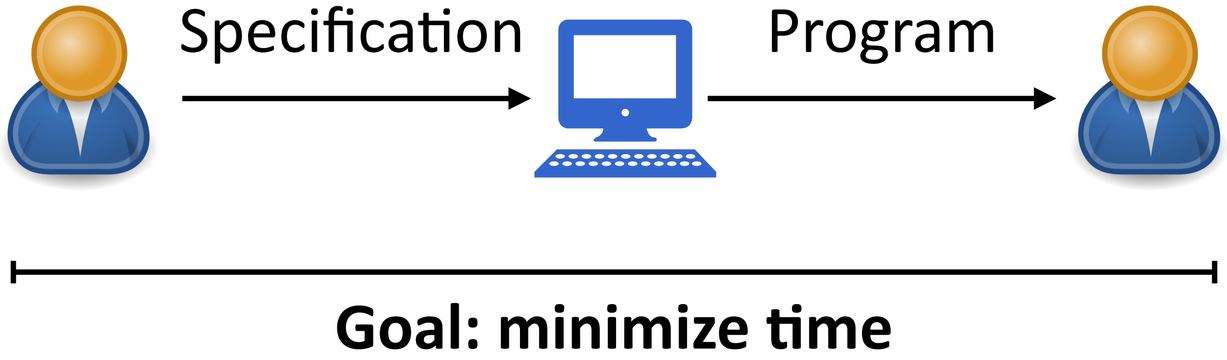
EECS 598-008 & EECS 498-008: Intelligent Programming Systems

Lecture 10

Announcements

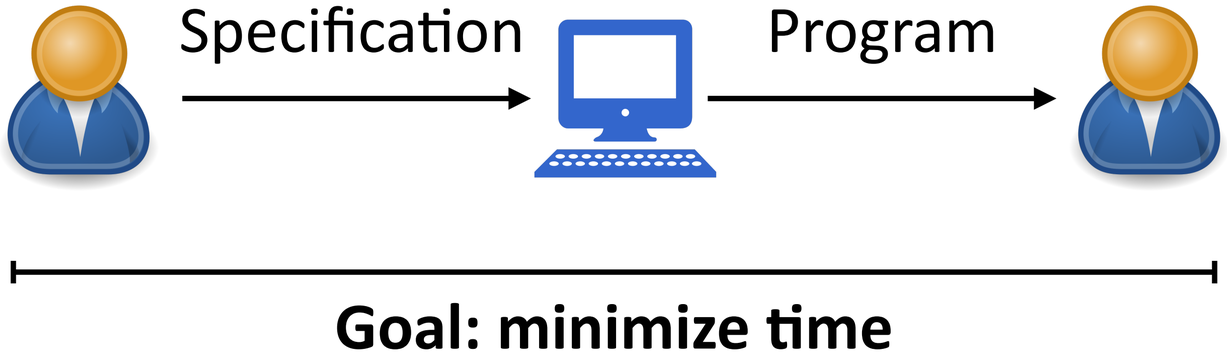
- **Live**, remote discussion 3-4pm Friday (tomorrow)
  - Zoom link on course website
  - Discuss A3
- Paper presentation assignment out by midnight today
  - 15 people submitted preferences
  - 12 slots

So Far…



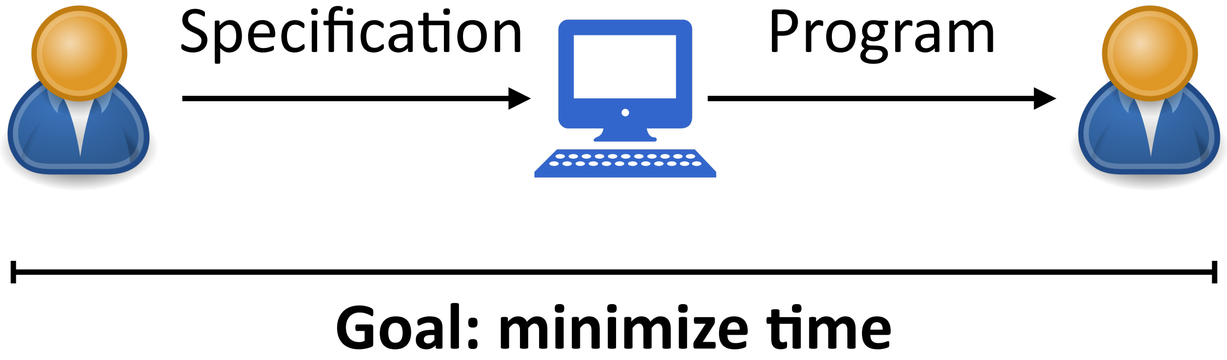
- Setup: given spec, find a program that satisfies spec

So Far…



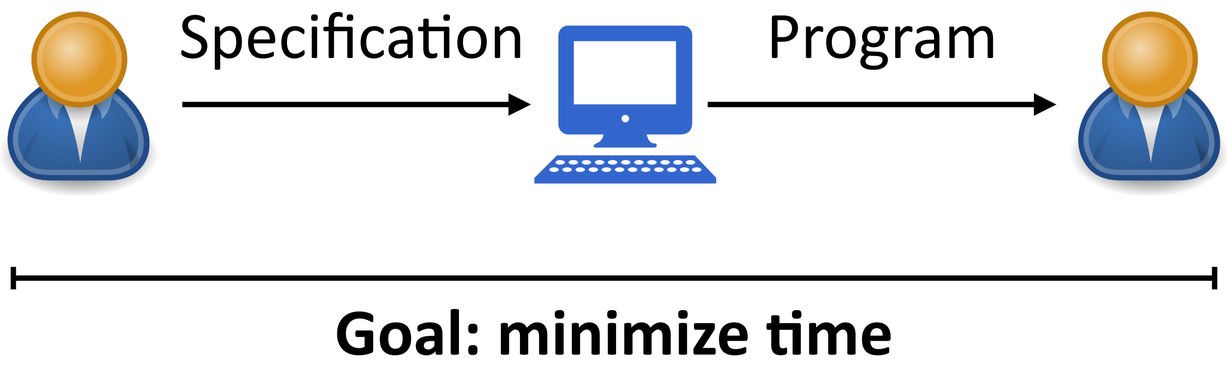
- Setup: given spec, find a program that satisfies spec
- Problems we've looked at:
  - Search space: how to define search space?
  - Search: how to find programs that satisfy specification?
  - Efficiency: How to efficiently search programs?

