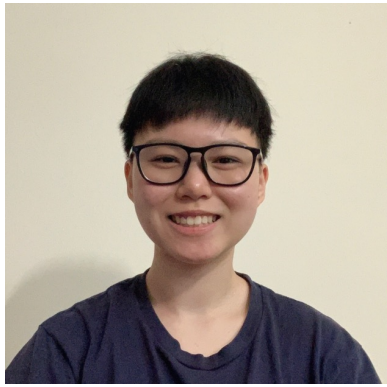
WebRobot

PLDI San Diego 2022

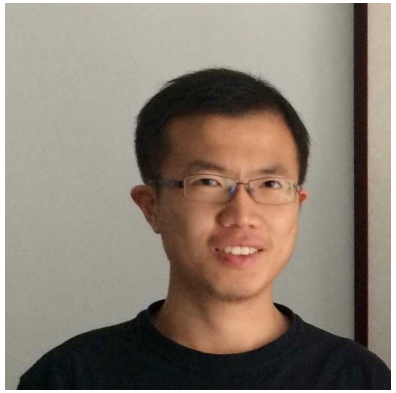
An Interactive System for Web Automation using Programming by Demonstration

Rui Dong[1]

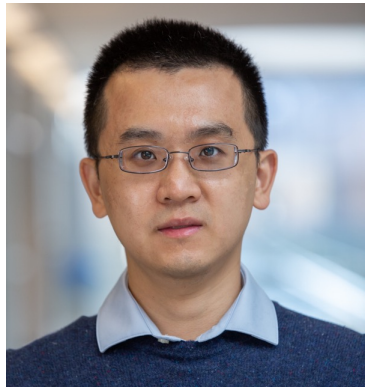
Zhicheng Huang[1]



Ian Iong Lam[1]



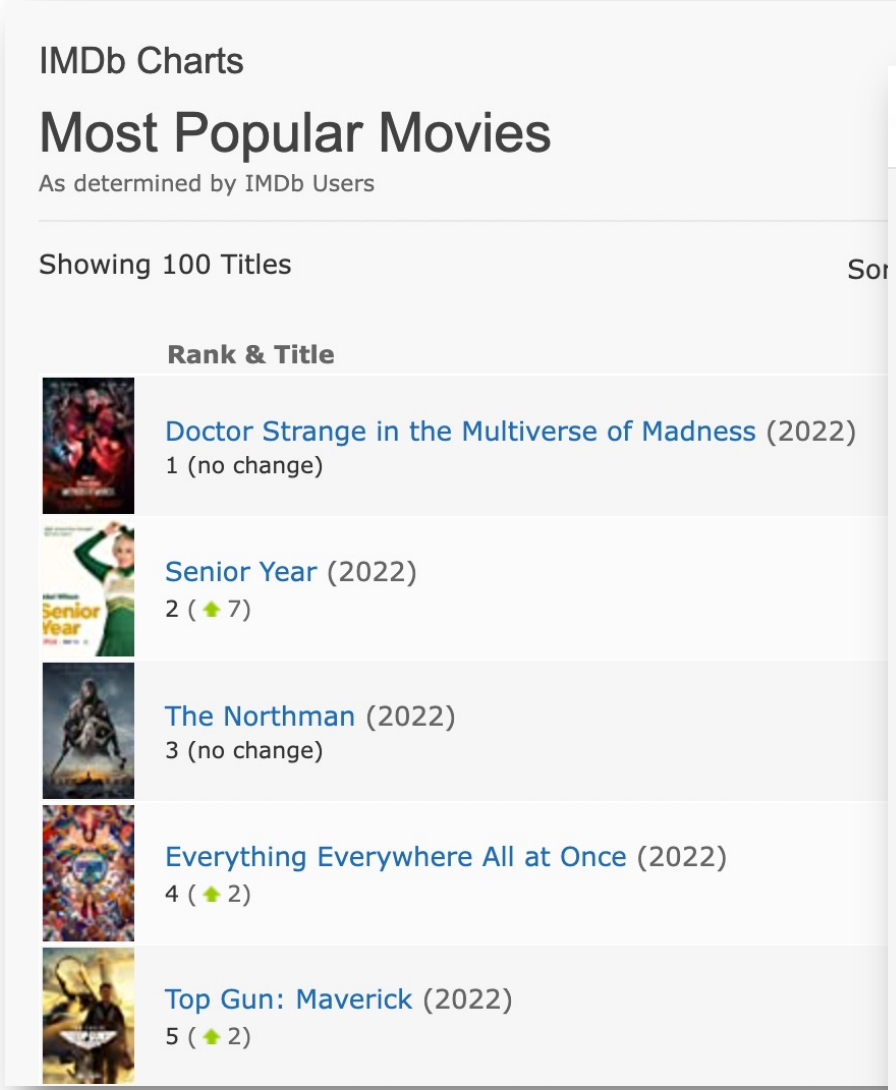
Yan Chen[2]

