить – ведь ищем один узел... если бы не семантика

Поиск цвета

```
selector {
    font: 10px arial black;
    animation-name: red;
    whatever: green;
    width: blue;
    color: superpurple;
}
```

Сколько здесь значений цвета?

Поиск цвета

```
selector {
    font: 10px arial black;
    animation-name: red;
    whatever: green;
    width: blue;
    color: superpurple;
}
```

Часть имени шрифта "arial black"

Имя анимации

Неизвестное свойство

В width нельзя указывать цвет

Нет такого цвета

Ни одного!

Повышаем ставки – давайте найдем имена анимаций

Типичная задача для удаления не используемых @keyframes

Поиск имени анимации

```
selector {
    animation: infinite forwards normal forwards;
    animation: reverse normal forwards;
    animation: "reverse" normal forwards;
    animation: reverse rotate normal;
    animation: "reverse" normal rotate;
}
```

Что является именем анимации?

Поиск имени анимации

```
selector {
    animation: infinite forwards normal forwards;
    animation: reverse normal forwards;
    animation: "reverse" normal forwards;
    animation: reverse rotate normal;
    animation: "reverse" normal rotate;
}
```

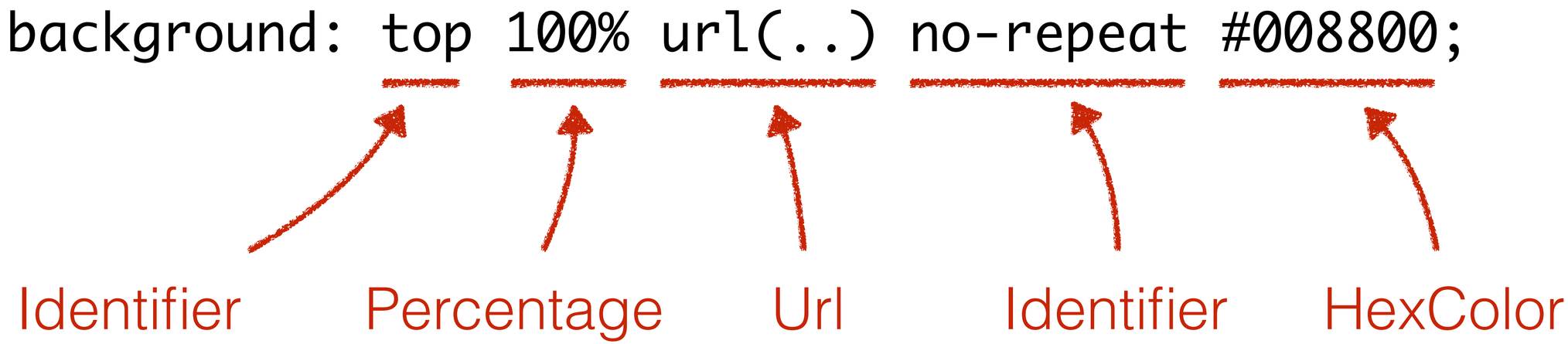
Сдавайтесь! Это нетривиальная задача

Нет времени объяснять (с)
тут могут помочь только машины, об этом дальше

Короче, чтобы решать такие задачи нужно понимать лексическое значение

Лексическое значение

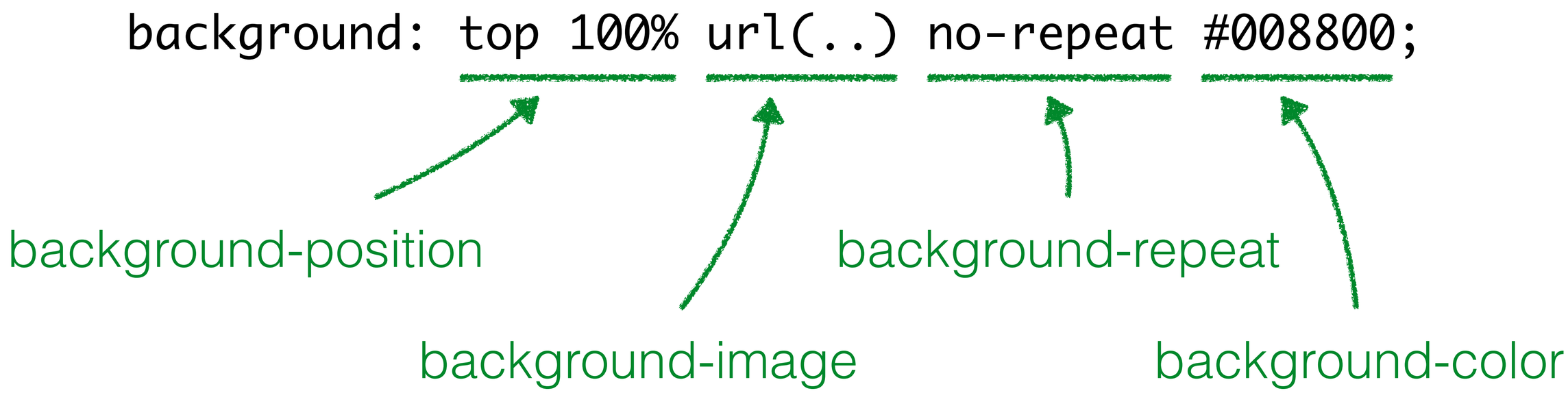
AST – абстрактные узлы



Лексическое значение

```
background: top 100% url(..) no-repeat #008800;
```

Лексическое значение



Как вы поняли что означает

та или иная часть значения?

