Joker: A Unified Interaction Model For Web Customization

ACM Reference Format:
. 2021. Joker: A Unified Interaction Model For Web Customization. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Many websites do not meet the exact needs of all of their users, so millions of people use browser extensions and userscripts [4, 5] to customize them. However, these tools only allow end-users to install customizations built by programmers. End-user web customization systems like Sifter [15], Vegemite [18] and Wildcard [19] provide a more accessible approach, allowing anyone to create bespoke customizations without performing traditional programming.

These tools each provide different useful mechanisms for end-user customization, but they share a common design limitation: they have a rigid separation between the two stages of the web customization process. First, in the *extraction* or *scraping* phase, users get data from the website into a structured, tabular format. Second, in the *augmentation* phase, users perform augmentations like adding new columns derived from the data, or sorting the data. For example, in Vegemite, a user can extract a list of addresses from a housing catalog, and then augment the data by computing a walkability score for each address.

This separation between extraction and augmentation poses an important barrier to usability. A user study [18] of Vegemite wrote that "it was confusing to use one technique to create the initial table, and another technique to add information to a new column." The creators of Sifter similarly reported [15] that "the necessity for extracting data before augmentation could take place was poorly understood, if understood at all." In Wildcard, end-users cannot augment a website at all until a programmer has written and shared extraction code for that website in Javascript [19]. These tools all impose a sequential workflow in which users must first extract all the data they need, and then perform all their desired augmentations. This workflow exemplifies the more general problem in interface design of forcing users to make premature commitments to formal structure [8, 22].

In this paper, we present a new approach to web customization that combines extraction and augmentation in a unified interaction model. Our key idea is to develop a domain specific language (DSL) that encompasses both extraction and augmentation tasks, along with a programming-by-demonstration (PBD) interface that makes it easy for end-users to program in the language. This unified interaction model allows end-users to move seamlessly between extraction and augmentation, resulting in a more iterative and free-form workflow for web customization.

To demonstrate and evaluate this model, we have built a browser extension called Joker, an extension of the Wildcard customization tool. The original Wildcard system [19] adds a spreadsheet-like table to a website and establishes a bidirectional synchronization between the website and the table. This allows users to customize a website, by filtering and sorting page elements, and adding user annotations, derived values and calls to web services. Although Wildcard offers a declarative formula language for augmenting the page, a conventional imperative language (namely JavaScript) is used for the extraction step, and the extraction code cannot be modified during augmentation.

Joker makes two primary contributions:

A unified formula language for extraction & augmentation: Wildcard's formula language only supported primitive values like strings and numbers aimed at augmentation. Joker extends this language by introducing Document Object Model (DOM) elements as a new type of value, and adding a new set of formulas for performing operations on them. This includes querying elements with Cascading Style Sheets (CSS) selectors and traversing the DOM. With this approach, a single formula language is used to express both extraction and augmentation tasks, even within a single formula expression.

A PBD interface for creating extraction formulas: Directly writing extraction formulas can be challenging for end-users, so Joker provides a PBD interface that synthesizes formulas from user demonstrations. A key aspect of our design is that the program synthesized from the demonstration

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/nnnnnnn.nnnnnnn

is made visible as a spreadsheet formula that can be subsequently edited by the user, and is more easily understood than imperative code due to its declarative form.

Section 2 describes a concrete scenario, showing how Joker enables a user to complete a useful customization task. In Section 3, we outline the implementation of our formula language and user interface, as well as the algorithms used by our PBD interface. We evaluate our approach in Section 4 by describing a suite of case studies in which we used Joker to extract and augment a variety of websites in order to characterize its capabilities and limitations. Joker relates to existing work not only in end-user web customization, but also in end-user web scraping and program synthesis, which we discuss in Section 5. Finally, we discuss opportunities for future work in Section 6.

2 EXAMPLE USAGE SCENARIO

Here is an example scenario, illustrated in Figure 1, and demonstrated in the video accompanying this paper. Jen is searching for a karaoke machine on eBay, a shopping website. She wants to use Joker to sort products by price within a page of search results, a feature not supported by eBay.

Extracting Product Titles & Prices By Demonstration. (Figure 1 Part A): Jen initiates Joker through the browser context menu. As she hovers over the product title, Joker provides two kinds of live feedback. First, it highlights the titles of all the corresponding product listings on the page, to indicate how it has generalized Jen's intent based on her demonstration of a single example product. Second, a column of the table is populated with the values that will be extracted, giving a preview of how the extracted data would look. To commit to this extraction, Jen clicks on the product title. She repeats a similar process to extract the price.