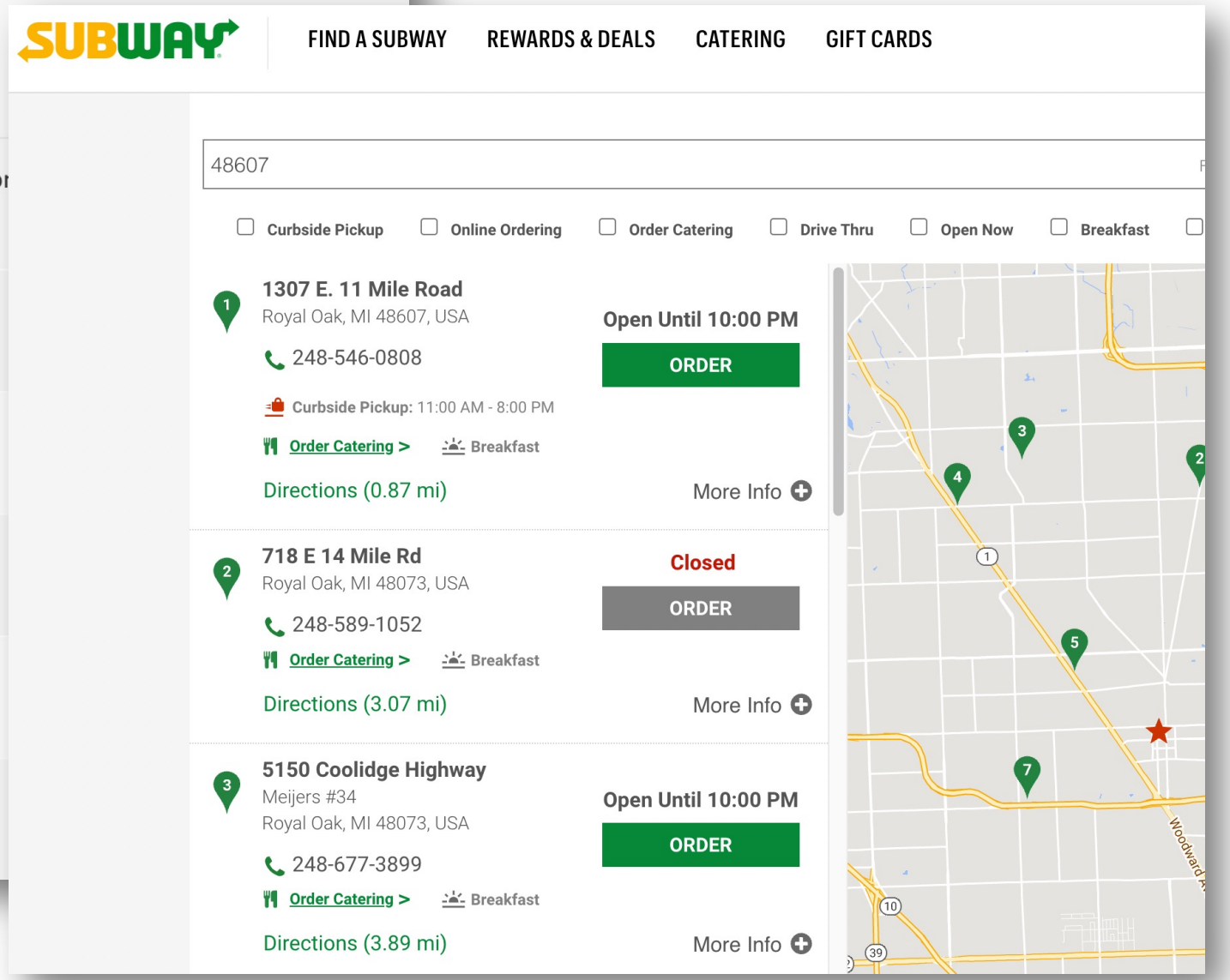


Xinyu Wang[1]

University of Michigan[1]

University of Toronto[2]





[1] https://forum.imacros.net/viewtopic.php?f=7&t=929
[2] https://forum.imacros.net/viewtopic.php?f=7&t=21028

Key challenges

```python
result = []
for zipcode in inputs:
    search_box = driver.find_element(By.XPATH,
                                     '//div[@class="searchBoxWrapper"]/input')
    search_box.send_keys(zipcode)
    driver.find_element(By.XPATH,
                        '//button[@class="squareButton buttonLarge btnDoSearch"]').click()
    for page_idx in range(5):
        store_list = driver.find_elements(By.XPATH,
                                          '//div[@class="locationList"]/div')
        for store in store_list:
            address_line_1 = store.find_element(By.CLASS_NAME,
                                                "storeMainAddress").text
            address_line_2 = store.find_element(By.CLASS_NAME,
                                                "locatorAddressCityState").text
            phone_number = store.find_element(By.CLASS_NAME,
                                              "locatorPhone").text
            result.append((address_line_1, address_line_2, phone_number))
    next_page_btn = driver.find_element(By.XPATH,
                                        '//div[@class="sprite-next-page-arrow"]')
    next_page_btn.click()
```

[1] R. Krosnick and S. Oney, "Understanding the Challenges and Needs of Programmers Writing Web Automation Scripts," *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2021, pp. 1-9, doi: 10.1109/VL/HCC51201.2021.9576476.

```
result = []
for zipcode in inputs:
    search_box = driver.find_element(By.XPATH,
                                     '//div[@class="searchBoxWrapper"]/input')
    search_box.send_keys(zipcode)
    driver.find_element(By.XPATH,
                        '//button[@class="squareButton buttonLarge btnDoSearch"]').click()
    for page_idx in range(5):
        store_list = driver.find_elements(By.XPATH,
                                          '//div[@class="locationList"]/div')
        for store in store_list:
            address_line_1 = store.find_element(By.CLASS_NAME,
                                                "storeMainAddress").text
            address_line_2 = store.find_element(By.CLASS_NAME,
                                                "locatorAddressCityState").text
            phone_number = store.find_element(By.CLASS_NAME,
                                              "locatorPhone").text
            result.append((address_line_1, address_line_2, phone_number))
    next_page_btn = driver.find_element(By.XPATH,
                                        '//div[@class="sprite-next-page-arrow"]')
    next_page_btn.click()
```

Key challenges

1. Complex structure

[1] R. Krosnick and S. Oney, "Understanding the Challenges and Needs of Programmers Writing Web Automation Scripts," *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2021, pp. 1-9, doi: 10.1109/VL/HCC51201.2021.9576476.