А как должна понять машина?

В спецификации

Name:	***background***
Value:	*<bg-layer>#* , *<final-bg-layer>*
Initial:	see individual properties
Applies to:	all elements
Inherited:	no
Percentages:	see individual properties
Media:	visual
Computed value:	see individual properties
Animatable:	see individual properties

Where

<bg-layer> = *<bg-image>* || *<position>* [/ *<bg-size>*]? || *<repeat-style>* || *<attachment>* || *<box>* || *<box>*

<final-bg-layer> = *<bg-image>* || *<position>* [/ *<bg-size>*]? || *<repeat-style>* || *<attachment>* || *<box>* || *<box>* || *<'background-color'>*

В спецификации

Name:	***background***
Value:	*<bg-layer>*# , *<final-bg-layer>*
Initial:	see individual properties
Applies to:	all elements
Inherited:	no
Percentages:	see individual properties
Media:	visual
Computed value:	see individual properties
Animatable:	see individual properties

Where

<bg-layer> = *<bg-image>* || *<position>* [/ *<bg-size>*]? || *<repeat-style>* || *<attachment>* || *<box>* || *<box>*

<final-bg-layer> = *<bg-image>* || *<position>* [/ *<bg-size>*]? || *<repeat-style>* || *<attachment>* || *<box>* || *<box>* || *<'background-color'>*

Грамматика

CSS Values and Units Module

похоже на RegExp, но немного проще

drafts.csswg.org/css-values/#value-defs

Шаг первый

Идея, которая поменяла мир (немного)

Идея!

- Собрать синтаксисы в словарь
- Разобрать
- Научиться мапить на AST
- ...
- PROFIT!!!

В 2016м нашелся готовый словарь

Mozilla Template:CSSData

Таблицы описания свойств из спецификаций
в формате JSON;
использовалось для генерации страниц
developer.mozilla.org

Реализовал разбор грамматики, загрузил словари, построил граф зависимостей и тут...

O – открытие

CSSData был составлен людьми,

и использовался для генерации страниц для людей

(developer.mozilla.org)

Хьюстон, у нас проблемы!

- Ошибки в синтаксисе (нельзя разобрать)
- Некоторые синтаксисы ссылались на не существующие определения
- ...
- Люди не замечали это годами...

Я погуглил, пофиксил,

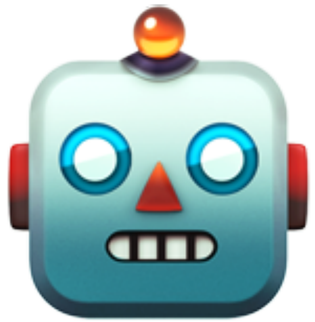
часть законтрибьютил в CSSData

(через bugtracker, sic!)

Забегая вперед

- Mozilla Template:CSSData – был служебным шаблоном в недрах MDN
- 20 сентября 2016 завязалась дискуссия относительно словарей
- Призвали ребят из MDN
- Пропушили их вынести CSSData на github
- 19 октября 2016 появился репозиторий mdn/data
- Запушил большую часть фиксов из CSSTree, добавил JSON Schema
- 22 июня 2017 ребята из MDN опубликовали пакет mdn-data 1.0 на npm
- 🎉

Словарь mdn/data (CSSData) стал пригодным для машин



О – открытие #2

До CSSTree никто не парсил синтаксисы из спек, авторы спецификаций (пока) не автоматизируют проверку

