

A Unified Interaction Model For Web Scraping & Customization

Kapaya Katongo
MIT CSAIL
Cambridge, MA, USA
kkatongo@mit.edu

Kathryn Jin
MIT CSAIL
Cambridge, MA, USA
kjin@mit.edu

Geoffrey Litt
MIT CSAIL
Cambridge, MA, USA
glitt@mit.edu

Daniel Jackson
MIT CSAIL
Cambridge, MA, USA
dnj@csail.mit.edu

ABSTRACT

Web scraping, the process of extracting structured data from a website, is a common building block of web customization systems. Prior approaches have allowed users to perform web scraping by directly demonstrating examples of the data to scrape. However, demonstrations alone do not allow for as much expressiveness as traditional programming which is vital for complex web scraping tasks.

In this paper, we present a new interaction model for web scraping that combines the ease of use of programming-by-demonstration and the expressiveness of traditional programming. When a user demonstrates examples of data to extract, a web scraping program is synthesized and presented in a formula language with pure functional semantics. Crucially, the user can directly edit and execute formulas, allowing them to express scraping operations which can not be achieved via demonstration alone.

To illustrate our model, we implement it as a browser extension called Joker. Through concrete examples, a small user study and a Cognitive Dimensions of Notation analysis, we show how Joker offers an expressive interaction model for web scraping and customization.

CCS CONCEPTS

• **Software and its engineering** → **Integrated and visual development environments.**

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference'17, July 2017, Washington, DC, USA

© 2021 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

<https://doi.org/10.1145/nnnnnnnn.nnnnnnn>

KEYWORDS

end-user programming, software customization, web scraping, programming-by-example, program synthesis

ACM Reference Format:

Kapaya Katongo, Geoffrey Litt, Kathryn Jin, and Daniel Jackson. 2021. A Unified Interaction Model For Web Scraping & Customization. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/nnnnnnnn.nnnnnnn>

1 INTRODUCTION

Many websites on the internet do not meet the exact needs of all of their users. Because of this, millions of people use browser extensions like Greasemonkey [4] and Tampermonkey [5] to install userscripts, snippets of Javascript code which customize the behavior of websites. To make the creation of web customizations more accessible to end-users without knowledge of programming, researchers have developed systems like Sifter [16], Vegemite [22] and Wildcard [23, 24].

A common building block of these web customization systems is web scraping, the extraction of structured data from websites. They achieve web scraping in one of two ways: programming-by-demonstration [16, 22] and traditional programming [23, 24]. Web scraping by demonstration automates the web scraping process by utilizing program synthesis to synthesize a web scraping program from user demonstrations. This approach is accessible to end-users with no programming experience and enables them to fully participate in the customization lifecycle. However, web scraping by demonstration is not powerful enough to perform certain complex web scraping tasks. On the other side, web scraping by programming involves programmers manually writing web scraping code. It is powerful enough to scrape a wider variety of websites but is not accessible to end-users who can only customize websites that programmers have written web scraping code for.

In this paper, we present a new interaction model for web scraping for customization that combines the ease of use of programming-by-demonstration and the expressiveness of traditional programming. At its core lies a simple formula language. The language offers pure functional semantics for expressing and executing data extractions from websites using Cascading Style Sheet (CSS) selectors. When a user demonstrates examples of data to scrape, a web scraping program is synthesized via wrapper induction [18] and is presented as a formula containing the CSS selector required to identify the Document Object Model (DOM) element with the data (Section 3). Crucially, the formula can be directly edited and executed to specify complex scraping operations which can not be achieved by demonstrations alone.

To illustrate our new interaction model for web scraping, we have implemented it as an extension of Wildcard [23, 24] called Joker. Wildcard is a browser extension that enables web customization by direct manipulation of a spreadsheet-like table it adds to websites. The table contains the website's underlying structured data and is bidirectionally synchronized with it. User interactions with the table, including sorting, adding columns and running computations in a spreadsheet formula language, are propagated to the website thereby customizing it. The web scraping required to create Wildcard's table can only be achieved via traditional programming in Javascript. This means that end-users can only customize websites that programmers have written web scraping code for. Joker enables end-users to achieve the web scraping tasks required for customization while maintaining some of the expressiveness of traditional programming.

By representing web scraping programs as formulas, our web scraping interaction model fit right into Wildcard's customization paradigm which utilizes formulas for web customization. This enabled us to go beyond providing an interaction model that combines web scraping by demonstration with web scraping by programming to providing an interaction model that combines web scraping *and* customization. We refer to this as a *unified interaction model for web scraping and customization*. In Section 2, we show how this offers more expressive web scraping and customization.

Our contributions are as follows:

- A **unified interaction model for web scraping and customization** that combines web scraping (by demonstration and programming) with web customization through a shared formula language with pure functional semantics
- An implementation of the unified interaction model for web scraping and customization called Joker that combines three key design principles (**prodirect manipulation, functional reactive programming & unified user interaction**) in a novel way (Section 4)

- The results of an evaluation of the model via an example gallery, a five person user study and a Cognitive Dimensions of Notion analysis (Section 5)

We end by discussing opportunities for future work and how we can make CSS selectors more accessible to end-users without development experience (Section 7).

2 EXAMPLE USAGE SCENARIO

To concretely illustrate the user experience of our tool, we present a scenario of customizing eBay, a popular online marketplace. The goal of the user, Jen, is to filter out search results that are sponsored, i.e. product listings that a seller paid to promote. On the webpage, sponsored listings are marked with a "Sponsored" label that Jen can try to scrape, then she can sort the scraped labels to move all sponsored listings to the bottom of the page. Figure 1 shows accompanying screenshots.

Scraping Names by Demonstration (Figure 1 Part A): Jen starts the adapter creation process by clicking a context menu item within the eBay.com page, then hovering over the data that she would like to scrape: the page element that contains the name of the listing. The system provides live feedback as Jen hovers: it annotates the row of data with a border, highlights the column of data in the page with a green background, and displays how the values will appear in the data table. When she clicks, the highlighted data is saved into the table.