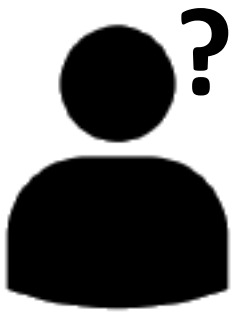
```
result = []
for zipcode in inputs:
    search_box = driver.find_element(By.XPATH,
                                     '//div[@class="searchBoxWrapper"]/input')
    search_box.send_keys(zipcode)
    driver.find_element(By.XPATH,
                        '//button[@class="squareButton buttonLarge btnDoSearch"]').click()
    for page_idx in range(5):
        store_list = driver.find_elements(By.XPATH,
                                          '//div[@class="locationList"]/div')
        for store in store_list:
            address_line_1 = store.find_element(By.CLASS_NAME,
                                                "storeMainAddress").text
            address_line_2 = store.find_element(By.CLASS_NAME,
                                                "locatorAddressCityState").text
            phone_number = store.find_element(By.CLASS_NAME,
                                              "locatorPhone").text
            result.append((address_line_1, address_line_2, phone_number))
    next_page_btn = driver.find_element(By.XPATH,
                                        '//div[@class="sprite-next-page-arrow"]')
    next_page_btn.click()
```

Key challenges

1. Complex structure

2. Generalizable selectors

[1] R. Krosnick and S. Oney, "Understanding the Challenges and Needs of Programmers Writing Web Automation Scripts," *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2021, pp. 1-9, doi: 10.1109/VL/HCC51201.2021.9576476.



Can someone write scripts for me?

Our idea:

Interactive Programming by Demonstration

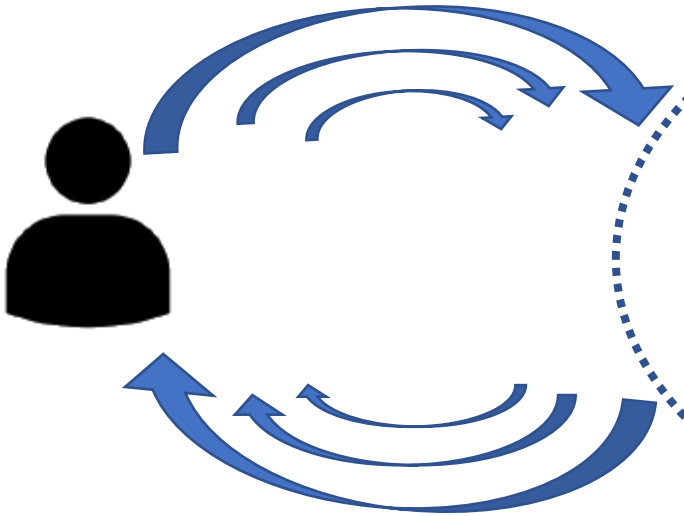
WebRobot

a trace of actions

Scrape */html/…/div[1]//h3*
Scrape */html/…/div[1]//div[@class="citystate"]*
Scrape */html/…/div[1]//div[@class="phone"]*
Scrape */html/…/div[2]//h3*
Scrape */html/…/div[2]//div[@class="citystate"]*
Scrape */html/…/div[2]//div[@class="phone"]*

Key idea: interactive programming-by-demonstration

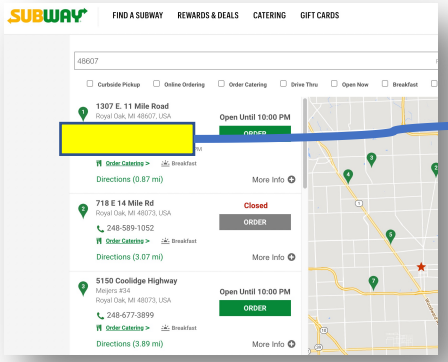
Demonstrate actions



Synthesizer

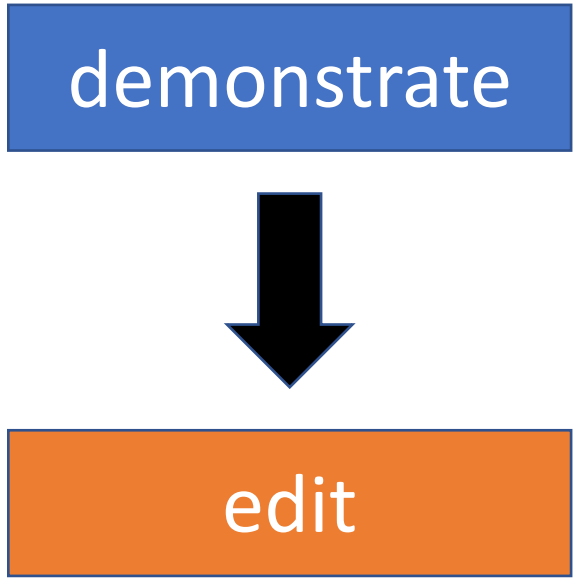
automate

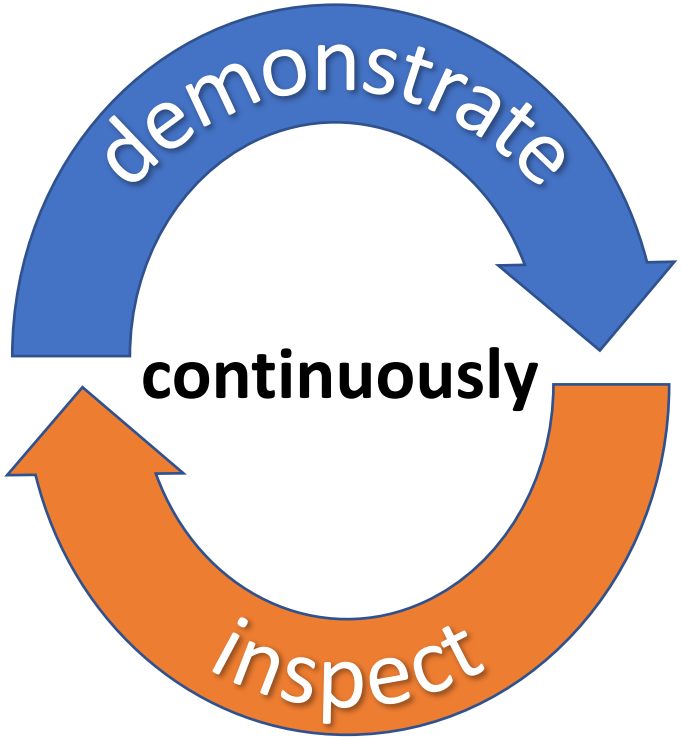
Visualize and inspect



Scrape this?