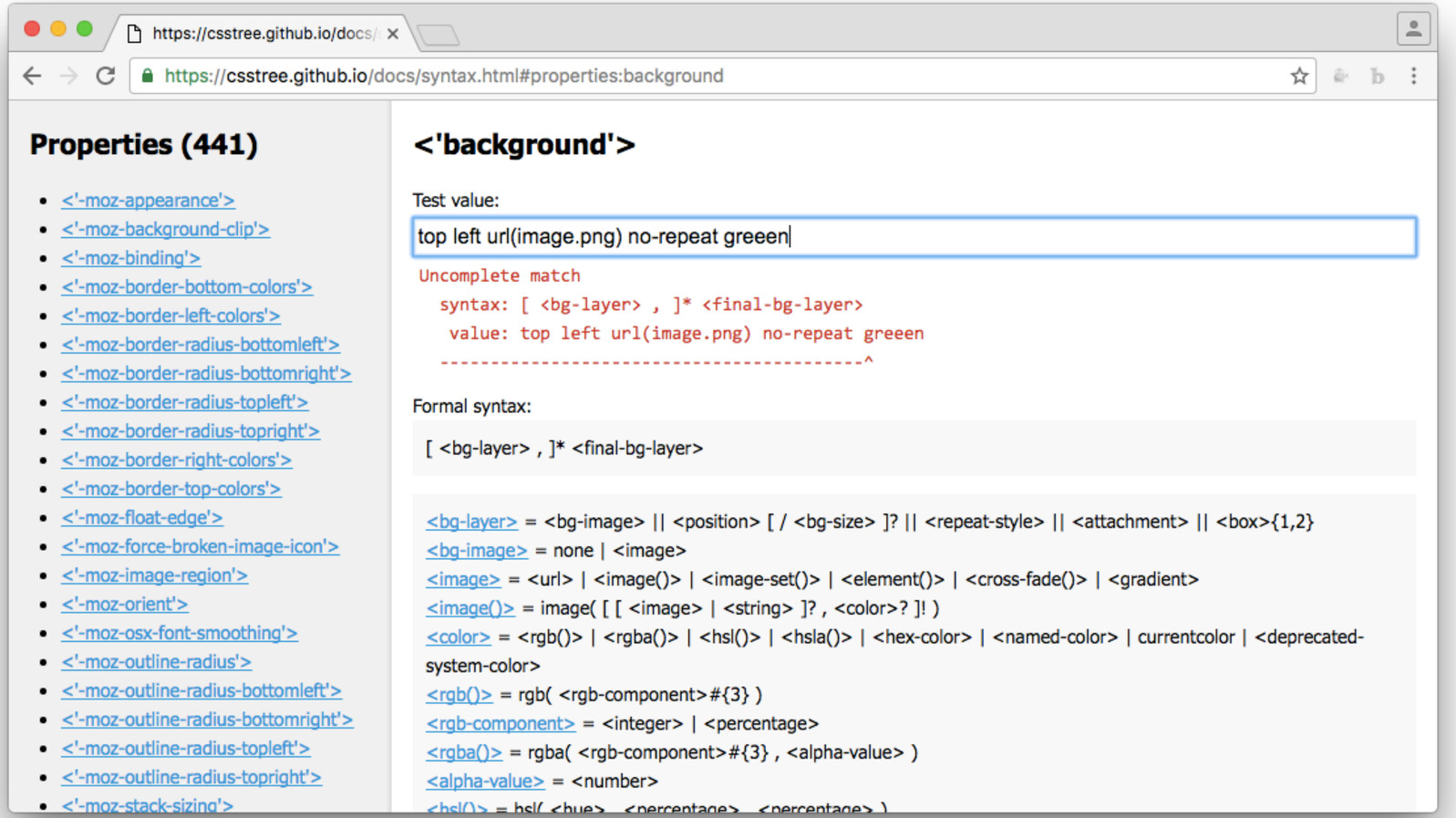
Шаг второй

Валидация и побочные продукты

Сразу не получилось полноценно матчить
AST узлы на синтаксисы,
но получилось валидировать

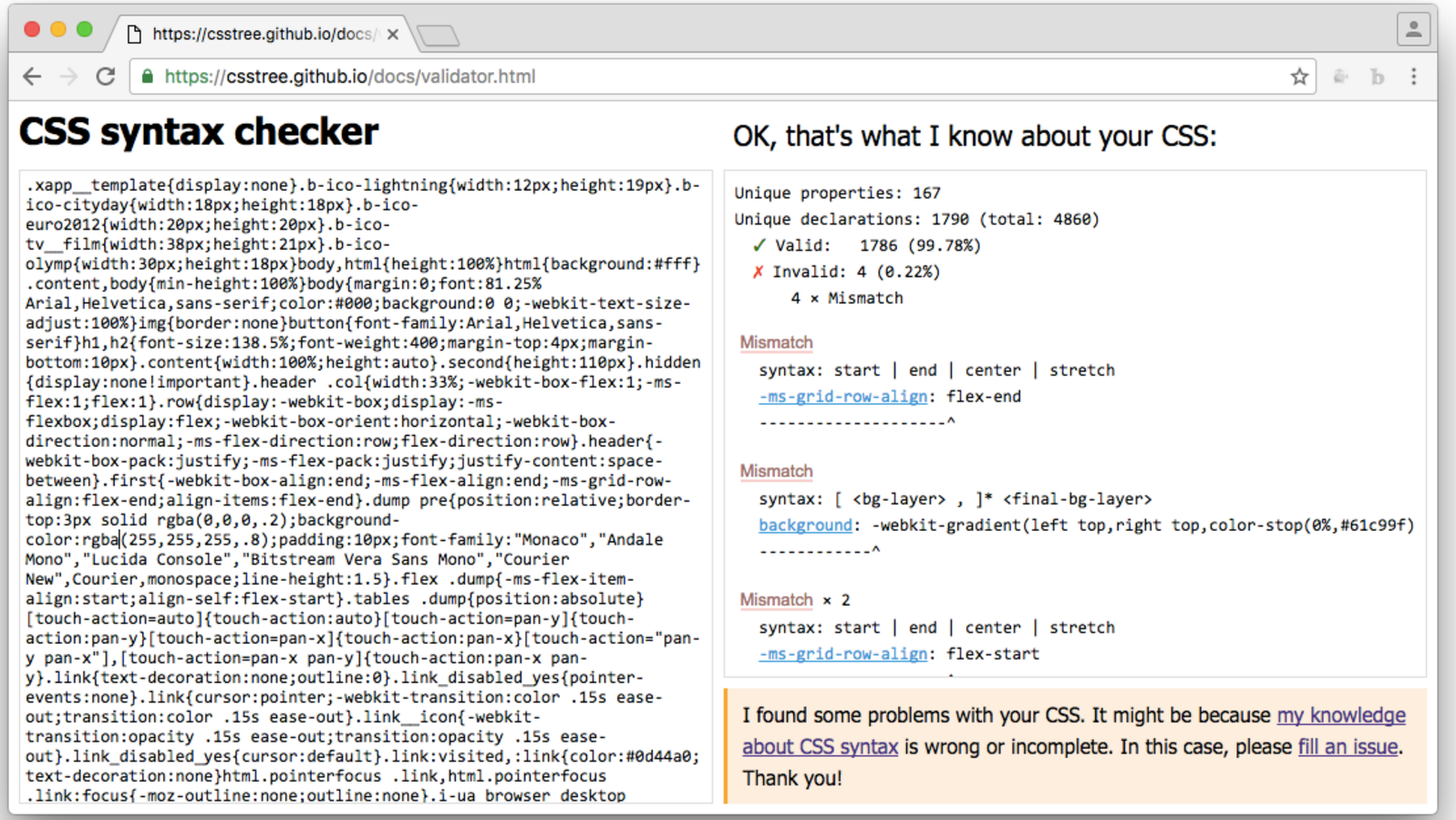
Инструменты «хелперы» на CSSTree для нужд CSSTree