The text content of the highlighted element is now displayed in the first column of the table, with one cell for each listing containing the name of the listing. When she clicks on a cell, Jen can see that the textual data is also represented by a formula:

```
QuerySelector(rowElement, "h3.s-item__title")
```

This formula reveals how Joker is actually scraping this column's data behind the scenes: it calls a query selector on each listing (i.e. each `rowElement`) on the page. Scraped data is stored as functions that act on the data on the page, instead of as static text, based on our design principle of **functional reactive programming**. Joker exposes this formula to the user to exemplify how scraping is achieved with formulas and to provide a template for scraping with formulas.

After this first demonstration, Jen has a reference to each of the page's row elements (i.e. each product listing) in the Joker table. She can now use these references to scrape elements within the product listing with formulas, such as the Sponsored label.

Scraping Sponsored Labels with Formulas (Figure 1 Part B): Next, Jen attempts to scrape contents of the "Sponsored" label element into a new column; having this column would allow her to sort the table based on whether the listing contains the "Sponsored" label. She tries to scrape the label using

A) Scrapping Names by Demonstration: (1) The user scrapes the name of each listing by clicking on the name on the webpage. (2) The table reveals the underlying formula that was synthesized to scrape this data.

Alt + click (option instead of alt on Mac) on a field you wish to scrape

Restart Cancel Done

Best Match

Shipping to: 02139

70,463 results for laptop

Price: Under \$110.00 \$110.00 to \$230.00 Over \$230.00

Operating System: Not Included Windows 10 S Windows 10 Windows 7 Windows XP

Sponsored

Dell Chromebook 3120 11.6" | Intel Celeron N2840 2.16GHz | 4GB RAM | 16GB SSD

Refurbished - Dell - 16 GB

\$85.99

Free 4 day shipping

Free returns

Top Rated Plus

Edit Wildcard Table

Close Wildcard Table

Wildcard vol. =QuerySelector(rowElement, "h3.s-item_title")

	f(x) A	B
1	Dell Chromebook 3120 11.6" Intel Celeron N2840 2.16GHz 4GB RAM 16GB SSD	
2	DELL LAPTOP LATITUDE WINDOWS 10 CORE 2 DUO 4GB RAM WIN DVD WIFI PC HD C	
3	Dell Chromebook 3120 11.6" Celeron N2840 2.16GHz 16GB SSD 4GB RAM	
4	HP ProBook 655 G1 15.6" Laptop AMD CPU 2.5GHz 4GB 250GB Windows 10	
5	HP - 14" Laptop - AMD Athlon Silver - 4GB Memory - 128GB SSD - Jet Black	

B) Scrapping Sponsored Labels with Formulas: The user cannot scrape the "Sponsored" label by demonstration, but the user can enter a web scraping formula instead. (3, 4) The user determines the CSS selector that corresponds to the "Sponsored" label by using the developer console, and uses it in the formula.

Alt + click (option instead of alt on Mac) on a field you wish to scrape

Restart Cancel Done

Best Match

Shipping to: 02139

70,463 results for laptop

Price: Under \$110.00 \$110.00 to \$230.00 Over \$230.00

Operating System: Not Included Windows 10 S Windows 10 Windows 7 Windows XP

Sponsored

Dell Chromebook 3120 11.6" | Intel Celeron N2840 2.16GHz | 4GB RAM | 16GB SSD

Refurbished - Dell - 16 GB

\$85.99

Free 4 day shipping

Free returns

Top Rated Plus

757 sold

DELL LAPTOP LATITUDE WINDOWS 10 CORE 2 DUO 4GB RAM

Wildcard vol. =QuerySelector(rowElement, "div.s-item_title-tagblock")

	f(x) A	f(x) B	C
1	Dell Chromebook 3120 11.6" Intel JS3onMsoV3rXNYFedZ8		
2	DELL LAPTOP LATITUDE WINDOWS 10 CORE 2 DUO 4GB RAM		
3	Dell Chromebook 3120 11.6" Celeron N2840 2.16GHz 16GB SSD 4GB RAM		
4	HP ProBook 655 G1 15.6" Laptop AMD CPU 2.5GHz 4GB 250GB Windows 10		
5	HP - 14" Laptop - AMD Athlon Silver - 4GB Memory - 128GB SSD - Jet Black		

C) Filtering Sponsored Results: (5) Autocomplete with documentation helps the user write a formula that determines whether the "Sponsored" label contains garbled text. (6) When the user sorts the table by this column, the web page is sorted such that all non-sponsored results appear at the top.

Alt + click (option instead of alt on Mac) on a field you wish to scrape

Restart Cancel Done

Best Match

Shipping to: 02139

70,463 results for laptop

Price: Under \$110.00 \$110.00 to \$230.00 Over \$230.00

Operating System: Not Included Windows 10 S Windows 10 Windows 7 Windows XP

Sponsored

Dell Chromebook 3120 11.6" | Intel Celeron N2840 2.16GHz | 4GB RAM | 16GB SSD

Refurbished - Dell - 16 GB

\$85.99

Free 4 day shipping

Free returns

Top Rated Plus

757 sold

DELL LAPTOP LATITUDE WINDOWS 10 CORE 2 DUO 4GB RAM

Wildcard vol. =Includes(B, "Sponsored")

	f(x) A	f(x) B	f(x) C	D
1	HP Chromebook 14 Intel Celeron N3350 4GB RAM 32GB eMMC Chalkboard Gray			
2	EVOO 15.6" FHD Thin Laptop Black 1080p			
3	Lenovo IdeaPad Gaming 3 15.6" Laptop 120i			
4	NEW HP 14 Intel Dual Core 2.6GHz 64GB S			
5	NEW EvoO 15.6" FHD Ultra-Thin Notebook			

Figure 1: Scrapping and customizing eBay by unified demonstration and formulas.

demonstration, but she is unable to scrape more than one letter of the word at a time. To diagnose the issue, Jen inspects the page's source code using her browser's developer tools. Jen discovers that eBay's developers have inserted invisible letters into word "Sponsored" (possibly to obfuscate against ad blockers). Each letter is in its own HTML element, and the inserted letters are rendered invisible by CSS. Scraping by demonstration does not work in this case because Jen wants the element that contains the whole word, but the system is giving her the leaf-node elements (individual letters) instead.