When she clicks one of the cells in column `B`, the formula bar above the table displays the extraction formula that was generated from her demonstration:

`=QuerySelector(rowElement, "span.s-item__price")`

This formula produces the values in column `B`. For each row in the table, Joker has a created a reference to the corresponding DOM element, accessible through the special identifier `rowElement`. The `QuerySelector` function runs the specified CSS selector (`span.s-item__price`) within the row element to extract the price of each item. While Jen could directly edit this formula, in this case the values in the table are correct, so there's no need to edit the formula.

Sorting Products By Price. (Figure 1 Part B): Next, Jen wants to sort the table by price, but she realizes that the column of prices is a list of strings containing the `$` symbol. She needs to *augment* this raw data by turning the strings into numbers. In the next column (`C`), Jen starts typing a formula, and uses an autocomplete dropdown to find a relevant function that extracts numeric values from strings:

`=ExtractNumber(B)`

Now column C contains numeric values, so Jen can sort the table by price. Because the table is synchronized with the website, the product listings become sorted as well.

Extracting Product URL With Formulas (Figure 1 Part C): While completing the previous task, Jen realizes there is another customization she would find useful: prioritizing products for which she has not yet visited the details page for the listing. To do this, she must first return to *extracting* more relevant data from the page.

Each product listing links to a page for the specific product, but because the URL is not visible in the page, it's not possible to directly extract it by demonstration. However, Jen can still achieve the goal by directly writing an extraction formula.

Jen has some basic knowledge of HTML, which she can leverage to write the formula. She opens the browser developer tools, and observes that the listing title is represented by a link tag with a heading inside:

```
<a href="LISTING PAGE URL">
  <h3>LISTING NAME</h3>
</a>
```

Since there is already a column `A` in the table representing the product's title, she can use this as a starting point to write the formula:

`=GetAttribute(GetParent(A), "href")`

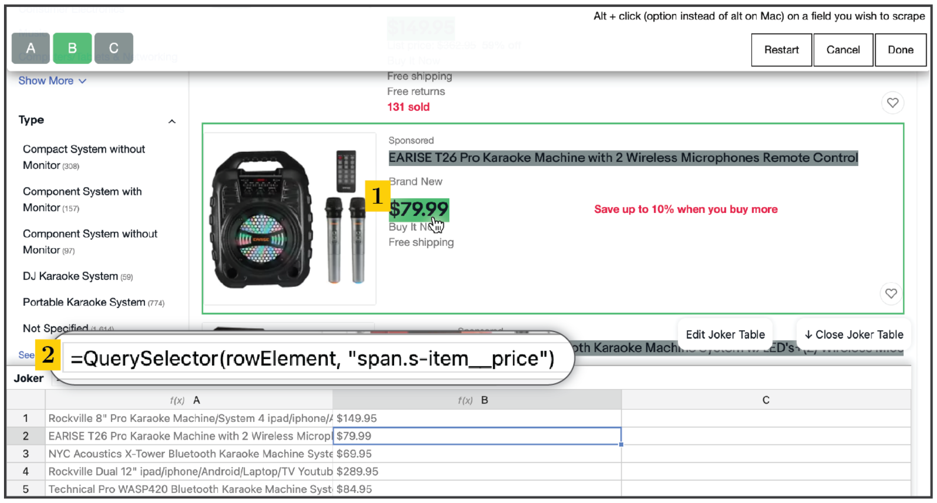
This formula first calls the `GetParent` function on column `A`, traversing up a level from the `<h3>` elements to the `<a>` elements. Then, `GetAttribute` extracts the `href` value containing the link URL. After running this formula, the table contains a column `D` with the URL for each product.

Sorting Products By Whether They Have Been Visited (Figure 1 Part C): After performing that extraction, Jen can write a final augmentation formula to indicate whether she has visited the corresponding product page:

`=Visited(D)`

The `Visited` function checks whether a URL is present in the browser history and returns a boolean value represented by a checkbox. Jen can then sort by this column to put listings that she has not yet visited at the top of the page.

Using Joker, Jen was able to not only achieve her initial customization goal to sort the products by price but also perform a customization she did not plan to do. This was made possible by Joker's unified interaction model for web customization which enabled her to interleave extraction and augmentation. While we have described only a single illustrative scenario in this example, Joker is flexible enough to support a wide range of other useful customizations and workflows on various websites, described in more detail in Section 4.