Our system vs previous systems

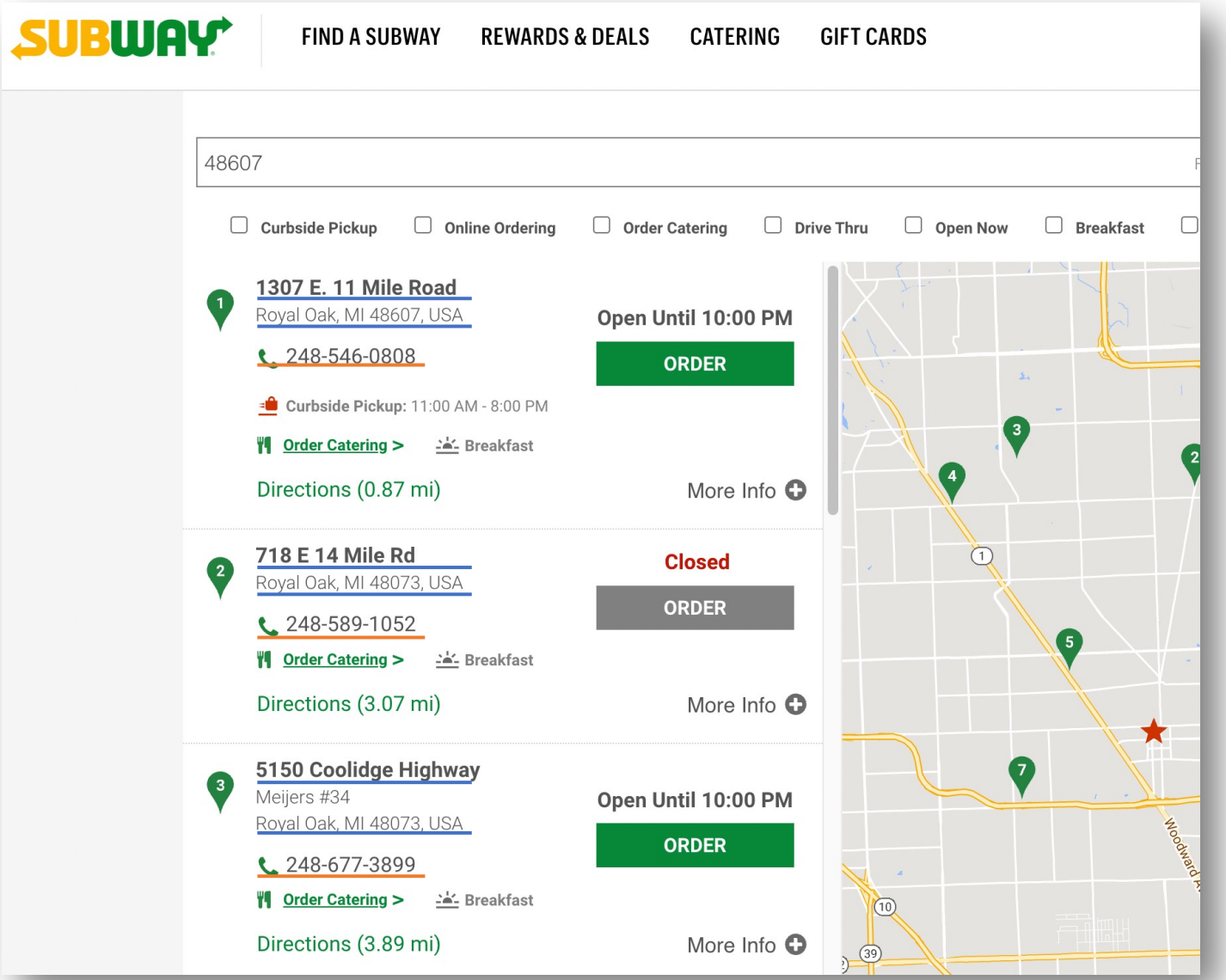




Previous works

Our system

Example: scraping subway store information



Task: extract each store's **address** and **phone number**

There are **10** stores on each page, **5** pages in total

[1] https://forum.imacros.net/viewtopic.php?f=7&t=21028

Synthesis input

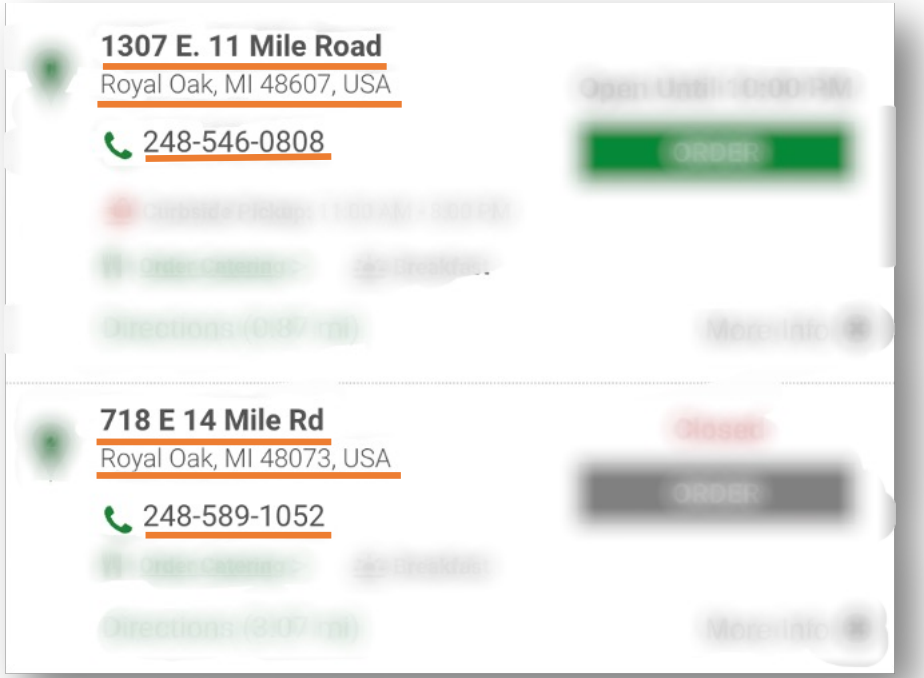
Extract first two stores

Scrape */html/…/div[1]//h3*
Scrape */html/…/div[1]//div[@class="citystate"]*
Scrape */html/…/div[1]//div[@class="phone"]*
Scrape */html/…/div[2]//h3*
Scrape */html/…/div[2]//div[@class="citystate"]*
Scrape */html/…/div[2]//div[@class="phone"]*

Action trace A

Action type

Selector(XPath)



Synthesis problem

Action trace A

Scrape */html/…/div[1]//h3*
Scrape */html/…/div[1]//div[@class="citystate"]*
Scrape */html/…/div[1]//div[@class="phone"]*
Scrape */html/…/div[2]//h3*
Scrape */html/…/div[2]//div[@class="citystate"]*
Scrape */html/…/div[2]//div[@class="phone"]*

synthesize → P

Such P should satisfy some requirements:

- Reproduce A (consistent with what users did)
- Generalize A (will produce more actions)

Synthesis problem

Action trace A

Scrape /html/…/div[1]//h3
Scrape /html/…/div[1]//div[@class="citystate"]
Scrape /html/…/div[1]//div[@class="phone"]
Scrape /html/…/div[2]//h3
Scrape /html/…/div[2]//div[@class="citystate"]
Scrape /html/…/div[2]//div[@class="phone"]

synthesize → P → execute

Action trace A'

Scrape /html/…/div[1]//h3
Scrape /html/…/div[1]//div[@class="citystate"]
Scrape /html/…/div[1]//div[@class="phone"]
Scrape /html/…/div[2]//h3
Scrape /html/…/div[2]//div[@class="citystate"]
Scrape /html/…/div[2]//div[@class="phone"]
Scrape /html/…/div[3]//h3
Scrape /html/…/div[3]//div[@class="citystate"]
Scrape /html/…/div[3]//div[@class="phone"]

Such P should satisfy some requirements:

- Reproduce A (consistent with what users did)
- Generalize A (will produce more actions)

Synthesis problem

Action trace A

Scrape */html/…/div[1]//h3*
Scrape */html/…/div[1]//div[@class="citystate"]*