Документация CSS синтаксиса



csstree.github.io/docs/syntax.html

Валидатор CSS (пока только значения)



csstree.github.io/docs/validator.html

Прикладное применение

Плагины/пакеты

- csstree-validator – npm пакет + консольная команда
- stylelint-csstree-validator – плагин для stylelint
- gulp-csstree – плагин для gulp
- SublimeLinter-contrib-csstree – плагин для Sublime Text
- vscode-csstree – плагин для VS Code
- atom-plugin – плагин для Atom

Огромное спасибо Сергею Мелюкову за его вклад и поддержку!

Шаг третий

Кто ищет – тот найдет

Доработка матчинга –

новые возможности

Вспомним пример ...

<'background'>

Test a value:

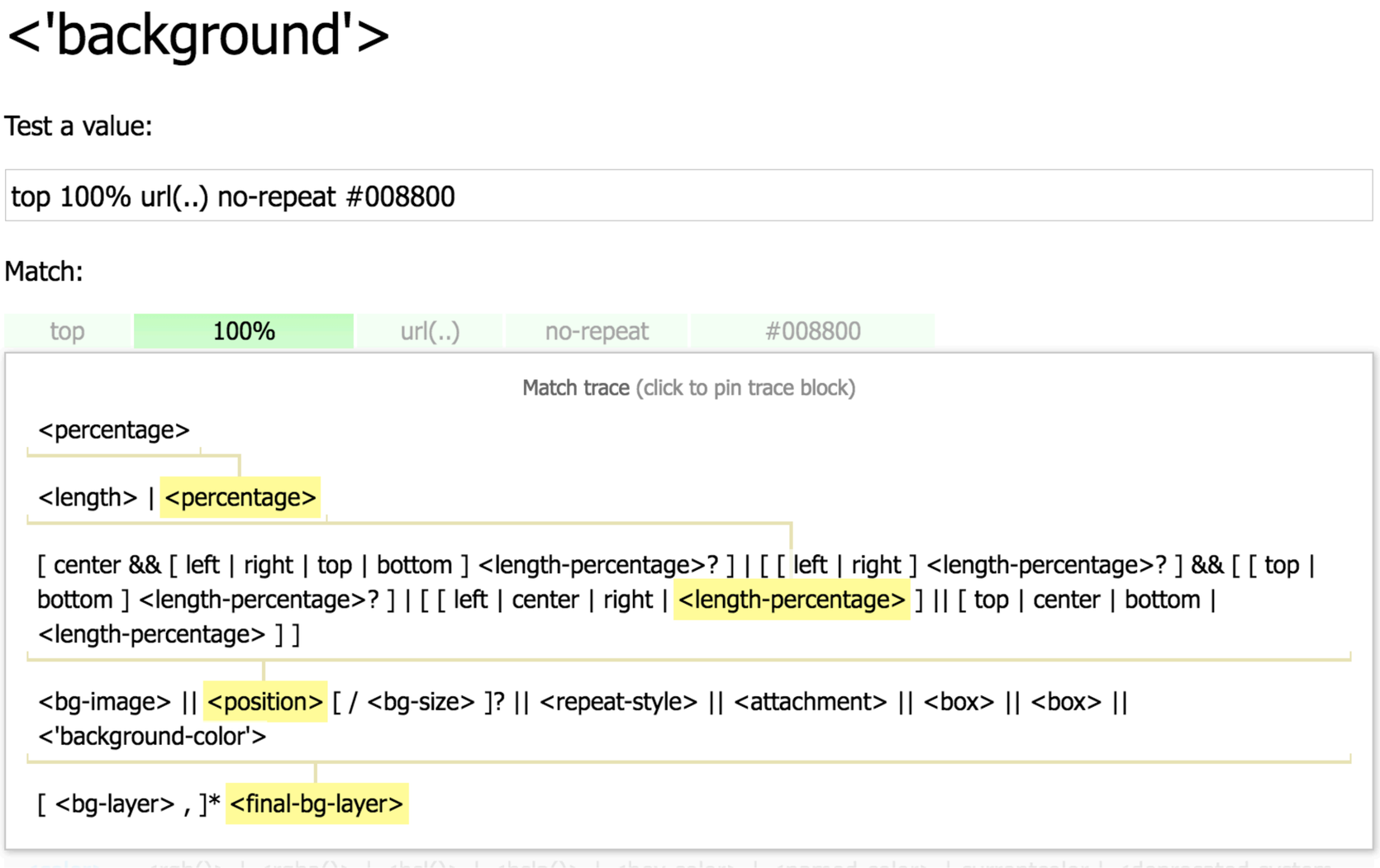
```
top 100% url(..) no-repeat #008800
```

Match:

top	100%	url(..)	no-repeat	#008800
Keyword:top	Type:percentage	Type:url	Keyword:no-repeat	Type:hex-color
	Type:length-percentage	Type:image	Type:repeat-style	Type:color
Type:position		Type:bg-image		Property:background-color
Type:final-bg-layer				
Property:background				