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

A B C Restart Cancel Done

Sponsored
EARISE T26 Pro Karaoke Machine with 2 Wireless Microphones Remote Control
Brand New
1 $79.99
Save up to 10% when you buy more

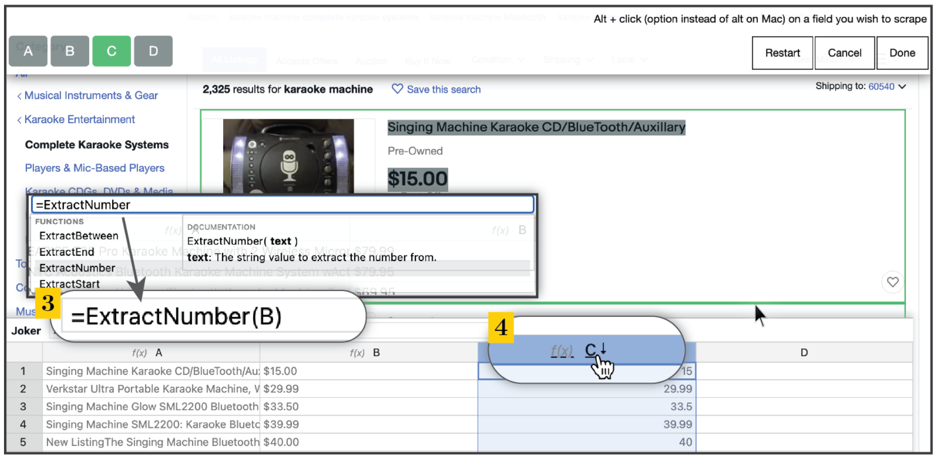
Edit Joker Table ↓ Close Joker Table

2 =QuerySelector(rowElement, "span.s-item__price")

Joker

	f(x) A	f(x) B	C
1	Rockville 8" Pro Karaoke Machine/System 4 ipad/iphone/A	$149.95	
2	EARISE T26 Pro Karaoke Machine with 2 Wireless Microp	$79.99	
3	NYC Acoustics X-Tower Bluetooth Karaoke Machine Syste	$69.95	
4	Rockville Dual 12" ipad/iphone/Android/Laptop/TV Youtub	$289.95	
5	Technical Pro WASP420 Bluetooth Karaoke Machine Syst	$84.95	

A) Extracting Names and Prices by Demonstration:
(1) The user scrapes the name and price of each listing by successively clicking on a name and price 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

A B C D Restart Cancel Done

2,325 results for **karaoke machine** Save this search

Singing Machine Karaoke CD/BlueTooth/Auxillary
Pre-Owned
$15.00

=ExtractNumber

FUNCTIONS
ExtractBetween
ExtractEnd
ExtractNumber
ExtractStart

DOCUMENTATION
ExtractNumber(**text**)
text: The string value to extract the number from.

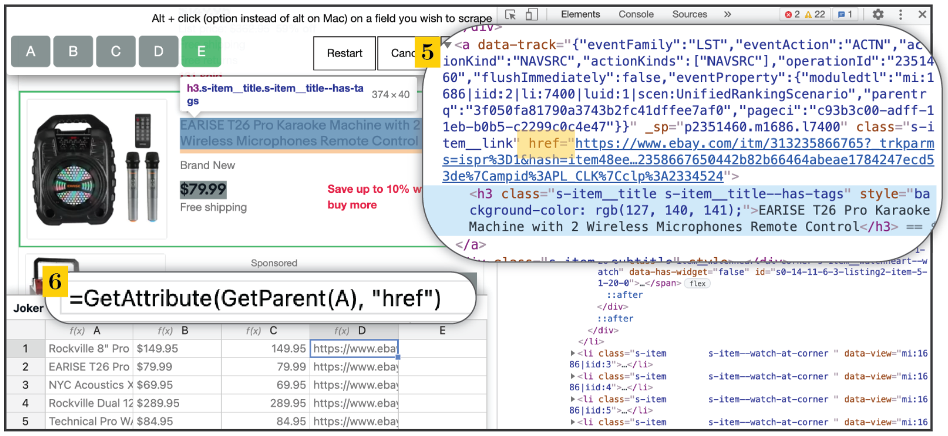
3 =ExtractNumber(B)