Scrape */html/…/div[1]//div[@class="phone"]*
Scrape */html/…/div[2]//h3*
Scrape */html/…/div[2]//div[@class="citystate"]*
Scrape */html/…/div[2]//div[@class="phone"]*

synthesize → P → execute

Action trace A'

Scrape */html/…/div[1]//h3*
Scrape */html/…/div[1]//div[@class="citystate"]*
Scrape */html/…/div[1]//div[@class="phone"]*
Scrape */html/…/div[2]//h3*
Scrape */html/…/div[2]//div[@class="citystate"]*
Scrape */html/…/div[2]//div[@class="phone"]*
Scrape */html/…/div[3]//h3*
Scrape */html/…/div[3]//div[@class="citystate"]*
Scrape */html/…/div[3]//div[@class="phone"]*

Such P should satisfy some requirements:

- Reproduce A (consistent with what users did)
- Generalize A (will produce more actions)

Synthesis problem

Action trace A

Scrape */html/…/div[1]//h3*
Scrape */html/…/div[1]//div[@class="citystate"]*
Scrape */html/…/div[1]//div[@class="phone"]*
Scrape */html/…/div[2]//h3*
Scrape */html/…/div[2]//div[@class="citystate"]*
Scrape */html/…/div[2]//div[@class="phone"]*

synthesize → P → execute

Action trace A'

Scrape */html/…/div[1]//h3*
Scrape */html/…/div[1]//div[@class="citystate"]*
Scrape */html/…/div[1]//div[@class="phone"]*
Scrape */html/…/div[2]//h3*
Scrape */html/…/div[2]//div[@class="citystate"]*
Scrape */html/…/div[2]//div[@class="phone"]*
Scrape */html/…/div[3]//h3*
Scrape */html/…/div[3]//div[@class="citystate"]*
Scrape */html/…/div[3]//div[@class="phone"]*

Such P should satisfy some requirements:

- Reproduce A (consistent with what users did)
- Generalize A (will produce more actions)

Search based synthesis doesn’t work

Search based Synthesis

- Search in grammar
- Widely used for Programming by Example

Our grammar is too large

Our idea: leverage action trace

Action trace A

Scrape */html/…/div[1]//h3*
Scrape */html/…/div[1]//div[@class="citystate"]*
Scrape */html/…/div[1]//div[@class="phone"]*
Scrape */html/…/div[1]//h3*
Scrape */html/…/div[1]//div[@class="citystate"]*
Scrape */html/…/div[1]//div[@class="phone"]*

Leverage this to guide synthesis!

Rewrite-based synthesis!