What's Wrong with This Setup?



- Setup: given spec, find a program that satisfies spec

What's Wrong with This Setup?



- Setup: given spec, find a program that satisfies spec
- Generalization: satisfy spec ≠ satisfy user intent
  - Because inductive specification is fundamentally ambiguous

Generalization

- Eventually, we want programs that satisfy user intent, not just examples

Generalization

- Eventually, we want programs that satisfy user intent, not just examples
- For some domains, one example suffices
  - E.g., table transformations for R

Generalization

- Eventually, we want programs that satisfy user intent, not just examples
- For some domains, one example suffices
  - E.g., table transformations for R
- For many other domains, need multiple examples
  - E.g., regular expressions

Generalization

- Eventually, we want programs that satisfy user intent, not just examples
- For some domains, one example suffices
  - E.g., table transformations for R
- For many other domains, need multiple examples
  - E.g., regular expressions
- Can we guarantee to satisfy user intent using inductive specifications?

Generalization

- Eventually, we want programs that satisfy user intent, not just examples
- For some domains, one example suffices
  - E.g., table transformations for R
- For many other domains, need multiple examples
  - E.g., regular expressions
- Can we guarantee to satisfy user intent using inductive specifications?
  - In theory, no, b/c need (potentially in general infinitely?) many examples

Generalization

- Eventually, we want programs that satisfy user intent, not just examples
- For some domains, one example suffices
  - E.g., table transformations for R
- For many other domains, need multiple examples
  - E.g., regular expressions
- Can we guarantee to satisfy user intent using inductive specifications?
  - In theory, no, b/c need (potentially in general infinitely?) many examples
  - In practice, yes, with right approach

Improving Generalization

- **<u>Ranking (inductive bias)</u>**
- Interaction
- Multi-modality

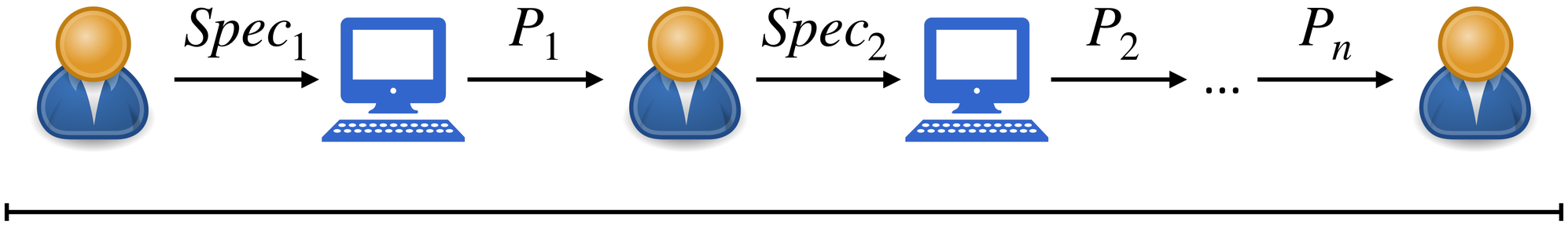
Inductive Bias / Ranking

- Covered in Lecture 7 (search prioritization)
  - Occam's razor (smallest program generalizes better)
  - Weighted search (explicit cost/ranking/scoring functions)
  - Statistical models (e.g., n-gram, neural nets)

Improving Generalization

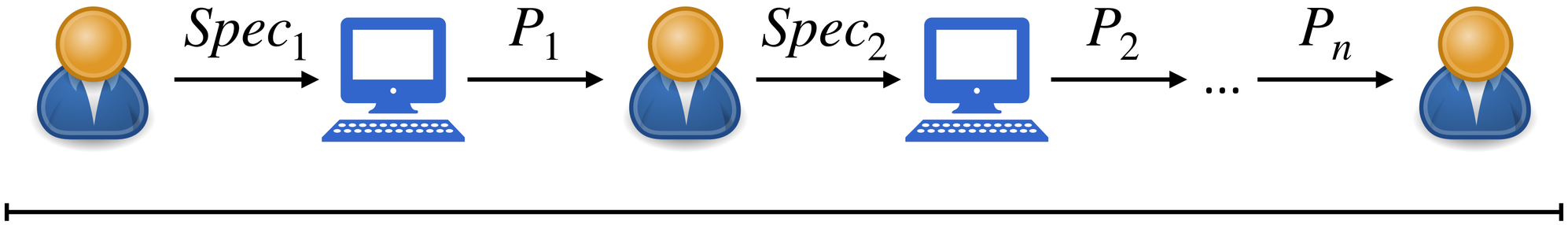
- Ranking (inductive bias)
- **<u>Interaction</u>**
- Multi-modality

Interactive Program Synthesis



Goal: minimize time and n

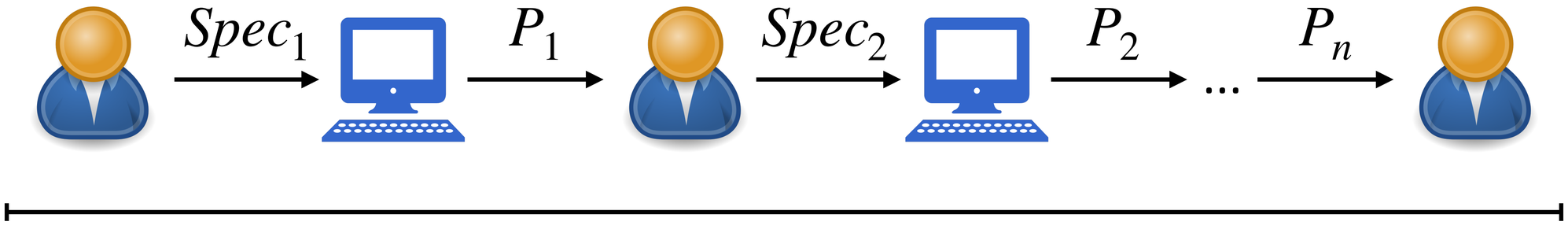
Interactive Program Synthesis



Goal: minimize time and n

- This new setup also introduces a few new problems..
  - How to pick "good" specifications?