Теперь CSSTree так видит значение 🤘

Можно трассировать каждый узел



Узнать у CSSTree лексический тип

```
csstree.lexer
    .matchDeclaration(this.declaration)
    .isType(node, 'color')
```

Используется в CSSO при минификации значений цвета

Синтаксис цвета (без L4)

<color>

Formal syntax:

<rgb()> | <rgba()> | <hsl()> | <hsla()> | <hex-color> | <named-color> | currentcolor | <deprecated-system-color>

<rgb()> = rgb([[<percentage>{3} | <number>{3}] [/ <alpha-value>]?] | [[<percentage>#{3} | <number>#{3}] , <alpha-value>?])
<alpha-value> = <number> | <percentage>
<rgba()> = rgba([[<percentage>{3} | <number>{3}] [/ <alpha-value>]?] | [[<percentage>#{3} | <number>#{3}] , <alpha-value>?])
<hsl()> = hsl([<hue> <percentage> <percentage> [/ <alpha-value>]?] | [<hue> , <percentage> , <percentage> , <alpha-value>?])
<hue> = <number> | <angle>
<hsla()> = hsla([<hue> <percentage> <percentage> [/ <alpha-value>]?] | [<hue> , <percentage> , <percentage> , <alpha-value>?])
<named-color> = transparent | aliceblue | antiquewhite | aqua | aquamarine | azure | beige | bisque | black | blanchedalmond | blue | blueviolet | brown | burlywood | cadetblue | chartreuse | chocolate | coral | cornflowerblue | cornsilk | crimson | cyan | darkblue | darkcyan | darkgoldenrod | darkgray | darkgreen | darkgrey | darkkhaki | darkmagenta | darkolivegreen | darkorange | darkorchid | darkred | darksalmon | darkseagreen | darkslateblue | darkslategray | darkslategrey | darkturquoise | darkviolet | deeppink | deepskyblue | dimgray | dimgrey | dodgerblue | firebrick | floralwhite | forestgreen | fuchsia | gainsboro | ghostwhite | gold | goldenrod | gray | green | greenyellow | grey | honeydew | hotpink | indianred | indigo | ivory | khaki | lavender | lavenderblush | lawngreen | lemonchiffon | lightblue | lightcoral | lightcyan | lightgoldenrodyellow | lightgray | lightgreen | lightgrey | lightpink | lightsalmon | lightseagreen | lightskyblue | lightslategray | lightslategrey | lightsteelblue | lightyellow | lime | limegreen | linen | magenta | maroon | mediumaquamarine | mediumblue | mediumorchid | mediumpurple | mediumseagreen | mediumslateblue | mediumspringgreen | mediumturquoise | mediumvioletred | midnightblue | mintcream | mistyrose | moccasin | navajowhite | navy | oldlace | olive | olivedrab | orange | orangered | orchid | palegoldenrod | palegreen | paleturquoise | palevioletred | papayawhip | peachpuff | peru | pink | plum | powderblue | purple | rebeccapurple | red | rosybrown | royalblue | saddlebrown | salmon | sandybrown | seagreen | seashell | sienna | silver | skyblue | slateblue | slategray | slategrey | snow | springgreen | steelblue | tan | teal | thistle | tomato | turquoise | violet | wheat | white | whitesmoke | yellow | yellowgreen | <-non-standart-color>
<-non-standart-color> = -moz-ButtonDefault | -moz-ButtonHoverFace | -moz-ButtonHoverText | -moz-CellHighlight | -moz-CellHighlightText | -moz-Combobox | -moz-ComboboxText | -moz-Dialog | -moz-DialogText | -moz-dragtargetzone | -moz-EvenTreeRow | -moz-Field | -moz-FieldText | -moz-html-CellHighlight | -moz-html-CellHighlightText | -moz-mac-accentdarkestshadow | -moz-mac-accentdarkshadow | -moz-mac-accentface | -moz-mac-accentlightesthighlight | -moz-mac-accentlightshadow | -moz-mac-accentregularhighlight | -moz-mac-accentregularshadow | -moz-mac-chrome-active | -moz-mac-chrome-inactive | -moz-mac-focusring | -moz-mac-menuselect | -moz-mac-menushadow | -moz-mac-menutextselect | -moz-MenuHover | -moz-MenuHoverText | -moz-MenuBarText | -moz-MenuBarHoverText | -moz-nativehyperlinktext | -moz-OddTreeRow | -moz-win-communicationstext | -moz-win-mediatext | -moz-activehyperlinktext | -moz-default-background-color | -moz-default-color | -moz-hyperlinktext | -moz-visitedhyperlinktext | -webkit-activelink | -webkit-focus-ring-color | -webkit-link | -webkit-text
<deprecated-system-color> = ActiveBorder | ActiveCaption | AppWorkspace | Background | ButtonFace | ButtonHighlight | ButtonShadow | ButtonText | CaptionText | GrayText | Highlight | HighlightText | InactiveBorder | InactiveCaption | InactiveCaptionText | InfoBackground | InfoText | Menu | MenuText | Scrollbar | ThreeDDarkShadow | ThreeDFace | ThreeDHighlight | ThreeDLightShadow | ThreeDShadow | Window | WindowFrame | WindowText