Existing rewrite-based synthesis

egg: e-graphs good

State-of-the-art rewrite library using E-graph [POPL 2021]

Their key idea: sound rewrite rules

Sound rewrite rules

Sound rewrite rules

$$(x * y)/z = x * (y/z)$$

$$x/x = 1$$

$$x * 1 = x$$

$$(a * 2)/2$$

match

rewrite

$$a * (2/2)$$

match

rewrite

$$a * 1$$

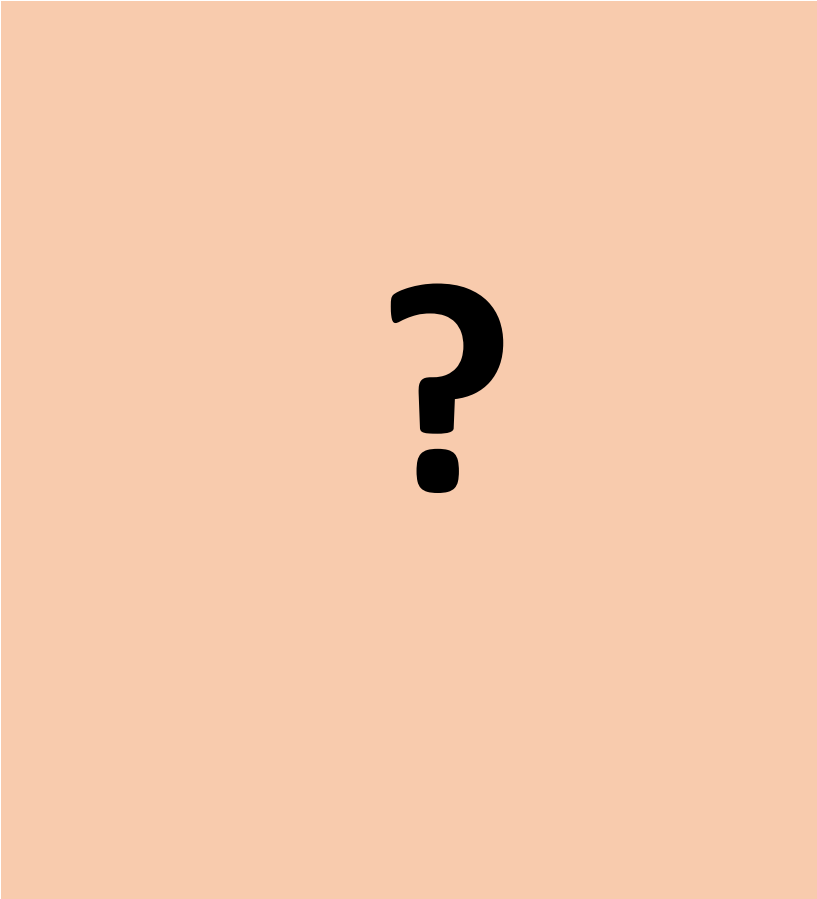
match

rewrite

$$a$$

It’s challenging to write sound rules in our domain!

Rewrite rules



Scrape */html/…/div[1]//h3*
Scrape */html/…/div[1]//div[@class=“citystate”]*
Scrape */html/…/div[1]//div[@class=“phone”]*
…
Scrape */html/…/div[10]//h3*
Scrape */html/…/div[10]//div[@class=“citystate”]*
Scrape */html/…/div[10]//div[@class=“phone”]*
Click //html/…/span
Scrape */html/…/div[1]//h3*
Scrape */html/…/div[1]//div[@class=“citystate”]*
Scrape */html/…/div[1]//div[@class=“phone”]*
…
Scrape */html/…/div[10]//h3*
Scrape */html/…/div[10]//div[@class=“citystate”]*
Scrape */html/…/div[10]//div[@class=“phone”]*
Click //html/…/span

A possible way to rewrite using sound rules

Scrape */html/…/div[1]//h3*
Scrape */html/…/div[1]//div[@class="citystate"]*
Scrape */html/…/div[1]//div[@class="phone"]*
…
Scrape */html/…/div[10]//h3*
Scrape */html/…/div[10]//div[@class="citystate"]*
Scrape */html/…/div[10]//div[@class="phone"]*
Click //html/…/span
Scrape */html/…/div[1]//h3*
Scrape */html/…/div[1]//div[@class="citystate"]*
Scrape */html/…/div[1]//div[@class="phone"]*
…
Scrape */html/…/div[10]//h3*
Scrape */html/…/div[10]//div[@class="citystate"]*
Scrape */html/…/div[10]//div[@class="phone"]*
Click //html/…/span

split
set boundaries

Scrape */html/…/div[1]//h3*
Scrape */html/…/div[1]//div[@class="citystate"]*
Scrape */html/…/div[1]//div[@class="phone"]*
…
Scrape */html/…/div[10]//h3*
Scrape */html/…/div[10]//div[@class="citystate"]*
Scrape */html/…/div[10]//div[@class="phone"]*

Click //html/…/span

Scrape */html/…/div[1]//h3*
Scrape */html/…/div[1]//div[@class="citystate"]*
Scrape */html/…/div[1]//div[@class="phone"]*
…
Scrape */html/…/div[10]//h3*
Scrape */html/…/div[10]//div[@class="citystate"]*
Scrape */html/…/div[10]//div[@class="phone"]*

Click //html/…/span

A possible way to rewrite using sound rules

Scrape */html/…/div[1]//h3*
Scrape */html/…/div[1]//div[@class="citystate"]*
Scrape */html/…/div[1]//div[@class="phone"]*
…
Scrape */html/…/div[10]//h3*
Scrape */html/…/div[10]//div[@class="citystate"]*
Scrape */html/…/div[10]//div[@class="phone"]*

Click //html/…/span

Scrape */html/…/div[1]//h3*
Scrape */html/…/div[1]//div[@class="citystate"]*
Scrape */html/…/div[1]//div[@class="phone"]*
…
Scrape */html/…/div[10]//h3*
Scrape */html/…/div[10]//div[@class="citystate"]*
Scrape */html/…/div[10]//div[@class="phone"]*

Click //html/…/span

rewrite rules

rewrite rules

For loop 1
Click //html/…/span
For loop 2
Click //html/…/span

Split and rewrite

While ..
For loop
Click /html/…/span

Using existing techniques is not practical

- Hard to write **sound rewrite rules**
- Rule matching is slow for a **complex trace**