Jen sees in the source code that the all of the letters of the "Sponsored" label, both visible and invisible, are contained in a single ancestor element with the CSS selector `"div.s-item__title--tagblock"`. Thus, she is able to scrape the full word by writing a query selector formula with that CSS selector. She can copy the formula from the previous column as a template.

This formula populates the column with the text content of the "Sponsored" element, which she can now use for string manipulation and sorting. By writing a formula, Jen is able to overcome the limitations of scraping by demonstration. The mixed methods of scraping exemplify our tool's support of **prodirect manipulation**.

Filtering Sponsored Results (Figure 1 Part C): The query selector that Jen just wrote returns all of the text within each of the targeted "Sponsored" elements, including any invisible letters. eBay's web design is that sponsored listings have a visible "Sponsored" label with invisible inserted letters, and non-sponsored listings have an invisible "Sponsored" label. Thus, for sponsored listings, the query returns garbled text (e.g. "JSp3onMsoV3rXNYFedZB"), and for non-sponsored listings, the query returns "Sponsored." Jen identifies this correlation by scrolling through the scraped data in the column and comparing them to what she sees on the web page. Then, in a new column, Jen writes a formula that returns whether or not the previous column's text includes the word "Sponsored." Finally, she sorts the listings by whether or not they are sponsored by sorting this column, thus hiding sponsored results from view. This customization is possible because Joker provides a **unified user model**, where scraping and customization are performed in conjunction.

In this way, Jen is able to use our system to customize the eBay website, without needing to learn how to program in JavaScript and without even leaving the webpage. Our model of customization by unified demonstration and formulas is flexible enough to support a wide range of other useful modifications and web programming proficiency levels, and we present a greater variety of use cases in Section 5.

3 SYSTEM IMPLEMENTATION

In this section, we outline the *wrapper induction* [18] algorithm that Joker uses to synthesize web scraping programs from user demonstrations. Then, we briefly describe the formula language used to represent the synthesized web scraping programs and include a list of the formulas that are currently available and their roles.

3.1 Wrapper Induction

In order to create web scraping programs from users demonstrations, Joker solves the wrapper induction [18] task: generalizing from a few examples of data in a data set to a specification for all the data in the data set.

Joker takes an approach similar to that used in systems like Vegemite [22] and Sifter [16]. It synthesizes a single *row selector* for the website: a CSS selector that identifies a set of DOM elements corresponding to the rows of the data set. For each column in the data set, it synthesizes a *column selector*, a CSS selector that identifies the element containing the column value.

One important difference is that our algorithm only accepts row elements that have direct siblings with a similar structure. We refer to this as the *row-sibling* constraint. Later, we describe how this constraint provides a useful simplification of the wrapper induction task and in [?] discuss the resulting limitations this puts on our system. We proceed to describe how CSS selectors are synthesized for row and column elements and then explain the criteria used to determine row elements.

3.1.1 Synthesizing CSS Selectors. Joker synthesizes two types of CSS selectors: a single row selector that selects a set of DOM elements corresponding to the rows of the data set and a column selector for each column which selects the element containing the column value within a given row.

For a given row element, its row selector is synthesized using the following criteria:

Plausibility. A selector is set of classes. It is a plausible row selector if it 1) consists of a subset of the classes on the row element and 2) consists of a subset of the classes all the row element's siblings. The second requirement is the *row-sibling* constraint we mentioned. Notice how it simplifies the problem by eliminating selectors. It is necessary for the end goal of customization because of customizations like sorting. If row elements are not siblings, the website would end up in an invalid state when the DOM elements are re-organized to match the sorting order.

Weight. A selector has a weight equal to the number of classes it consists of.

Best. A selector is the best if it is plausible and there is no other selector that has a lower weight than it. We favor selectors with the lowest weight to ensure that only the

minimum required classes are utilized. If there are multiple selectors that are plausible and have the lowest weight, we only pick one.

For a given column element, its column selector is synthesized using the following criteria:

Plausibility. A selector is a plausible column selector if it 1) consists of a subset of the classes on the column element and 2) only selects the give column element when applied on the corresponding row element.

Weight. A selector has a weight equal to the number of classes it consists of.

Best. A selector is the best if it is plausible and there is no other selector that has a lower weight than it has. As before, we favor selectors with the lowest weight and only pick one if there are multiple that fulfill the criteria.

One aspect of future work is saving the list of all selectors that fulfill the criteria and making them available to users to view and pick from via the formula. This would be similar to Mayer et al's user interaction model called *program navigation* [25] that gives users the opportunity to navigate all valid, synthesized programs and pick the best one.

3.1.2 Determining Row Elements. When a user first demonstrates a column value, Joker uses the demonstration to synthesize a row selector. The row selector will be used to identify all the row elements in the website. Additionally, a column selector is synthesized that will be used to identify the element that contains the column value. During subsequent demonstrations, Joker simply synthesizes a column selector for the column element that contains the demonstrated column value. Like similar approaches [10, 16, 22], all demonstrations have to be made from the same row element.

Given a demonstrated column value, row elements are determined using the following criteria:

Plausibility. An element R is a plausible row element if 1) it is within the BODY element of the DOM, 2) it is in the parent path of the column element C containing the demonstrated column value V and 3) the CSS selector S of element C identifies a column element C' when applied to R

Weight. A row element R has a weight W equal to the number of its siblings for which the CSS selector S of column element C only identifies C when applied to it. This is an enforcement of the *row-sibling* constraint. We want to find an R with the most number of siblings with a similar structure

Best. A row element R is the best if it is plausible and there is no other row element that has a higher weight than it. We favor row elements with the highest weight to ensure that we end up with a data set with the highest number of column values corresponding to the column that V belongs to. If there are multiple plausible row elements with the highest weight, we pick the one closet to the column element C in its parent path.