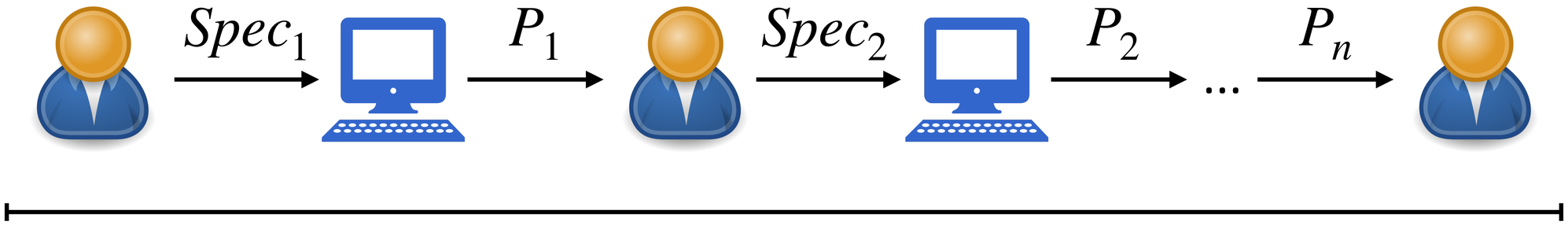
Interactive Program Synthesis



Goal: minimize time and n

- This new setup also introduces a few new problems..
  - How to pick "good" specifications?
  - How to explain each P_i to (non-expert) users?

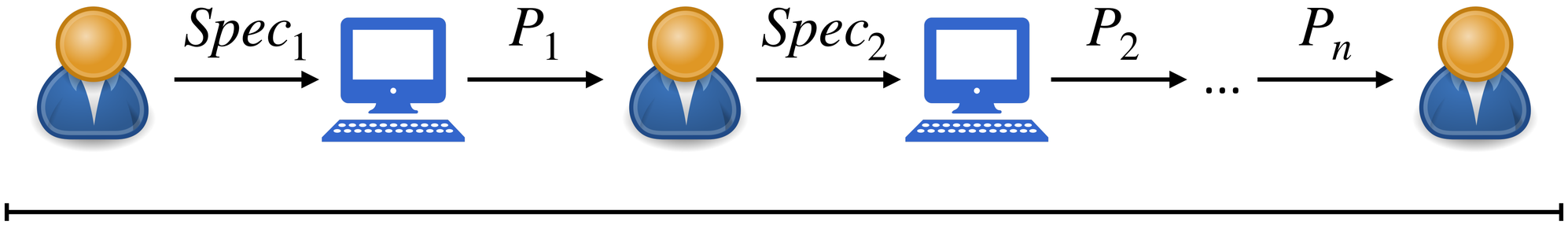
Interactive Program Synthesis



Goal: minimize time and n

- This new setup also introduces a few new problems..
  - How to pick "good" specifications?
  - How to explain each P_i to (non-expert) users?
  - How to know P_n is correct?

Interactive Program Synthesis



Goal: minimize time and n

- This new setup also introduces a few new problems..
  - How to pick "good" specifications?
  - How to explain each P_i to (non-expert) users?
  - How to know P_n is correct?
  - How to reuse past computation?
  - Etc.

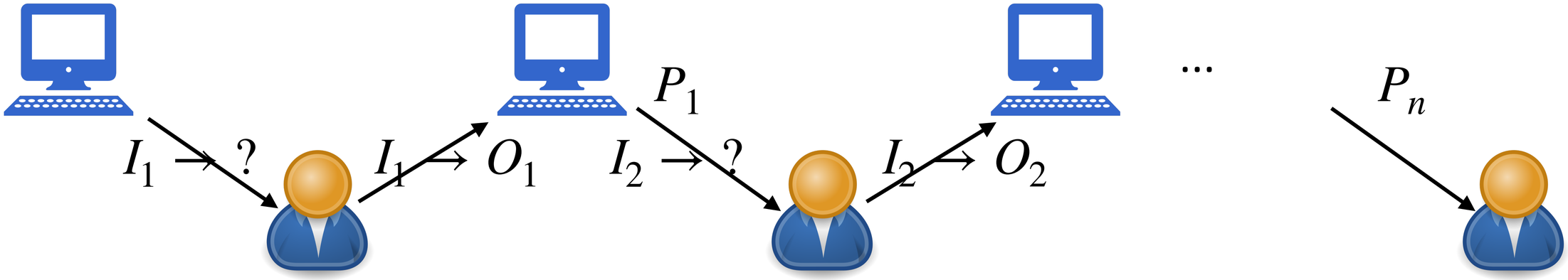
How To Pick “Good” Specifications?

How To Pick “Good” Specifications?

- Idea: let synthesizer **suggest** specifications!
  - A form of “active learning”

How To Pick "Good" Specifications?

- Idea: let synthesizer **suggest** specifications!
  - A form of "active learning"
- Active learning
  - Machine learning approach
  - Interact with users to prioritize data labeling process



Synthesizers Suggest Input, Users Label Output

- Related work
  - Oracle-guided component-based program synthesis (ICSE'10)
  - User Interaction Models for Disambiguation in Programming by Example (UIST'15)
  - Interactive Query Synthesis from Input-Output Examples (SIGMOD'17)
  - FlashProfile: a framework for synthesizing data profiles (OOPSLA'18)
  - Synthesis with Abstract Examples (CAV'21)
  - Question Selection for Interactive Program Synthesis (PLDI'21)

Synthesizers Suggest Input, Users Label Output

- Related work
  - Oracle-guided component-based program synthesis (ICSE'10)
  - User Interaction Models for Disambiguation in Programming by Example (UIST'15)
  - Interactive Query Synthesis from Input-Output Examples (SIGMOD'17)
  - FlashProfile: a framework for synthesizing data profiles (OOPSLA'18)
  - Synthesis with Abstract Examples (CAV'21)
  - Question Selection for Interactive Program Synthesis (PLDI'21)
- **Core problem: what input to suggest?**

FlashProg System

- FlashProg
  - PBE-based data extraction (from textual docs, spreadsheets, webpages)
  - Implemented as web application