Traditional rewrite-based synthesis doesn't work well for our domain

Our idea: specs and rewrite rules

The idea: sound rewrite rules + validation

Our idea: speculation and validation

rewrite rules + **sound**

⬇ ⬇

unsound rewrite rules
guess the rewrite locally

validate afterwards

Speculation **Validation**

Our idea: speculation and validation

sound
rewrite rules

guess
+
check

Speculation

Key idea: Guess using local and simple patterns

Scrape /html/div[1]/div[1]
Scrape /html/div[1]/div[1]//h3
Scrape /html/div[1]/div[2]
Scrape /html/div[1]/div[2]//h3
Scrape /html/div[1]/div[3]
Scrape /html/div[1]/div[3]//h3
Scrape /html/div[2]/div[1]
Scrape /html/div[2]/div[1]//h3
Scrape /html/div[2]/div[2]
Scrape /html/div[2]/div[2]//h3

Speculation

Key idea: Guess using local and simple patterns

Scrape /html/div[1]/div[1]
Scrape /html/div[1]/div[1]//h3
Scrape /html/div[1]/div[2]
Scrape /html/div[1]/div[2]//h3
Scrape /html/div[1]/div[3]
Scrape /html/div[1]/div[3]//h3
Scrape /html/div[2]/div[1]
Scrape /html/div[2]/div[1]//h3
Scrape /html/div[2]/div[2]
Scrape /html/div[2]/div[2]//h3

Speculation

Key idea: Guess using local and simple patterns

Scrape /html/div[1]/div[1]
Scrape /html/div[1]/div[1]//h3
Scrape /html/div[1]/div[2]
Scrape /html/div[1]/div[2]//h3
Scrape /html/div[1]/div[3]
Scrape /html/div[1]/div[3]//h3
Scrape /html/div[2]/div[1]
Scrape /html/div[2]/div[1]//h3
Scrape /html/div[2]/div[2]
Scrape /html/div[2]/div[2]//h3

Guess: these two actions map to two iterations of the same action in a loop body

guess

```
Foreach v in
Children(/html/div[1],div):
   Scrape v
   ....
```

Speculation

Key idea: Guess using local and simple patterns

Scrape /html/div[1]/div[1]
Scrape /html/div[1]/div[1]//h3
Scrape /html/div[1]/div[2]
Scrape /html/div[1]/div[2]//h3
Scrape /html/div[1]/div[3]
Scrape /html/div[1]/div[3]//h3
Scrape /html/div[2]/div[1]
Scrape /html/div[2]/div[1]//h3
Scrape /html/div[2]/div[2]
Scrape /html/div[2]/div[2]//h3

Guess: these two actions map to two iterations of the same action in a loop body

guess

Foreach v in Children(/html/div[1],div):
  Scrape v
  ….

Foreach v in Children(/html/div[1],div):
  Scrape v
  Scrape v//h3

Speculation

Advantages of the speculation:

- ✓ rewrite rules are easy to write
- ✓ rewrite rules are easy to match

BUT! Our speculation is unsound

Foreach v in Children(/html/div[1],div):

Scrape v

Scrape v//h3

Rewrite candidate

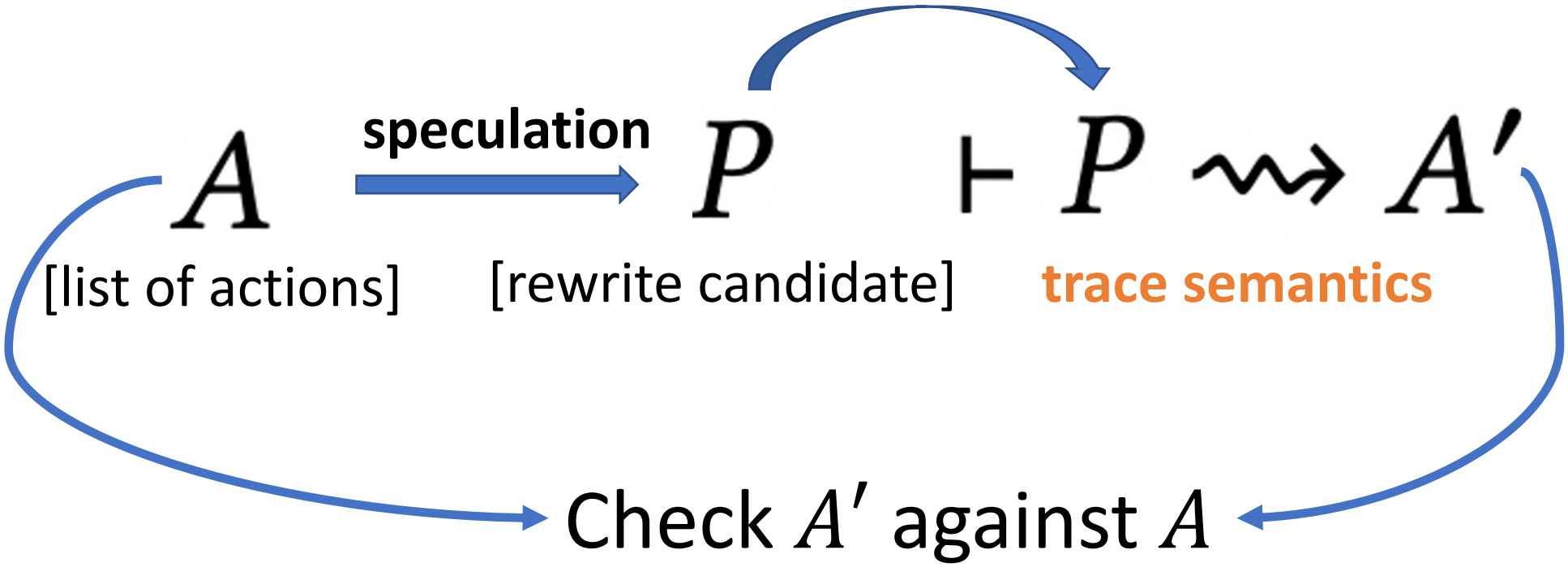
Questions

- Is this guess correct?
- Which part can it rewrite?