Figure 2 provides a concrete example of how the above criteria are applied to determine a row element from the demonstration of a column value.

3.2 Web Scraping Formulas

Joker's formula language is similar to that of visual database query systems like SIEUFERD [7] and Airtable [5]. Formulas automatically apply across an entire column of data and reference other column names instead of values in specific rows. This is more efficient than users having to copy a formula across a column as in traditional spreadsheets like Microsoft Excel and Google Sheets. It of course comes at the cost of not being able to specify a formula for only a subset of column cells but this hasn't yet come up in our use cases. The language currently consists of the following formulas:

3.2.1 QuerySelector(rowElement, selector). This formula is used to represent the web scraping program synthesized from demonstrations. rowElement is a special keyword that references a hidden column containing the DOM elements that correspond to the rows of the data set. selector is the synthesized CSS selector that specifies the column element.

3.2.2 GetParent(element). This formula is used to traverse the DOM when the data to be scraped is made of the values of its containing and sibling elements. Demonstrations alone cannot be used to scrape such data. element can be a reference to a column containing a QuerySelector formula (GetParent(A)) or a QuerySelector formula itself (=GetParent(QuerySelector(...))).

3.2.3 GetAttribute(element, attribute). This formula is used to scrape data from DOM attributes. An example of this are URLs which are available on the href attribute of link elements. element is as described for GetParent and attribute is the name of the attribute to scrape (e.g. GetAttribute(A, "href")).

4 DESIGN PRINCIPLES

Below, we describe three design principles that Joker embodies. We did not invent these principles but rather combined them in a novel manner for the domain of web scraping and web customization.

4.1 Prodirect Manipulation

Prodirect Manipulation is a term coined by Ravi Chugh in a position paper [11] in which he advocates for "novel software systems that tightly couple programmatic and direct manipulation." This principles embodies our goal of an interaction model for web scraping that combines the ease of use of programming-by-demonstration and the expressiveness of traditional programming.

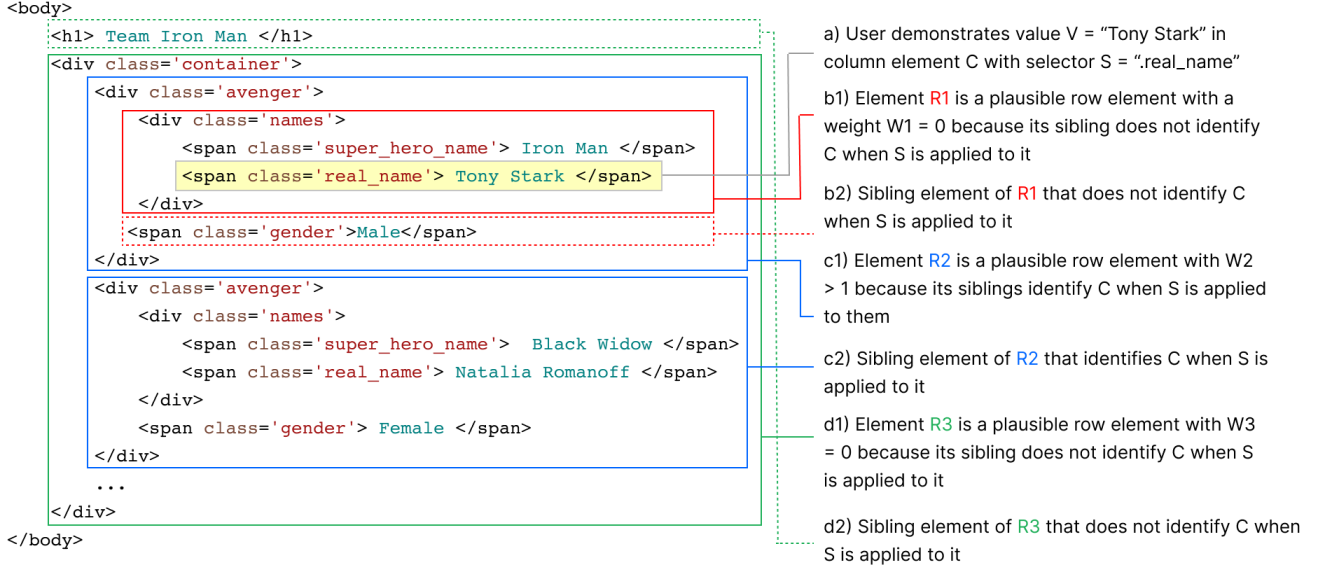


Figure 2: An example of how Joker’s wrapper induction algorithm is used to determine the row element from the demonstration of a column value. The row element is correctly determined to be R2 because it is plausible and has the highest weight.

In the eBay example in Section 2, we show how the “Sponsored” label of a listing can not be scraped by demonstration alone. However, through Joker’s prodirect manipulation interaction, a user can either directly edit the synthesized web scraping formula or author one from scratch to achieve the task. This is significant because the resulting customization to sort the listings by whether they are sponsored or not would not otherwise be possible

We can see prodirect manipulation in other programming-by-example systems. Sketch-N-Sketch [12] allows users to create an SVG shape via traditional programming and then switch to modifying its size or shape via direct manipulation. Wrex [13] takes examples of data transforms and generates readable and editable Python code.

4.2 Functional Reactive Programming

Functional reactive programming (FRP) utilizes pure functions to specify operations in a reactive programming paradigm. FRP has seen wide adoption in end-user programming through implementations such as spreadsheet formula languages (Microsoft Excel & Google Sheets) and formula languages for low-code programming environments (Microsoft Power Fx [6], Google AppSheet [1], Airtable [5], Glide [2], Coda [3] & Gneiss [9]).

Joker’s use of FRP makes it easier for users to specify scraping operations programmatically, thus making our web scraping interaction model more viable. With traditional programming, the eBay example in Section 2 in which demonstrations are not enough to scrape a listing’s “Sponsored” label would be much more complicated. The user would need to understand programming constructs such as state, variables, looping and data flow in order to write a program to scrape the label. In Joker, all a user needs to specify is the CSS selector that identifies the element containing the label. Joker takes care of managing state, variables, looping and data flow!