User Interaction Models for Disambiguation in Programming by Example

Mikaël Mayer[*] mikael.mayer@epfl.ch
Gustavo Soares[†] gsoares@dsc.ufcg.edu.br
Maxim Grechkin[‡] grechkin@cs.washington.edu
Vu Le[§] vmle@ucdavis.edu
Mark Marron[¶] marron
Oleksandr Polozov[‡] polozov@cs.washington.edu
Rishabh Singh[¶] risin
Benjamin Zorn[¶] zorn
Sumit Gulwani[¶] sumitg

ABSTRACT
Programming by Examples (PBE) has the potential to revo-

sizing small programs in various domain-specific languages (DSLs) such as string and table transformations [8] and data

FlashProg System

- FlashProg
  - PBE-based data extraction (from textual docs, spreadsheets, webpages)
  - Implemented as web application
- Two features
  - Program navigation: visualize multiple synthesized programs
  - Conversational clarification: disambiguation

User Interaction Models for Disambiguation in Programming by Example

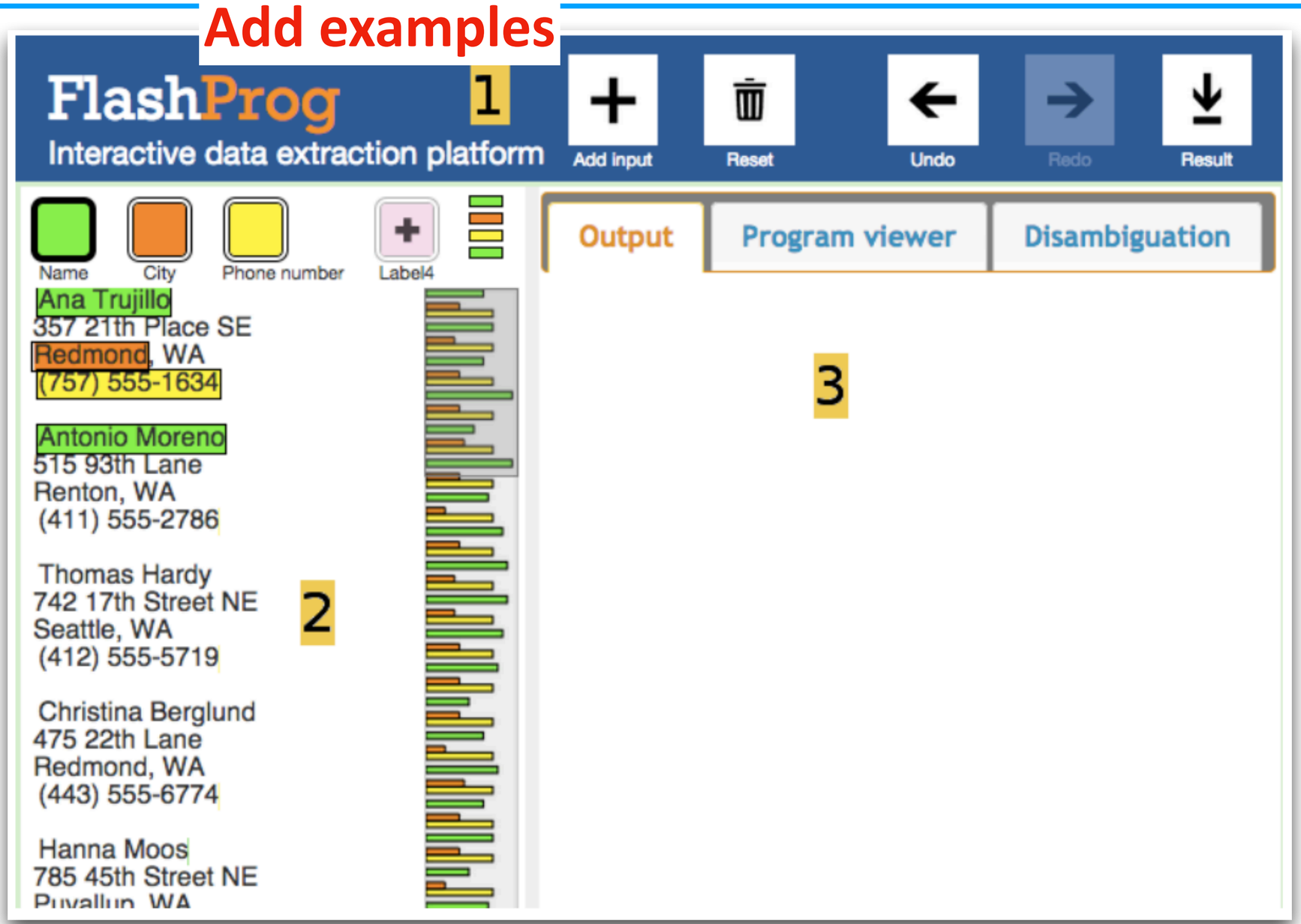
Mikaël Mayer[*] mikael.mayer@epfl.ch
Gustavo Soares[†] gsoares@dsc.ufcg.edu.br
Maxim Grechkin[‡] grechkin@cs.washington.edu
Vu Le[§] vmle@ucdavis.edu
Mark Marron[¶] marron
Oleksandr Polozov[‡] polozov@cs.washington.edu
Rishabh Singh[¶] risin
Benjamin Zorn[¶] zorn
Sumit Gulwani[¶] sumitg