4

Joker

	f(x) A	f(x) B	f(x) C↓	D
1	Singing Machine Karaoke CD/BlueTooth/Au:	$15.00	15	
2	Verkstar Ultra Portable Karaoke Machine, V	$29.99	29.99	
3	Singing Machine Glow SML2200 Bluetooth	$33.50	33.5	
4	Singing Machine SML2200: Karaoke Bluetc	$39.99	39.99	
5	New ListingThe Singing Machine Bluetooth	$40.00	40	

B) Extracting the Number and Sorting by Price:
(3) Autocomplete with function documentation helps the user write a formula that converts the price from a string with a dollar sign into a numerical value.
(4) Now, the user can sort this column such that the rows are ordered by increasing numerical value. Correspondingly, the web page is sorted such that the lowest-cost results appear at the top.



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

A B C D E Restart Cancel

Elements Console Sources

h3.s-item__title.s-item__title--has-tags 374 × 40

EARISE T26 Pro Karaoke Machine with 2 Wireless Microphones Remote Control
Brand New
$79.99
Free shipping
Save up to 10% when you buy more

5

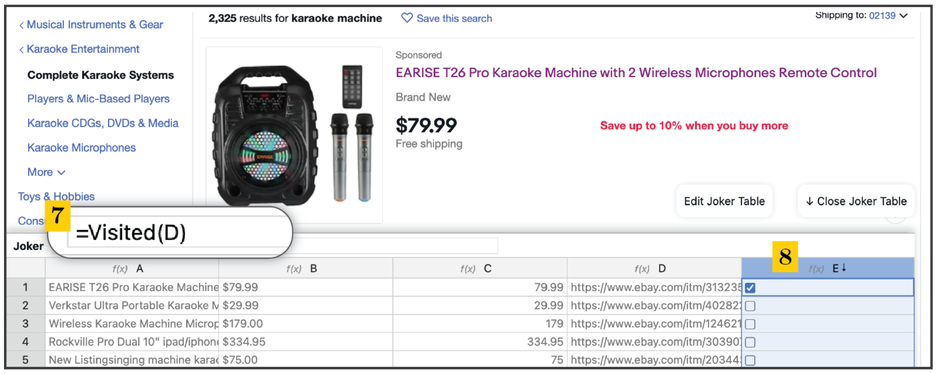
```
▼<a data-track="{"eventFamily":"LST","eventAction":"ACTN","ac
ionKind":"NAVSRC","actionKinds":["NAVSRC"],"operationId":"23514
60","flushImmediately":false,"eventProperty":{"moduledtl":"mi:1
686|iid:2|li:7400|luid:1|scen:UnifiedRankingScenario","parentr
q":"3f050fa81790a3743b2fc41dffee7af0","pageci":"c93b3c00-adff-1
1eb-b0b5-c2299c0c4e47"}}" _sp="p2351460.m1686.l7400" class="s-i
tem__link" href="https://www.ebay.com/itm/313235866765?_trkparm
s=ispr%3D1&hash=item48ee…2358667650442b82b66464abeae1784247ecd5
3de%7Campid%3APL_CLK%7Cclp%3A2334524">
   <h3 class="s-item__title s-item__title--has-tags" style="ba
   ckground-color: rgb(127, 140, 141);">EARISE T26 Pro Karaoke
   Machine with 2 Wireless Microphones Remote Control</h3> ==
</a>
```

6 =GetAttribute(GetParent(A), "href")

Joker

	f(x) A	f(x) B	f(x) C	f(x) D	E
1	Rockville 8" Pro	$149.95	149.95	https://www.eba	
2	EARISE T26 Pro	$79.99	79.99	https://www.eba	
3	NYC Acoustics X	$69.95	69.95	https://www.eba	
4	Rockville Dual 12	$289.95	289.95	https://www.eba	
5	Technical Pro W/	$84.95	84.95	https://www.eba	

C) Extracting Links using Formulas:
(5) The user wants to extract the link attached to the listing's name element, but the link cannot be extracted by demonstration. The user inspects the page to find that this is because the link is contained in the name element's parent.
(6) The user composes a function to get the parent element of the name element (in column A) and extract its link attribute ("href").