A key limitation of this approach is that users need to understand how CSS selectors work. Because the formula language utilizes pure functional semantics, users can iterate on CSS selectors as many times as they need to without having to worry about side effects. This makes the authoring of CSS selectors more accessible but more work remains for us to make this fully accessible to end-users.

4.3 Unified User Interaction

Prior to this work, web scraping and web customization in customization systems [16, 22] were divided: web scraping had to be performed prior to customization in a separate phase. Vegemite [22], a system for end-user programming

of mashups, reported findings from its user study in which participants thought that “it was confusing to use one technique to create the initial table, and another technique to add information to a new column.”

Joker represents web scraping and customization using the same formula language. Because of this, both can be performed in a single phase, with users being able to interleave the two as desired. This makes the process of customization more incremental which is vital for complex tasks. We can see this in the Ebay example in Section 2: when the user scrapes a listing’s “Sponsored” label, they observe that non-sponsored listings have an invisible “Sponsored” label while sponsored listings have a visible “Sponsored” label that consists of a garbled form “Sponsored.” Because of Joker’s unified user interaction, the user receives the results of scraping in the table and can immediately write a customization formula to validate this hypothesis. Without the unified interaction, the user would have to scrape all the desired columns to see all their values before ever getting to notice the pattern and validate their hypothesis.

5 EVALUATION

In this section, we evaluate Joker with an example gallery of websites, a small user study and a Cognitive Dimensions of Notion analysis [8]. The example gallery provides examples of websites that can be scraped and customized by Joker and known limitations. The user study provides insights on which aspects of Joker work well and which don’t. The Cognitive Dimensions of Notion analysis provides thoughts on how Joker’s design compares to similar approaches and interaction standards.

5.1 Example Gallery

Following a method used to evaluate visualizations through a diverse gallery of examples [27], our first evaluation of Joker provides an example gallery of popular websites on which Joker can be used for web customization and on which it can not be used. For the websites on which Joker can be used, we provide the sequence of interactions needed to achieve the customizations. For the websites on which Joker cannot be used, we provide an explanation.

5.1.1 Websites Joker Can Be Used On. We have used Joker to achieve a variety of purposes across many popular websites. For example, we have used Joker to sort search results by price within the Featured page on Amazon. (Using Amazon’s sort by price feature often returns irrelevant results.) In Amazon’s source code, the price is split into three HTML elements: the dollar sign, the dollar amount, and the cents amount. A user can only scrape the cents element by demonstration into column A. However, because the parent element of the cents element contains all three of the price elements, the user

can scrape the full price using the formula `GetParent(A)`. Next, the user can write the formula `ExtractNumber(B)` to convert the string into a numeric value. Finally, the user can sort this column by low-to-high prices. In a similar manner, we have used Joker to scrape and sort prices and ratings on the product listing pages of Target and eBay.

We have also found Joker to be useful for filtering based on text inputs. For example, we have used Joker to filter the titles of a researcher’s publications on their Google Scholar profile. Specifically, a user can first scrape the titles into column A by demonstration. Then, the user can write the formula `Includes(A, "compiler")` that returns whether or not the title contains the keyword “compiler.” Finally, the user can sort by this column to get all of the publications that fit their constraint at the top of the page. We have also used Joker to filter other text-based directory web pages such as Google search results and the MIT course catalog, in similar ways.

Additionally, we have used Joker to augment web pages with external information. For example, Joker can augment Reddit’s old user interface, which has a list of headlines with links to articles and images. A user can first scrape the headline elements into column A by demonstration. The user can then extract the link into column B with the formula `GetAttribute(A, "href")`. Then, the user can write the formula `ReadTimeInSeconds(B)` that calls an API that returns the links’ read times. Similarly, the user can write the formula `Visited(B)` that returns whether that link has been visited in the user’s browser history. The user can also scrape elements such as the number of comments and the time of posting and sort by these values. We have performed similar customizations on websites such as ABC news.

5.1.2 Limitations. Joker’s unified model of demonstrations and formulas is most effective on webpages with data that is presented as many similarly-structured HTML elements. However, certain websites have designs that make it difficult for Joker to scrape data. These are some of those designs:

- *Multiple row elements.* The layout of some web pages has multiple types of row as siblings that contain different children elements. For example, the news aggregator website HackerNews has a page design that alternates between rows containing a title and rows containing supplementary data (e.g. number of likes and the time of posting). Because Joker only chooses a single row selector, when scraping by demonstration, Joker will only select one of the types of rows, and elements in the other types of rows will not be able to be scraped.
- *Infinite scroll.* Some web pages have an “infinite scroll” feature that adds new entries to the page when a user scrolls to the bottom. Joker’s table will only contain

elements that were rendered when the table was first created. Additionally, for websites with many elements, such as Facebook, Joker might run out of memory while running its wrapper induction algorithm and crash the page.

- *Data hidden behind an interaction.* On some sites, a user must click on an element to reveal data corresponding to that entry (e.g. time of posting, the author). However, Joker is restricted to scraping what is visible on the page at one point in time.

5.2 User Study

Our second evaluation of Joker is a user study which provides insights on what aspects of Joker work well and what don't. Because of this, successful completion of tasks does not equal validation of Joker. It does however inform us about usability issues that limit its effectiveness.

5.2.1 Participants. We recruited 5 participants with backgrounds ranging from limited programming experience to Software Engineers. All participants were familiar with Microsoft Excel spreadsheets but not all had used Excel spreadsheet formulas. 4 of the participants had web development experience with 3 of them having extensive experience. 3 of the participants had web scraping experience with only 1 having extensive experience.