ABSTRACT

Programming by Examples (PBE) has the potential to revo-

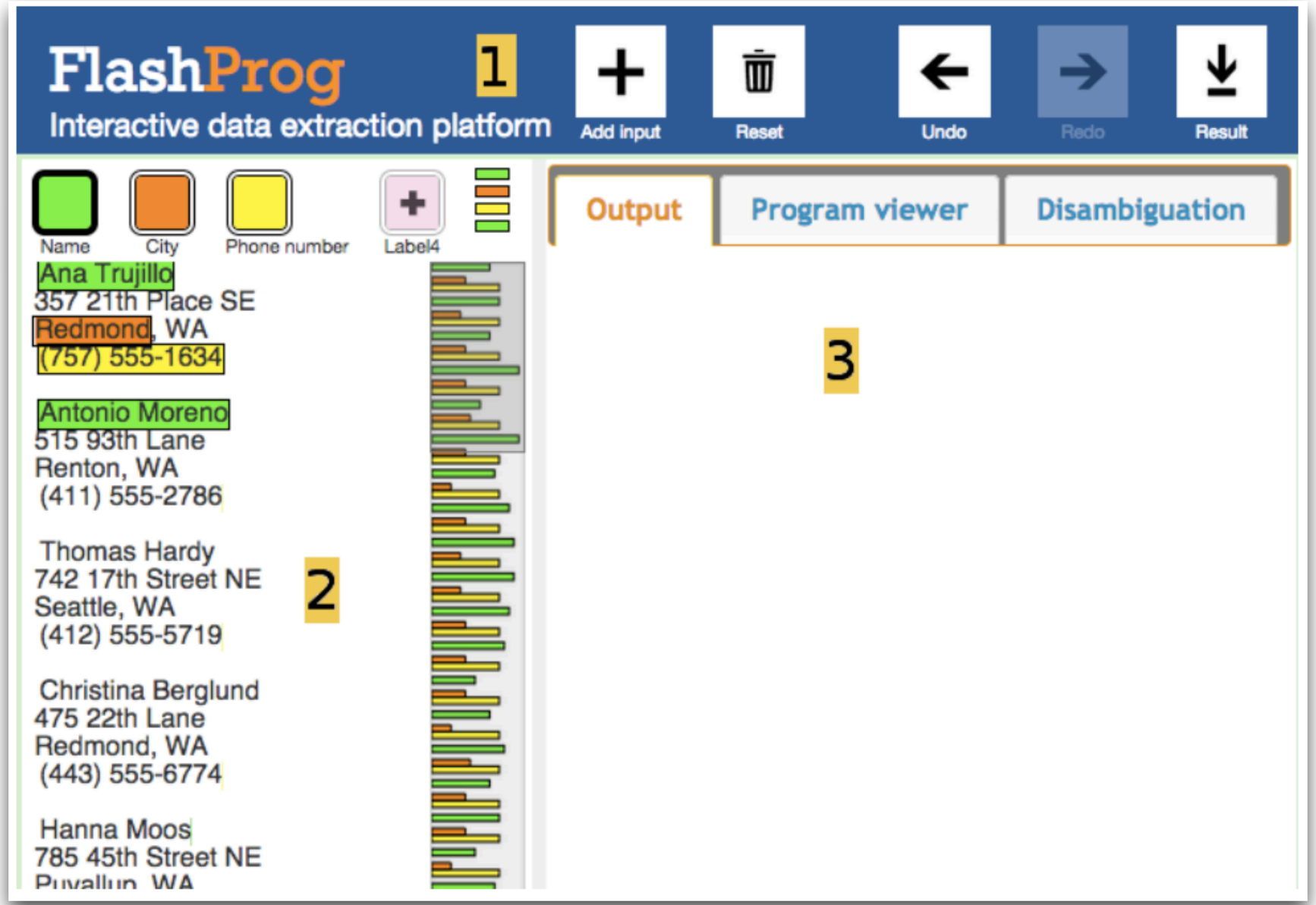
sizing small programs in various domain-specific languages (DSLs) such as string and table transformations [8] and data