< Musical Instruments & Gear
< Karaoke Entertainment
Complete Karaoke Systems
Players & Mic-Based Players
Karaoke CDGs, DVDs & Media
Karaoke Microphones
More
Toys & Hobbies

2,325 results for **karaoke machine** Save this search

Sponsored
EARISE T26 Pro Karaoke Machine with 2 Wireless Microphones Remote Control
Brand New
$79.99
Free shipping
Save up to 10% when you buy more

Edit Joker Table ↓ Close Joker Table

7 =Visited(D)

Joker

8

	f(x) A	f(x) B	f(x) C	f(x) D	f(x) E↓
1	EARISE T26 Pro Karaoke Machine	$79.99	79.99	https://www.ebay.com/itm/313235	☑
2	Verkstar Ultra Portable Karaoke M	$29.99	29.99	https://www.ebay.com/itm/40282	☐
3	Wireless Karaoke Machine Microp	$179.00	179	https://www.ebay.com/itm/124621	☐
4	Rockville Pro Dual 10" ipad/iphon	$334.95	334.95	https://www.ebay.com/itm/30390	☐
5	New Listingsinging machine karac	$75.00	75	https://www.ebay.com/itm/20344	☐

D) Sorting to Filter by Unread Links:
(7) The user writes a formula that determines whether a link has already been visited in the user's browser history.
(8) The user can sort this column to see all listings with unread links at the top of the page, with already-read links at the bottom.

3 SYSTEM IMPLEMENTATION

In this section, we describe Joker's formula language in more detail. Then, we briefly outline the *wrapper induction* [16] algorithm that Joker's PBD interface uses to synthesize the row element and column selectors presented in formulas.

3.1 Extraction Formulas

The Wildcard customization tool includes a formula language for augmentation, including operators for basic arithmetic and string manipulation, as well as more advanced operators that fetch data from web APIs. As in other tabular interfaces like SIEUFERD [7] and Airtable [5], formulas apply to a whole column at a time rather than a single cell, and can reference other columns by name. Joker extends this base language with new constructs which enable it to apply to *data extraction* instead of just augmentation.

We added DOM elements as a data type in the language, alongside strings, numbers, and booleans. Because the language runs in a JavaScript interpreter, we simply use native JavaScript values to represent DOM elements in the language. DOM elements are displayed visually by showing their inner text contents. They can also be implicitly typecast to strings for use in other formulas; for example, a string manipulation formula like `Substring` can be called on a DOM element value, and will operate on its text contents.

We also added several functions to the formula language for traversing the DOM and performing extractions, summarized below with their types:

- `QuerySelector(el: Element, sel: string): Element`. Executes the CSS selector `sel` inside of element `el`, and returns the first matching element.
- `GetAttribute(el: Element, attribute: string): string`. Returns the value for an attribute on an element.
- `GetParent(el: Element): Element`. Returns the parent of a given element.

To extract data from a row, formulas need a way to reference the current row, so we added a construct to support this use case. Every row in the table maps to one DOM element in the page; we allow formulas to access this DOM element via a special keyword, `rowElement`. In some sense, `rowElement` can be seen as a hidden extra column of data in the table containing DOM elements.

While many more functions could be added to expose more of the underlying DOM API, we found that in practice these three functions provided ample power through composition. For example, in Section 2 we showed how `GetParent` and `GetAttribute` can be composed to traverse the DOM and extract the URL associated with a product listing.

By providing a single formula language to express extractions and augmentations, Joker enables a *unified interaction model* that supports interleaving the two. Furthermore, the formula language enables users to specify logic using pure, stateless functions that reactively update in response to upstream changes. This *functional reactive paradigm* is easier to reason about than traditional imperative programming, as demonstrated by the use of formulas by millions of end users in spreadsheet programs and end-user programming environments [1–3, 5, 6, 9].

3.2 Wrapper Induction

When users demonstrate a specific column value to extract, Joker must synthesize a program that reflects the user's general intent. This is an instance of the *wrapper induction* problem of synthesizing a web data extraction query from examples. Prior work on this topic [13, 16] prioritizes accuracy and robustness to future changes, which makes sense for a fully automated system, but can lead to very complex queries. In our work, we chose to prioritize the readability of queries by less sophisticated users, so that users can more easily author queries and repair them when they break. We implemented a set of heuristics inspired by Vegemite [18] for wrapper induction, illustrated in Figure 2.