5.2.2 Tasks. The participants completed 7 tasks across 2 websites towards the goal of web customization. The first website was MIT's course catalog and the second was a listing of iPhones after searching for "iPhone" on eBay. The tasks were as follows:

MIT Course Catalog. This set of tasks involved web scraping by demonstration and the use of web customization formulas: 1a) Scrape course titles, 1b) Scrapes course prerequisites, 1c) Add a column that indicates whether a course has a prerequisite & 1d) Add a column that indicates whether a course does not have a prerequisite and is offered in the fall.

eBay. This set of tasks involved web scraping by demonstration, the use of web scraping formulas and the use of web customization formulas: 2a) Scrape iPhone listing title, 2b) Scrape iPhone listing price, & 2c) Create a column that indicates whether an iPhone listing is sponsored.

5.2.3 Protocol. All participants attempted the MIT course catalog and eBay tasks in the order we have described. We started each session with a description of Joker and provided a brief tutorial of its main features on a website not used for the tasks. There was no time limit for tasks but we provided hints whenever a participant had exhausted the knowledge available to them.

5.2.4 Results. All participants were able to complete all the tasks with the help of hints when they got stuck and were unable to make progress on their own. Again, this does not equal validation. We describe the results along the following dimensions:

Web Scraping By Demonstration. All participants were able to complete the tasks that only involved scraping by demonstration (1a, 1b & 2a) quickly and without any hints. For tasks that involved switching between demonstrations and writing formulas, there was some confusion about what column demonstrated values would be scraped into. The active column is managed by a toolbar at the top of the website but participants hardly noticed it and assumed all operations had to be performed on the table.

Web Scraping Formulas. Participants were unsure why demonstrations did not work when completing tasks that required using web scraping formulas (2b, 2c). As a result, they wasted time attempting to demonstrate using various means. The participant with no web development experience had the hardest time as expected but was able to utilize a series of hints we provided to accomplish 2b. The rest of the participants were able to accomplish 2b with minimal hints but required a lot more when completing 2c.

Web Customization Formulas. Participants had varying degrees of trouble when completing the tasks that involved using web customization formulas (1c, 1d, 2c). Most of the confusion resulted from formula parameters and return values. The formula bar has an autocomplete feature that shows the documentation for a given formula. The documentation consists of the parameters a formula it takes and a brief description of each parameter. However, this wasn't always sufficient to enable participants to use formulas correctly.

5.2.5 Discussion. One participant, upon seeing our tutorial on scraping by demonstration, said "that's like black magic." Tasks that involved customization formulas took longer than those that involved demonstrations but this was mostly related to the formula documentation not providing enough usage information. More work is needed to show users what exactly a given formula does. When asked how they would accomplish the tasks without Joker, the participant with limited programming experience responded with "I don't think I would know how to do it." That Joker enabled them to achieve such a complex task is very promising!

Participants that were familiar with web development were able to take advantage of Joker's web scraping formulas. This included the participants that had limited web scraping experience and suggests that the pure functional semantics and use of CSS selectors for identifying elements to scrape makes the task easier. As predicted, participants not familiar with web development didn't have the required background to understand how CSS selectors work. This highlights the

need to further explore how web scraping formulas can be made more accessible to end-users.

All participants had trouble figuring out which column the demonstrated values would be scraped into, especially when demonstrations were interleaved with use of formulas. This points to the need for the switch between demonstrating and using formulas to be more tightly coupled visually. For web scraping formulas, the most notable issue was the lack of feedback about why demonstrations were not sufficient to scrape a value. As a result, users wasted time trying to demonstrate in various ways instead of thinking about using web scraping formulas to achieve the task. In order for traditional programming to augment demonstrations as we propose, users need to receive some kind of feedback that demonstrations are not sufficient.

5.3 Cognitive Dimensions Analysis

Our third evaluation of Joker analyzes it using Cognitive Dimensions of Notation [8], a heuristic evaluation framework that has been used to evaluate programming languages and visual programming systems [20, 28, 29]. When contrasting our tool with traditional scraping and other visual tools, we find particularly meaningful differences along several of the dimensions:

Progressive evaluation. In Joker, a user can see the intermediate results of their scraping and customization work at any point, and adjust their future actions accordingly. The table user interface makes it easy to inspect the intermediate results and notice surprises like missing values.

In contrast, traditional scraping typically requires editing the code, manually re-running it, and inspecting the results in an unstructured textual format, making it harder to progressively evaluate the results. Also, many end-user scraping tools [10, 22] require the user to demonstrate all the data extractions they want to perform before showing any information about how those demonstrations will generalize across multiple examples.¹

Premature commitment. Many scraping tools require making a *premature commitment* to a data schema: first, the user extracts a dataset, and then they perform downstream analysis or customization using that data. Wildcard suffered from this problem: when writing code for a scraping adapter, a user would need to try to anticipate all future customizations and extract the necessary data.

Our current system instead supports extracting data *on demand*. The user can decide on their desired customizations and data transformations, and extract data as needed to fulfill

¹This is an instance where Joker's limitation of only scraping a single page at a time proves beneficial. All the relevant data is already available on the page without further network requests, making it possible to support progressive evaluation with low latency. Other tools that support scraping across multiple pages necessarily require a slower feedback loop.

those tasks. There is never a need to eagerly guess what data will be needed in advance.

We have also borrowed a technique from spreadsheets for avoiding premature commitment: default naming. New data columns are automatically assigned a single-letter name, so that the user does not need to prematurely think of a name before extracting the data. We have not yet implemented the capability to optionally rename demonstrated columns, but it would be straightforward to do so, and would provide a way to offer the benefits of names without requiring a premature commitment.

Provisionality. Joker makes it easy to try out a scraping action without fully committing to it. When the user hovers over any element in the page, they see a preview of how that data would be entered into the table, and then they can click if they'd like to proceed. This makes it feel very fast and lightweight to try scraping different elements on the page.

Viscosity. Some scraping tools have high viscosity: they make it difficult to change a single part of a scraping specification without modifying the global structure. For example, in Rousillon [10], changing the desired demonstration for a single column of a table requires re-demonstrating all of the columns. In contrast, Joker allows a user to change the specification for a single column of a table without modifying the others, resulting in a lower viscosity in response to changes.