FlashProg System

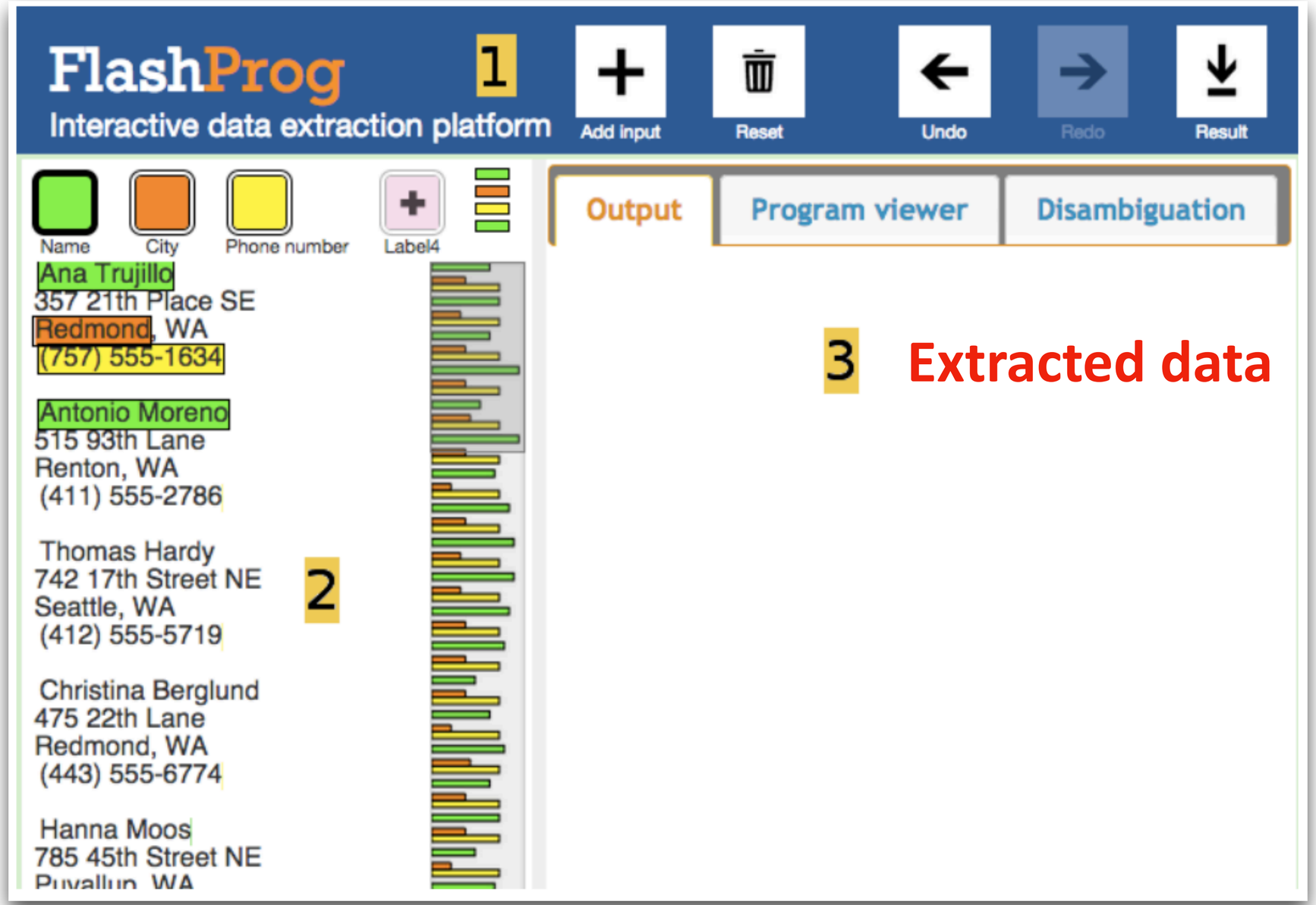


FlashProg System

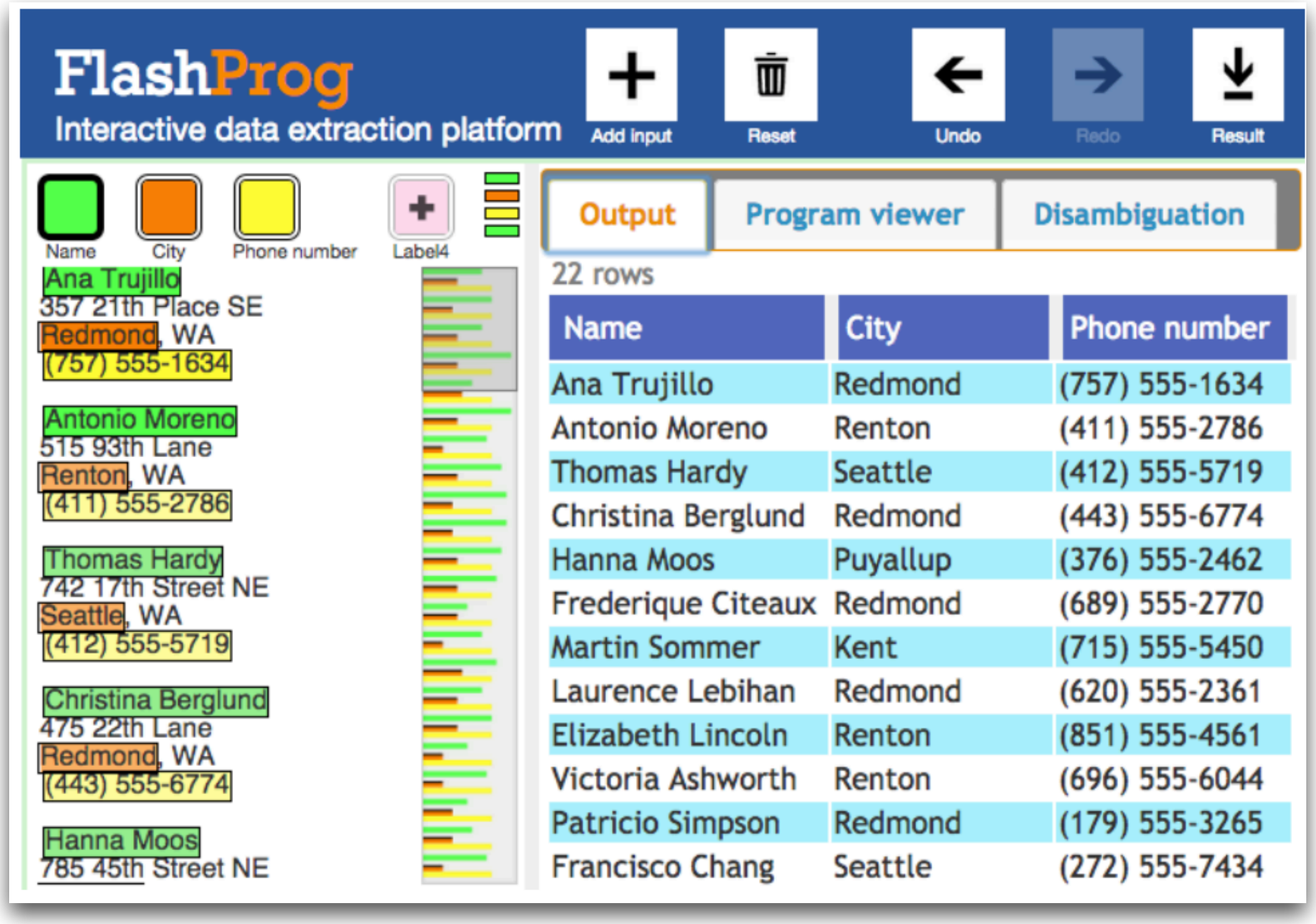
Input text



FlashProg System

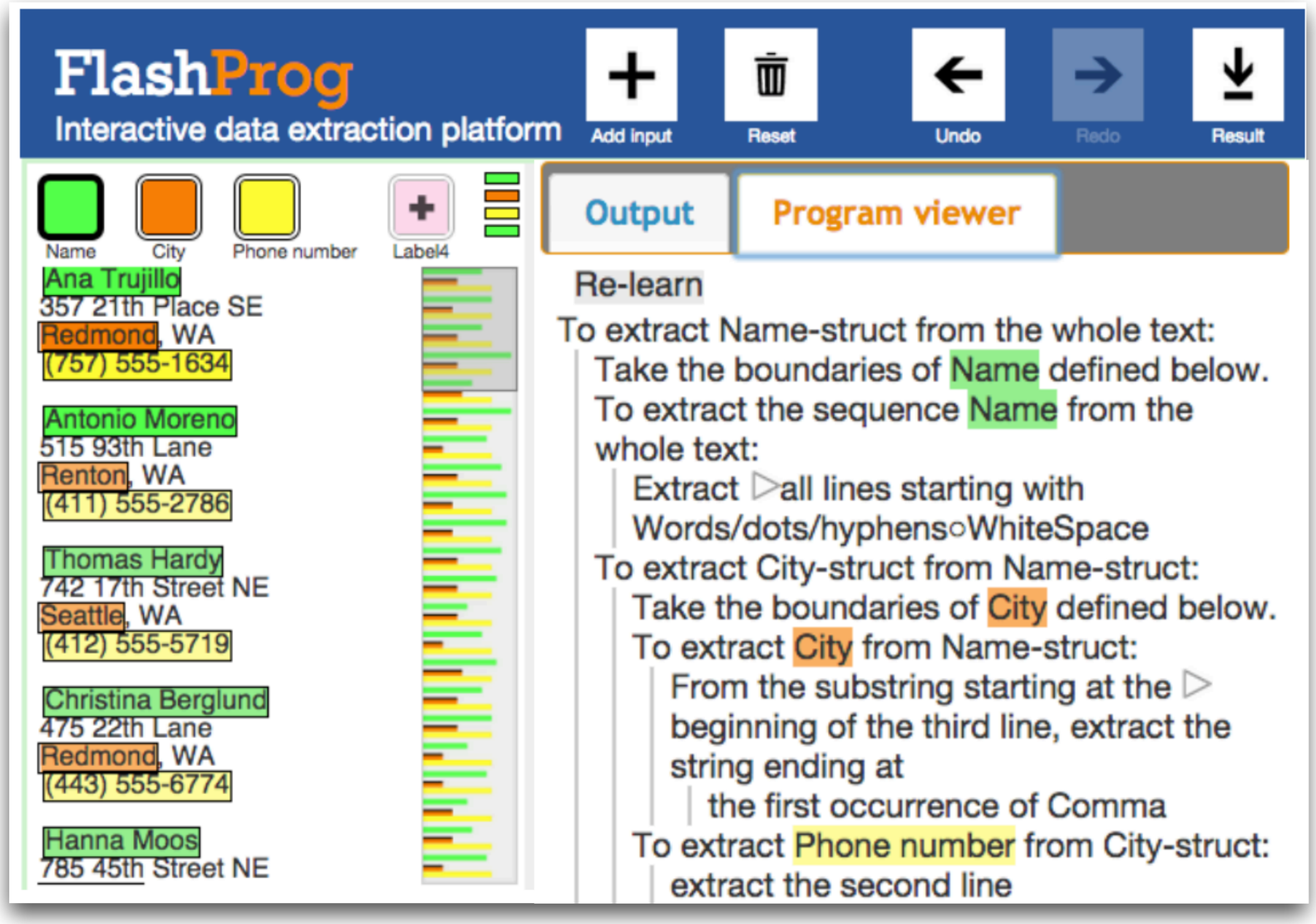


FlashProg System



Output Preview

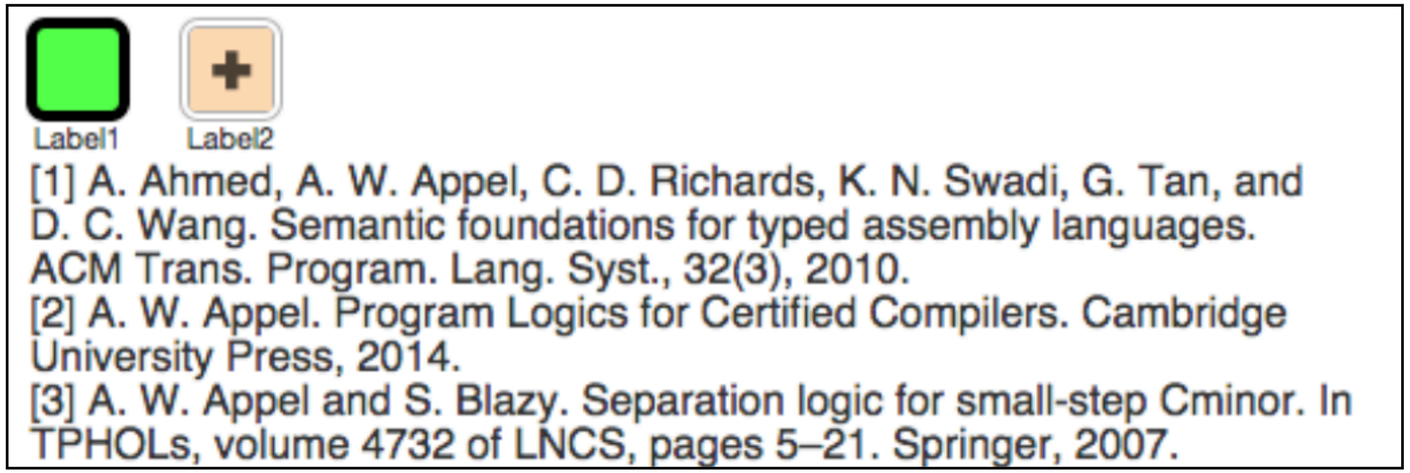