We **validate**!

Validation

Key idea: Validate candidate rewrites using trace semantics



Validation

Key idea: Validate candidate rewrites using trace semantics

Foreach v in Children(/html/div[1],div):
Scrape v
Scrape v//h3

evaluate →

A'

Scrape /html/div[1]/div[1]
Scrape /html/div[1]/div[1]//h3
Scrape /html/div[1]/div[2]
Scrape /html/div[1]/div[2]//h3
Scrape /html/div[1]/div[3]
Scrape /html/div[1]/div[3]//h3

check against A →

Scrape /html/div[1]/div[1]
Scrape /html/div[1]/div[1]//h3
Scrape /html/div[1]/div[2]
Scrape /html/div[1]/div[2]//h3
Scrape /html/div[1]/div[3]
Scrape /html/div[1]/div[3]//h3
Scrape /html/div[2]/div[1]
Scrape /html/div[2]/div[1]//h3
Scrape /html/div[2]/div[2]
Scrape /html/div[2]/div[2]//h3

Validation

Key idea: Validate candidate rewrites using trace semantics

```
Scrape /html/div[1]/div[1]
Scrape /html/div[1]/div[1]//h3
Scrape /html/div[1]/div[2]
Scrape /html/div[1]/div[2]//h3
Scrape /html/div[1]/div[3]
Scrape /html/div[1]/div[3]//h3
Scrape /html/div[2]/div[1]
Scrape /html/div[2]/div[1]//h3
Scrape /html/div[2]/div[2]
Scrape /html/div[2]/div[2]//h3
```

(true) rewrite

```
Foreach v in Children(/html/div[1],div):
    Scrape v
    Scrape v//h3
Scrape /html/div[2]/div[1]
Scrape /html/div[2]/div[1]//h3
Scrape /html/div[2]/div[2]
Scrape /html/div[2]/div[2]//h3
```

Validation

Our rewrite is sound after validation

sound rewrite rules

speculation
+
validation

Put it together

```
Scrape /html/div[1]/div[1]
Scrape /html/div[1]/div[1]//h3
Scrape /html/div[1]/div[2]
Scrape /html/div[1]/div[2]//h3
Scrape /html/div[1]/div[3]
Scrape /html/div[1]/div[3]//h3
Scrape /html/div[2]/div[1]
Scrape /html/div[2]/div[1]//h3
Scrape /html/div[2]/div[2]
Scrape /html/div[2]/div[2]//h3
```

```
Foreach v in
Children(/html/div[1],div):
   Scrape v
   Scrape v//h3
Scrape /html/div[2]/div[1]
Scrape /html/div[2]/div[1]//h3
Scrape /html/div[2]/div[2]
Scrape /html/div[2]/div[2]//h3
```

speculation
validation

```
Foreach v in Children(/html/div[1],div):
   Scrape v
   Scrape v//h3
Foreach v in Children(/html/div[2],div):
   Scrape v
   Scrape v//h3
```

speculation
validation

```
Foreach v1 in Children(/html,div):
   Foreach v2 in Children(v1,div)
      Scrape v2
      Scrape v2//h3
```

speculation
validation

Evaluation

76 tasks from iMacros Forum

- Selected tasks with a complete description
- Manually wrote python scripts

Required Features	Number of Problems
Data Scraping	76
Data entry	29
Navigation	60
Pagination	33

Results

WebRobot can synthesize intended programs

- for **69/76** tasks
- within **1** second
- with around **10** manually demonstrated actions

Loop Nesting level	Number of Tasks
1	31
2	32
>3	6

Complex structures

User study

8 participants

5 tasks

Result:

- All participants can solve each problem within **90**s and **10** manually demonstrated actions
- Feedbacks: “**quite decent**”, “**smooth**”

Comparison with existing synthesis techniques

egg: e-graphs good

State-of-the-art library for rewrite-based synthesizer

Baseline built with egg	WebRobot
takes 200 s to generate doubly nested for loop can't generate three level nested for loop within 5 mins timeout	solves all problems within 1 s

Summary

WebRobot proposed …

- a formal basis for Web Automation
- an interactive Program-by-Demonstration model
- a novel speculate-and-validate methodology

Thanks! *Rui Dong(ruidong@umich.edu)*

Rules using Egg

Example trace:

$$
\begin{array}{ll}
[& \\
a_1: & Click\ a[1]/b[1] \\
a_2: & GoBack \\
a_3: & Click\ a[1]/b[2] \\
a_4: & GoBack \\
a_5: & Click\ a[1]/b[3] \\
a_6: & GoBack \\
a_7: & Click\ a[2]/b[1] \\
a_8: & GoBack \\
a_9: & Click\ a[2]/b[2] \\
] &
\end{array}
$$

1. InsertSplitUnsplit

$$[a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9] \mapsto Unsplit(Split([a_1, a_2, ..., a_8, a_9], 6))$$

2. ActuallySplit

$$Split([a_1, a_2, ..., a_8, a_9], 6) \mapsto [[a_1, a_2, a_3, a_4, a_5, a_6], [a_7, a_8, a_9]]$$

3. LoopRerolling

$$[a_1, a_2, a_3, a_4, a_5, a_6] \mapsto [\ ForEach\ j \text{ from 1 until the end } \{\ Click\ a[1]/b[j];\ GoBack;\ \}\ ]$$

$$[a_7, a_8, a_9] \mapsto [\ ForEach\ j \text{ from 1 until the end } \{\ Click\ a[2]/b[j];\ GoBack;\ \}\ ]$$

4. ActuallyUnsplit

$$UnSplit([[Loop_1], [Loop_2]]) \mapsto [Loop_1, Loop_2]$$