Role-expressiveness. One dimension we are still exploring in our tool is role-expressiveness: having different elements of a program clearly indicate their role to the user. In particular, in our current design, the visual display of references to DOM elements in the table is similar to the display of primitive values. In our experience, this can sometimes make it difficult to understand which parts of the table are directly interacting with the website, vs. processing the downstream results. In the future we could consider adding more visual differentiation to help users understand the role of different parts of their customization.

6 RELATED WORK

Joker builds on existing work in end-user web scraping, end-user web customization and program synthesis by a number of systems.

6.1 End-user Web Scraping

FlashExtract [19] is a programming-by-example tool for data extraction. In addition to demonstrating whole values, it supports demonstrating substrings of values. Joker only supports this through formulas. This is not as end-user friendly but allows for a wider range of operations such as indicating whether demonstrated values contain a certain value or are greater than or less than a certain value.

Rousillon [10] is a tool that enables end-users to scrape distributed, hierarchical web data. It presents the web scraping program generated by demonstration as an editable, high-level, block-based language called Helena [3]. While Helena can be used to specify complex web scraping tasks like adding control flow, it does not present the synthesized web scraping program. This means that users can only scrape what can be demonstrated. Joker on the other hand displays the synthesized program as a formula which can be modified to increase the expressiveness of scraping.

6.2 End-user Web Customization

Vegemite [22] is a tool for end-user programming of mashups. Unlike Joker, its table interface is only populated and can only be interacted with after all the demonstrations have been provided. This does not support interleaving of scraping and table operations to achieve an incremental workflow for users. Mashups are created using CoScripter [21] which records operations on the scraped values in the table for automation tasks. CoScripter provides the generated automation program as text-based commands, such as “paste address into ‘Walk Score’ input,” which can be edited via “sloppy programming” [22] techniques. However, this editing does not extend to the synthesized web scraping program which is not displayed and therefore cannot be modified. This means that users can only scrape what can be demonstrated.

Sifter [16] is a tool that augments websites with advanced sorting and filtering functionality. It attempts to automatically detect items and fields on the page with a variety of clever heuristics. If this fails, it gives the user the option of demonstrating to correct the result. In contrast, Joker is simpler and makes fewer assumptions about the structure of websites by giving control to the user from the beginning of the process and displaying the synthesized program which can be modified. We hypothesize that focusing on a tight feedback loop rather than automation may support a scraping process that is just as fast as an automated one, offers more expressive scraping and extends to a greater variety of websites. However, further user testing is required to validate this hypothesis.

6.3 Program Synthesis

FlashProg [25] is a framework that provides program navigation and disambiguation for programming-by-example tools like FlashExtract [19] and FlashFill [14]. The program viewer provides a high level description of synthesized programs as well as a way to navigate the list of alternative programs that satisfy the demonstrations. This is important because demonstrations are an ambiguous specification for program synthesis [26]: the set of synthesized programs for a demonstration can be very large. To further ensure that the

best synthesized program is arrived at, FlashProg has a disambiguation viewer that asks the user questions in order to resolve ambiguities in the user’s demonstrations. In contrast, Joker only presents the top-ranked synthesized program which may not be the best one. Furthermore, the program is presented in is low-level form as a CSS selector. This is not end-user friendly but allows for more expressiveness.

7 CONCLUSION AND FUTURE WORK

In this paper, we presented our unified interaction model for web scraping and customization through a browser extension called Joker. This model combines the ease of use of programming-by-demonstration and the expressiveness of traditional programming. At its core lies a simple formula language that offers pure functional semantics for expressing and executing data extractions from websites using Cascading Style Sheet (CSS) selectors.

The main area of future work involves providing feedback to users about when demonstrations are not sufficient. Otherwise, users won’t know to switch to formulas to take advantage of their increased expressiveness. FlashProg [25] and Wrangler [17], whose interfaces also have tables, offer clues for how to provide feedback about the execution of programs synthesized from demonstrations. Another important area involves making the formula language more accessible to end-users not familiar with CSS selectors. Again, FlashProg offers some clues through its use of high level descriptions to give meaning to synthesized programs. One avenue for associating meaning with CSS selectors could be by extracting semantic web content as seen in systems like Thresher [15].

REFERENCES

- [1] [n.d.]. *AppSheet: No-Code App Development | Google Cloud*. <https://cloud.google.com/appsheet>
- [2] [n.d.]. *Build an App from a Google Sheet in Five Minutes, for Free • Glide*. <https://www.glideapps.com/>
- [3] Coda | A new doc for teams. [n.d.]. *Coda | A New Doc for Teams*. Coda | A new doc for teams. <https://coda.io/welcome>
- [4] [n.d.]. *Greasespot*. <https://www.greasespot.net/>
- [5] [n.d.]. *Tampermonkey for Chrome*. <http://www.tampermonkey.net>
- [6] [n.d.]. *What Is Microsoft Power Fx?* <https://powerapps.microsoft.com/en-us/blog/what-is-microsoft-power-fx/>
- [7] Eirik Bakke and David R. Karger. [n.d.]. Expressive Query Construction through Direct Manipulation of Nested Relational Results. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco California USA, 2016-06-14). ACM, 1377–1392. <https://doi.org/10.1145/2882903.2915210>
- [8] A. F. Blackwell, C. Britton, A. Cox, T. R. G. Green, C. Gurr, G. Kadoda, M. S. Kutar, M. Loomes, C. L. Nehaniv, M. Petre, C. Roast, C. Roe, A. Wong, and R. M. Young. [n.d.]. Cognitive Dimensions of Notations: Design Tools for Cognitive Technology. In *Cognitive Technology: Instruments of Mind* (Berlin, Heidelberg, 2001), Meurig Beynon, Chrystopher L. Nehaniv, and Kerstin Dautenhahn (Eds.). Springer Berlin Heidelberg, 325–341.