FlashProg System



Synthesized Program

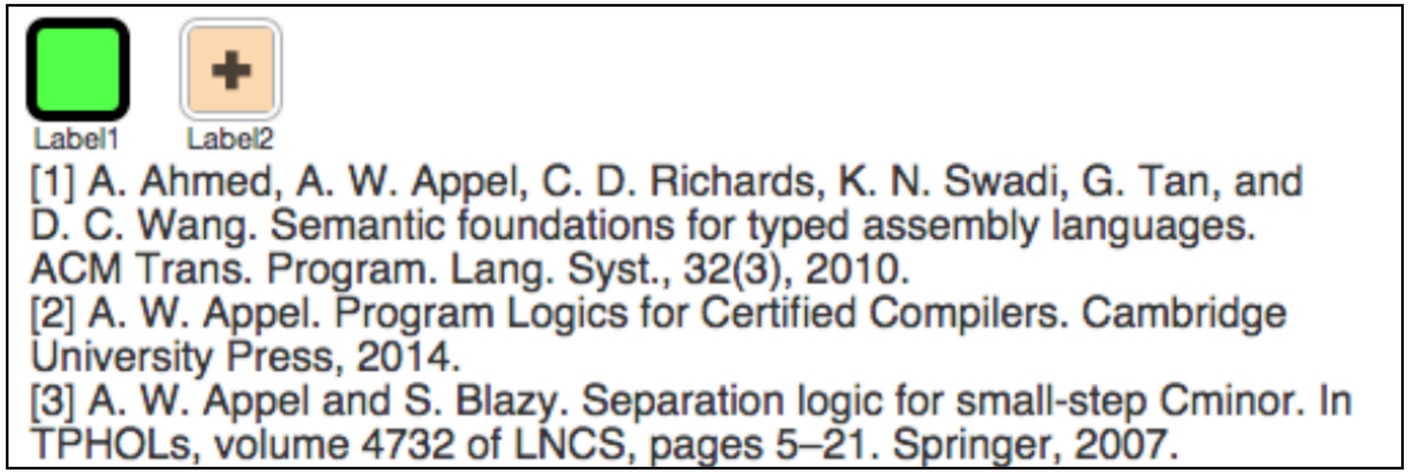
FlashProg System

- Consider task: given a list of papers, extract all authors



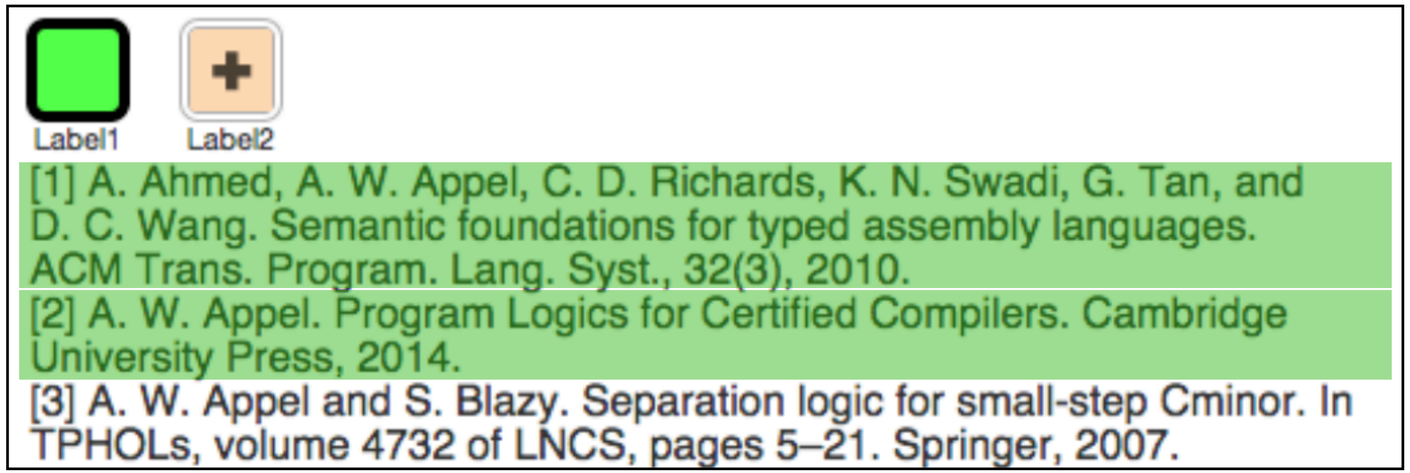
FlashProg System

- Consider task: given a list of papers, extract all authors