Синтаксис свойства animation

<'animation'>

Formal syntax:

```
<single-animation>#
```

```
<single-animation> = <time> || <single-timing-function> || <time> || <single-animation-iteration-count> || <single-animation-direction> || <single-animation-fill-mode> || <single-animation-play-state> || [ none | <keyframes-name> ]
<single-timing-function> = linear | <cubic-bezier-timing-function> | <step-timing-function> | <frames-timing-function>
<cubic-bezier-timing-function> = ease | ease-in | ease-out | ease-in-out | cubic-bezier( <number> , <number> , <number> , <number> )
<step-timing-function> = step-start | step-end | steps( <integer> [, [ start | end ] ]? )
<frames-timing-function> = frames( <integer> )
<single-animation-iteration-count> = infinite | <number>
<single-animation-direction> = normal | reverse | alternate | alternate-reverse
<single-animation-fill-mode> = none | forwards | backwards | both
<single-animation-play-state> = running | paused
<keyframes-name> = <custom-ident> | <string>
```

Поиск с CSSTree

```
// найти все цвета
csstree.lexer.matchAllFragments(ast, 'Type', 'color')

// ... имена анимаций
csstree.lexer.matchAllFragments(ast, 'Type', 'keyframes-name')

// ... имена шрифтов
csstree.lexer.matchAllFragments(ast, 'Type', 'family-name')
```

One more thing(s)

Чтобы добавить новые свойства и синтаксисы – не нужно менять код, нужно обновить словарь

Возможность экспериментировать с новыми синтаксисами, создавать свои синтаксисы

Почему стоит начать изучать грамматику

CSS Values and Units Module

уже сегодня

CSS Properties and Values API Level 1

Позволит задавать свой синтаксис

для ваших Custom properties

Итоги

Ключевое

- Появился инструмент для работы с грамматикой CSS Values and Units Module, которая может стать прикладной
- Приведен в порядок словарь mdn/data (теперь для машин)
- Можно искать фрагменты значений по лексическому типу
- Несколько действительно полезных инструментов – информационных и прикладных (пощупайте!)
- Новые возможности для ваших идей

Словари

Дальнейшее совершенствование инструментов не возможно без словарей для машин

Словари

- mdn/data – описание свойств (таблицы из спек), директив, селекторов и т.д.
- mdn/browser-compat-data – поддержка разных возможностей (JS/CSS/API) браузерами
- known-css-properties – список известных свойств
- caniuse – поддержка браузерами некоторых частей CSS