- [9] Kerry Shih-Ping Chang and Brad A. Myers. [n.d.]. Creating Interactive Web Data Applications with Spreadsheets. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (Honolulu Hawaii USA, 2014-10-05). ACM, 87–96. <https://doi.org/10.1145/2642918.2647371>
- [10] Sarah E. Chasins, Maria Mueller, and Rastislav Bodik. [n.d.]. Rousillon: Scraping Distributed Hierarchical Web Data. In *The 31st Annual ACM Symposium on User Interface Software and Technology - UIST '18* (Berlin, Germany, 2018). ACM Press, 963–975. <https://doi.org/10.1145/3242587.3242661>
- [11] Ravi Chugh. [n.d.]. Prodirect Manipulation: Bidirectional Programming for the Masses. In *Proceedings of the 38th International Conference on Software Engineering Companion* (Austin Texas, 2016-05-14). ACM, 781–784. <https://doi.org/10.1145/2889160.2889210>
- [12] Ravi Chugh, Brian Hempel, Mitchell Spradlin, and Jacob Albers. [n.d.]. Programmatic and Direct Manipulation, Together at Last. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Santa Barbara CA USA, 2016-06-02). ACM, 341–354. <https://doi.org/10.1145/2908080.2908103>
- [13] Ian Drosos, Titus Barik, Philip J. Guo, Robert DeLine, and Sumit Gulwani. [n.d.]. Wrex: A Unified Programming-by-Example Interaction for Synthesizing Readable Code for Data Scientists. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu HI USA, 2020-04-21). ACM, 1–12. <https://doi.org/10.1145/3313831.3376442>
- [14] William R Harris and Sumit Gulwani. [n.d.]. Spreadsheet Table Transformations from Examples. ([n. d.]), 12.
- [15] Andrew Hogue and David Karger. [n.d.]. Thresher: Automating the Unwrapping of Semantic Content from the World Wide Web. In *Proceedings of the 14th International Conference on World Wide Web - WWW '05* (Chiba, Japan, 2005). ACM Press, 86. <https://doi.org/10.1145/1060745.1060762>
- [16] David F. Huynh, Robert C. Miller, and David R. Karger. [n.d.]. Enabling Web Browsers to Augment Web Sites' Filtering and Sorting Functionalities. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology - UIST '06* (Montreux, Switzerland, 2006). ACM Press, 125. <https://doi.org/10.1145/1166253.1166274>
- [17] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. [n.d.]. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In *Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems - CHI '11* (Vancouver, BC, Canada, 2011). ACM Press, 3363. <https://doi.org/10.1145/1978942.1979444>
- [18] Nicholas Kushmerick. [n.d.]. Wrapper Induction: Efficiency and Expressiveness. 118, 1 ([n. d.]), 15–68. [https://doi.org/10.1016/S0004-3702\(99\)00100-9](https://doi.org/10.1016/S0004-3702(99)00100-9)
- [19] Vu Le and Sumit Gulwani. [n.d.]. FlashExtract: A Framework for Data Extraction by Examples. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Edinburgh United Kingdom, 2014-06-09). ACM, 542–553. <https://doi.org/10.1145/2594291.2594333>
- [20] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. [n.d.]. Evaluation Strategies for HCI Toolkit Research. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC Canada, 2018-04-19). ACM, 1–17. <https://doi.org/10.1145/3173574.3173610>
- [21] Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. [n.d.]. CoScripter: Automating & Sharing How-to Knowledge in the Enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2008-04-06) (CHI '08). Association for Computing Machinery, 1719–1728. <https://doi.org/10.1145/1357054.1357323>
- [22] James Lin, Jeffrey Wong, Jeffrey Nichols, Allen Cypher, and Tessa A. Lau. [n.d.]. End-User Programming of Mashups with Vegemite. In *Proceedings of the 14th International Conference on Intelligent User Interfaces* (New York, NY, USA, 2009-02-08) (IUI '09). Association for Computing Machinery, 97–106. <https://doi.org/10.1145/1502650.1502667>
- [23] Geoffrey Litt and Daniel Jackson. [n.d.]. Wildcard: Spreadsheet-Driven Customization of Web Applications. In *Companion Proceedings of the 4th International Conference on the Art, Science, and Engineering of Programming* (Porto, Portugal., 2020). Association for Computing Machinery, 10. <https://doi.org/10.1145/3397537.3397541>
- [24] Geoffrey Litt, Daniel Jackson, Tyler Millis, and Jessica Quaye. [n.d.]. End-User Software Customization by Direct Manipulation of Tabular Data. In *Proceedings of the 2020 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (Virtual USA, 2020-11-18). ACM, 18–33. <https://doi.org/10.1145/3426428.3426914>
- [25] Mikael Mayer, Gustavo Soares, Maxim Grechkin, Vu Le, Mark Marron, Aleksandr Polozov, Rishabh Singh, Benjamin Zorn, and Sumit Gulwani. [n.d.]. User Interaction Models for Disambiguation in Programming by Example. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (Charlotte NC USA, 2015-11-05). ACM, 291–301. <https://doi.org/10.1145/2807442.2807459>
- [26] Hila Peleg, Sharon Shoham, and Eran Yahav. [n.d.]. Programming Not Only by Example. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg Sweden, 2018-05-27). ACM, 1114–1124. <https://doi.org/10.1145/3180155.3180189>
- [27] Donghao Ren, Bongshin Lee, Matthew Brehmer, and Nathalie Henry Riche. [n.d.]. Reflecting on the Evaluation of Visualization Authoring Systems : Position Paper. In *2018 IEEE Evaluation and Beyond - Methodological Approaches for Visualization (BELIV)* (2018-10). 86–92. <https://doi.org/10.1109/BELIV.2018.8634297>
- [28] Arvind Satyanarayan and Jeffrey Heer. [n.d.]. Lyra: An Interactive Visualization Design Environment: Lyra: An Interactive Visualization Design Environment. 33, 3 ([n. d.]), 351–360. <https://doi.org/10.1111/cgf.12391>
- [29] Arvind Satyanarayan, Kanit Wongsuphasawat, and Jeffrey Heer. [n.d.]. Declarative Interaction Design for Data Visualization. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (Honolulu Hawaii USA, 2014-10-05). ACM, 669–678. <https://doi.org/10.1145/2642918.2647360>