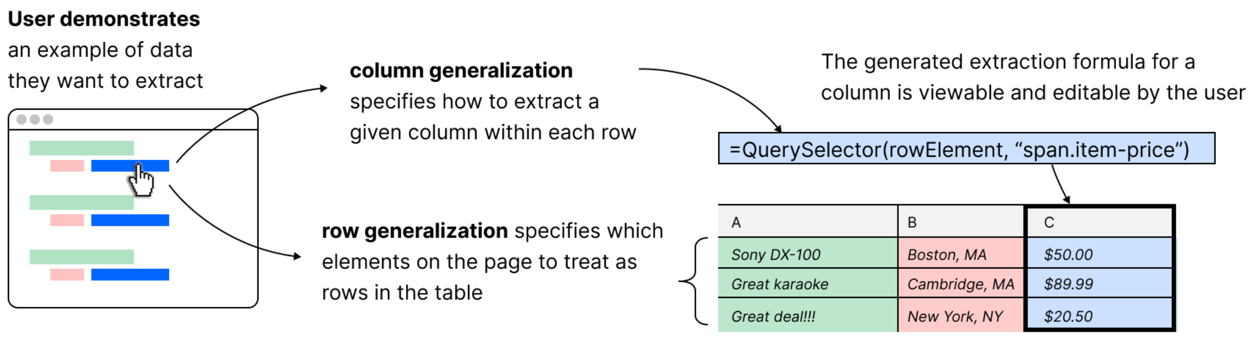
4 EVALUATION

We evaluate our interaction model and tool by describing the results of our use of Joker to extract data and perform customizations on popular websites. 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 fails, we explain the relevant limitations.

4.1 Successful Applications

We have used Joker to achieve a variety of customizations across many websites. Table 1 summarizes examples we have found on popular websites.

Sorting search results by price on Amazon. We have found Joker to be useful for sorting the contents of various websites. One example of a useful sort achieved by Joker is sorting 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 extract by demonstration only the cents element into column `A`. Subsequently, because the parent element of the cents element contains all three of the price elements, the user can extract 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


User demonstrates an example of data they want to extract
column generalization specifies how to extract a given column within each row
row generalization specifies which elements on the page to treat as rows in the table
The generated extraction formula for a column is viewable and editable by the user
=QuerySelector(rowElement, "span.item-price")

A	B	C
Sony DX-100	Boston, MA	$50.00
Great karaoke	Cambridge, MA	$89.99
Great deal!!!	New York, NY	$20.50

Figure 2: An overview of Joker's interaction model and wrapper induction process.

to extract and sort prices and ratings on the product listing pages of Target and eBay.

Filtering titles of publications on Google Scholar. We have also found Joker can be useful for filtering a website's listings based on the text content of an element in the listing. For example, we have used Joker to filter the titles of a researcher's publications on their Google Scholar profile which is not natively supported. First, a user can extract the titles into a 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.

Retrieving information about links on Reddit. Additionally, we have used Joker to augment web pages with external information. For example, Joker can augment Reddit's user interface, which has a list of headlines with links to articles, with the links' read times and whether the link has already been read. To achieve this customization, a user first extracts the headline elements into column (`A`) by demonstration. The user can then extract the link into the next 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)`, which uses another API that returns whether that link has been visited in the user's browser history. The user can also extract 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 and CNN.

4.2 Limitations

Joker is most effective on websites whose data is presented as a collection of similarly-structured HTML elements. Certain websites, however, have designs that make it difficult for Joker to extract data:

- *Heterogeneous row elements.* Some websites break their content into rows, but the rows do not have a consistent layout, and contain different types of child elements. For example, the page design of HackerNews 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 extracting by demonstration, Joker will only select one of the types of rows, and elements in the other types of rows will not be extracted.
- *Infinite scroll.* Some websites 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 that render a very large number of DOM elements, the speed of the live feedback provided by Joker's PBD interface might significantly decrease. This is because the wrapper induction process used by the PBD interface queries the DOM which takes longer as the size of the DOM increases.

5 RELATED WORK

5.1 End-user Web Customization

Joker builds on web customization ideas implemented by previous tools. Our contribution is a new interaction model that allows for interleaving extraction and augmentation, enabled by a unified formula language and a PBD interface.

Website	Example Customization Achieved by Joker
eBay, Amazon	Filter listings by whether they have a "Sponsored" label.
Amazon, Target	Sort search results by price and rating.
Google Scholar	Filter publications for those whose title contains a user-provided keyword.
Reddit, CNN, ABC	Sort by the read times of articles. Filter already-visited articles.
Weather.com	Filter hourly weather to find sunny times of day.
Github	Sort a user's code repositories by stars to find popular work.
Postmates, Uber Eats	Sort restaurants by delivery time and delivery fee.