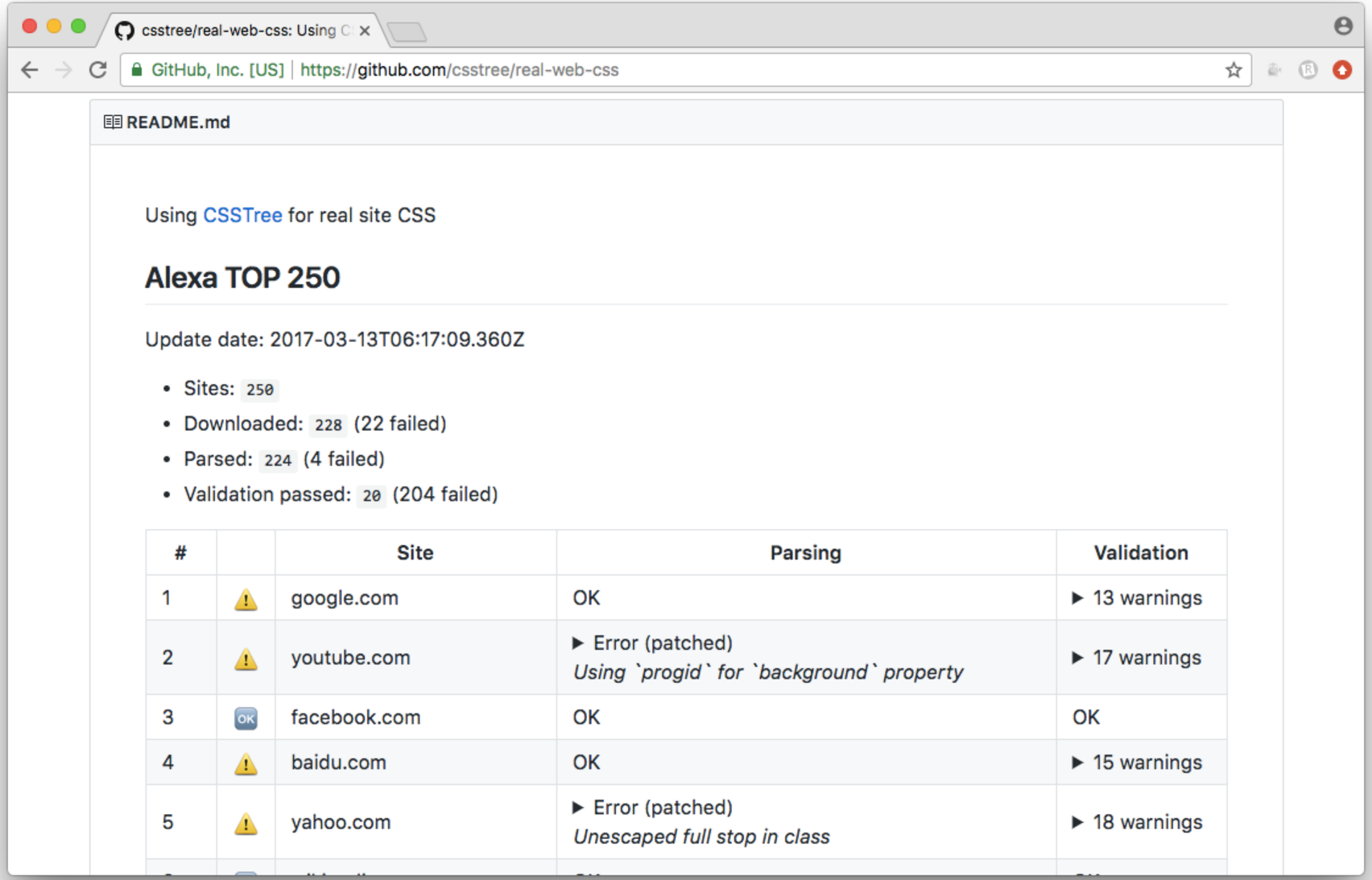
CSS свойств

- mdn/data – **367**
- Alexa Top 250 – **483**
- Известных свойств – **1150**

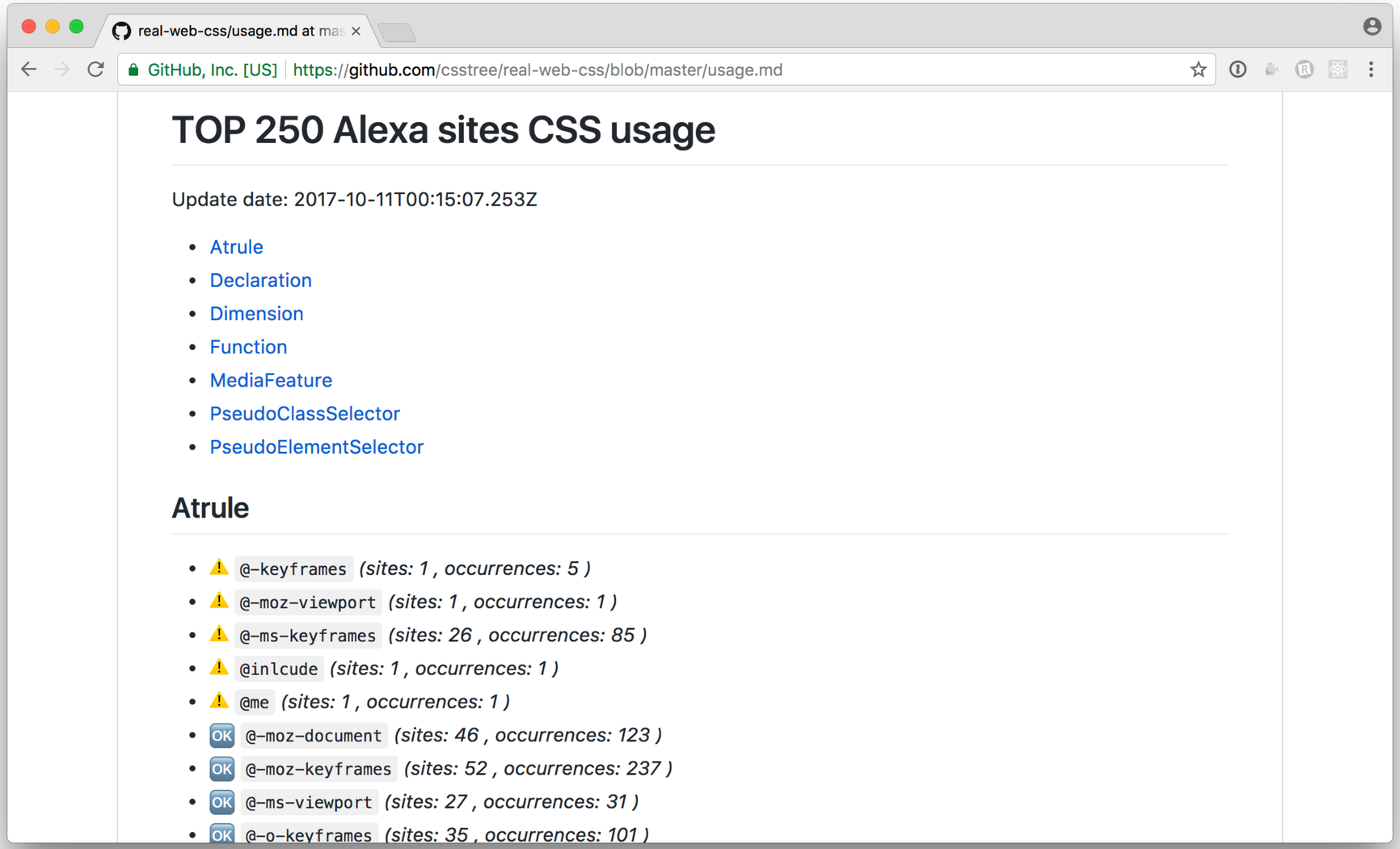
Что можно сделать:

- найти неописанное свойство
- нагуглить описание
- сделать PR с описанием в JSON

real-web-css

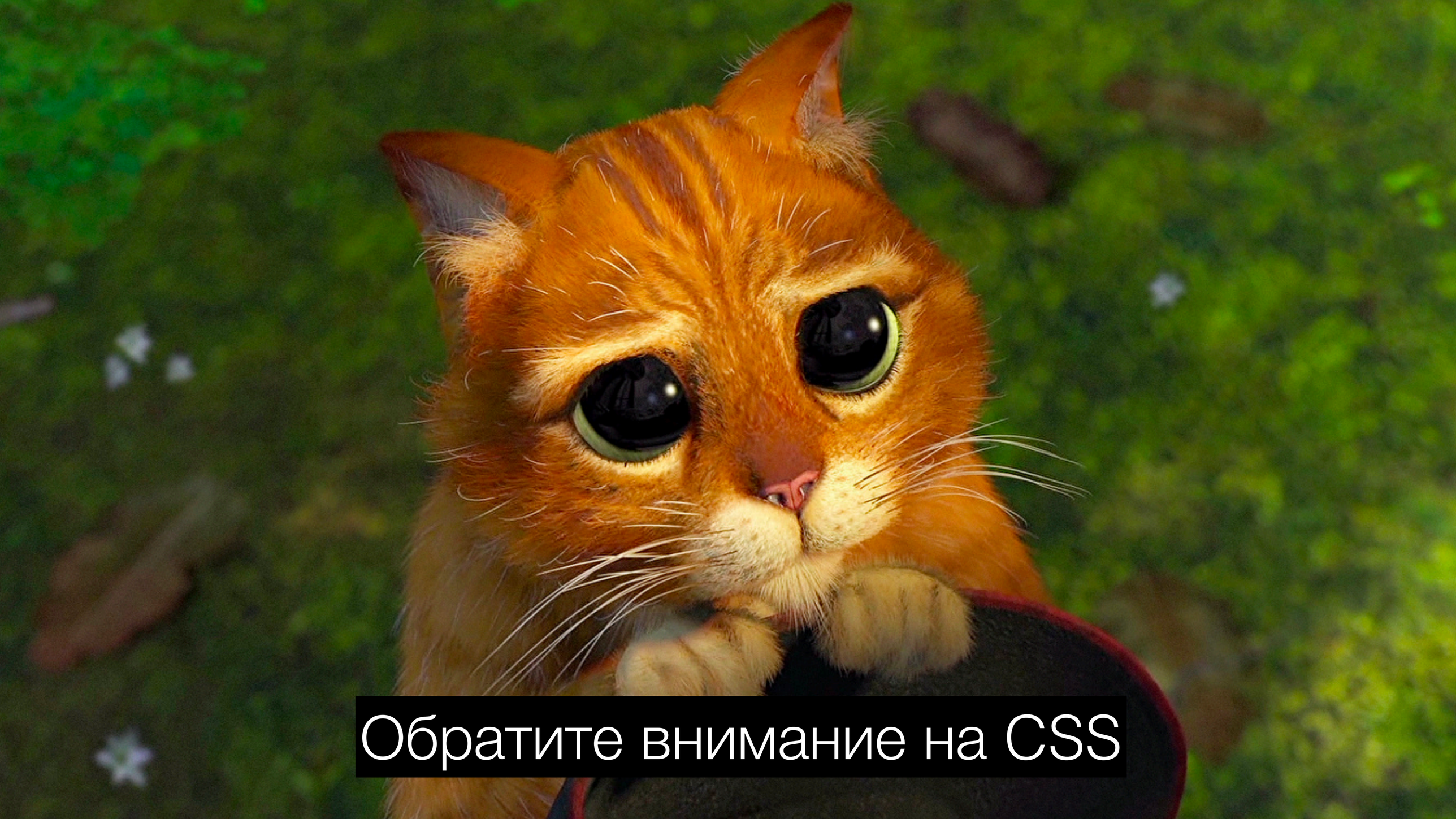

csstree/real-web-css: Using C
GitHub, Inc. [US] | https://github.com/csstree/real-web-css
README.md
Using CSSTree for real site CSS
Alexa TOP 250
Update date: 2017-03-13T06:17:09.360Z
Sites: 250
Downloaded: 228 (22 failed)
Parsed: 224 (4 failed)
Validation passed: 20 (204 failed)
| Site | Parsing | Validation
1 | google.com | OK | ▶ 13 warnings
2 | youtube.com | ▶ Error (patched) Using `progid` for `background` property | ▶ 17 warnings
3 | facebook.com | OK | OK
4 | baidu.com | OK | ▶ 15 warnings
5 | yahoo.com | ▶ Error (patched) Unescaped full stop in class | ▶ 18 warnings

real-web-css


real-web-css/usage.md at mas
GitHub, Inc. [US] https://github.com/csstree/real-web-css/blob/master/usage.md
TOP 250 Alexa sites CSS usage
Update date: 2017-10-11T00:15:07.253Z
Atrule
Declaration
Dimension
Function
MediaFeature
PseudoClassSelector
PseudoElementSelector
Atrule
⚠ @-keyframes (sites: 1 , occurrences: 5)
⚠ @-moz-viewport (sites: 1 , occurrences: 1)
⚠ @-ms-keyframes (sites: 26 , occurrences: 85)
⚠ @inlcude (sites: 1 , occurrences: 1)
⚠ @me (sites: 1 , occurrences: 1)
OK @-moz-document (sites: 46 , occurrences: 123)
OK @-moz-keyframes (sites: 52 , occurrences: 237)
OK @-ms-viewport (sites: 27 , occurrences: 31)
OK @-o-keyframes (sites: 35 , occurrences: 101)

Заключение

- Всё сложно
- Много нерешенный вопросов в обработке CSS
- Инструменты совершенствуются и переходят на новый уровень
- Новые возможности
- Словари наше всё
- Вы можете помочь!


Обратите внимание на CSS

Спасибо!

Роман Дворнов

@rdvornov

rdvornov@gmail.com