Table 1: Examples of websites that Joker can be used to customize, including extraction and augmentation

Joker is an extension of the Wildcard customization system [19], and preserves its foundational idea of synchronizing a table with a website. Wildcard only allows for extraction logic to be written by programmers in Javascript; our work has substantially extended the Wildcard formula language and added an entire new system for dynamically creating data extraction logic within the user interface. We also improved the formula editing interface by adding an autocomplete dropdown and documentation popup, which proved important in our testing for allowing end-users to reliably edit and create formulas.

Vegemite [18] is a tool for end-user programming of web mashups. Like Joker, it allows users to perform demonstrations to extract data, but Vegemite only displays a table after all the demonstrations have been provided, which rules out interleaving extraction and augmentation. Vegemite does allow users to directly view and edit some of the logic generalized from demonstrations, but it only allows for editing augmentation logic, not extraction logic. The wrapper induction algorithm used in Joker is also very similar to Vegemite's algorithm.

Sifter [15] is a tool that augments websites with advanced sorting and filtering functionality. It attempts to automatically detect items and fields on the website with a variety of heuristics. If these fail, it gives the user the option of demonstrating to correct some parts of the result. In contrast, Joker makes fewer assumptions about the structure of websites, by giving control to the user from the beginning of the process and displaying an editable synthesized program.

5.2 End-user Web Scraping and Program Synthesis

Joker builds on insights from other tools that synthesize web scraping (i.e. data extraction) code from user demonstrations, and give users ways to inspect and modify the generated code.

Rousillon [10] is a tool that enables end-users to extract hierarchical web data across multiple linked web pages. It presents the web extraction program generated from demonstrations in an editable, high-level, block-based language called Helena [3]. While both Rousillon and Joker create an editable program, they have different focuses. Because Rousillon allows users to extract data across multiple pages (e.g., extracting details from each linked page in a list), it uses an imperative language, with nested loops as a key construct. In contrast, Joker can only extract within a single page, and therefore can use a simpler declarative formula language. Also, Rousillon only allows editing high-level control flow and treats some details of the extraction logic as opaque; Joker offers finer-grained control over details like CSS selectors.

Mayer et al propose a user interaction model called *Program Navigation* [20] which aims to give users another mechanism beside examples for guiding the generalization process of PBE tools like FlashExtract [17] and FlashFill [14]. This is important because demonstrations are an ambiguous specification for program synthesis [21]: the set of synthesized programs for a demonstration can be very large. Joker shares the general idea of displaying synthesized programs, but only presents the top-ranked program.

More broadly, Joker's use of PBD to generate editable code embodies Ravi Chugh's notion of *prodirect manipulation* [11], implemented in Sketch-N-Sketch [12], which aims to bridge the divide between programmatic and direct manipulation.

6 CONCLUSION AND FUTURE WORK

In this paper, we presented a unified interaction model for web customization. Our key idea is a spreadsheet formula language that encompasses both extraction and augmentation tasks, along with a programming-by-demonstration (PBD) interface that makes it easy for end-users to program to create formulas. The main area of future work involves making the formula language more accessible to end-users not familiar with CSS selectors. Our ultimate goal is to enable anyone that uses the web to customize websites in the course of their daily use in an intuitive and flexible way.

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] Tim Furche, Jinsong Guo, Sebastian Maneth, and Christian Schallhart. [n.d.]. Robust and Noise Resistant Wrapper Induction. In *Proceedings of the 2016 International Conference on Management of Data* (New York, NY, USA, 2016-06-14) *(SIGMOD ’16)*. Association for Computing Machinery, 773–784. https://doi.org/10.1145/2882903.2915214

[14] William R Harris and Sumit Gulwani. [n.d.]. Spreadsheet Table Transformations from Examples. ([n. d.]), 12.

[15] 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

[16] 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

[17] 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

[18] 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

[19] 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

[20] Mikaël Mayer, Gustavo Soares, Maxim Grechkin, Vu Le, Mark Marron, Oleksandr 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

[21] 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

[22] Frank M. Shipman and Catherine C. Marshall. [n.d.]. Formality Considered Harmful: Experiences, Emerging Themes, and Directions on the Use of Formal Representations in Interactive Systems. 8, 4 ([n. d.]), 333–352. https://doi.org/10.1023/A:1008716330212