- FlashProg requires user to do this in steps



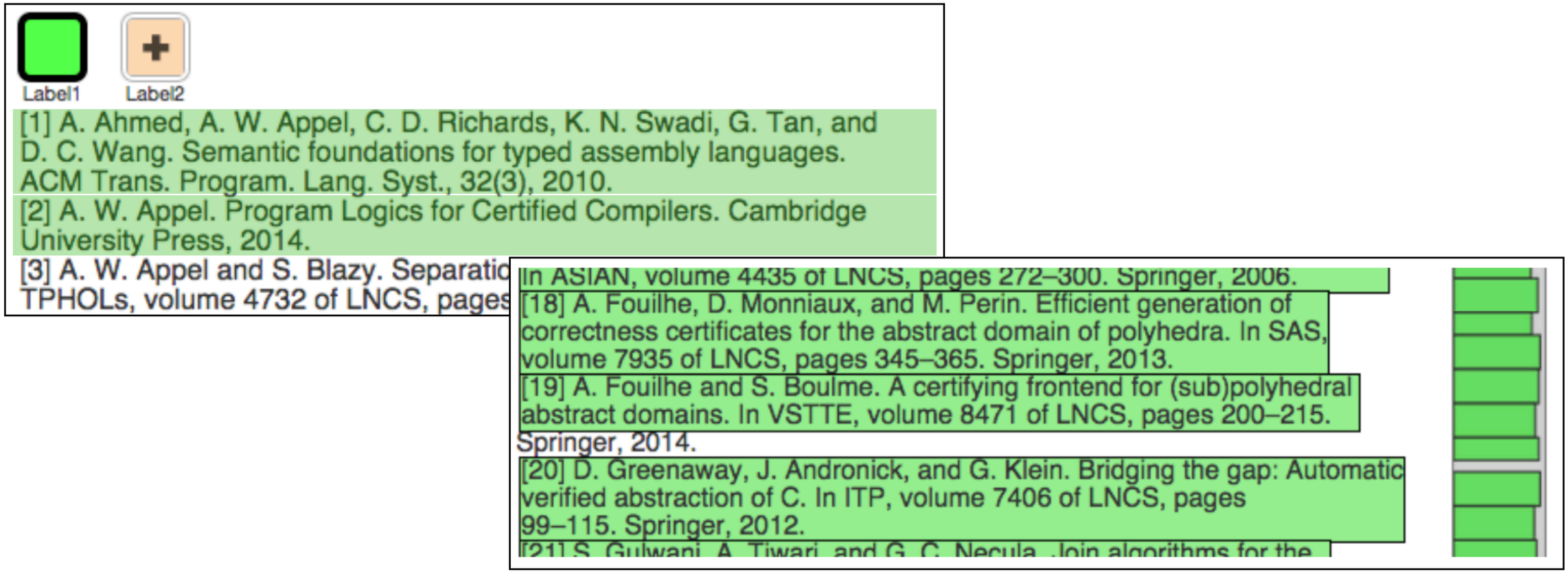
FlashProg System

- Consider task: given a list of papers, extract all authors
- FlashProg requires user to do this in steps
  - First, extract papers using examples



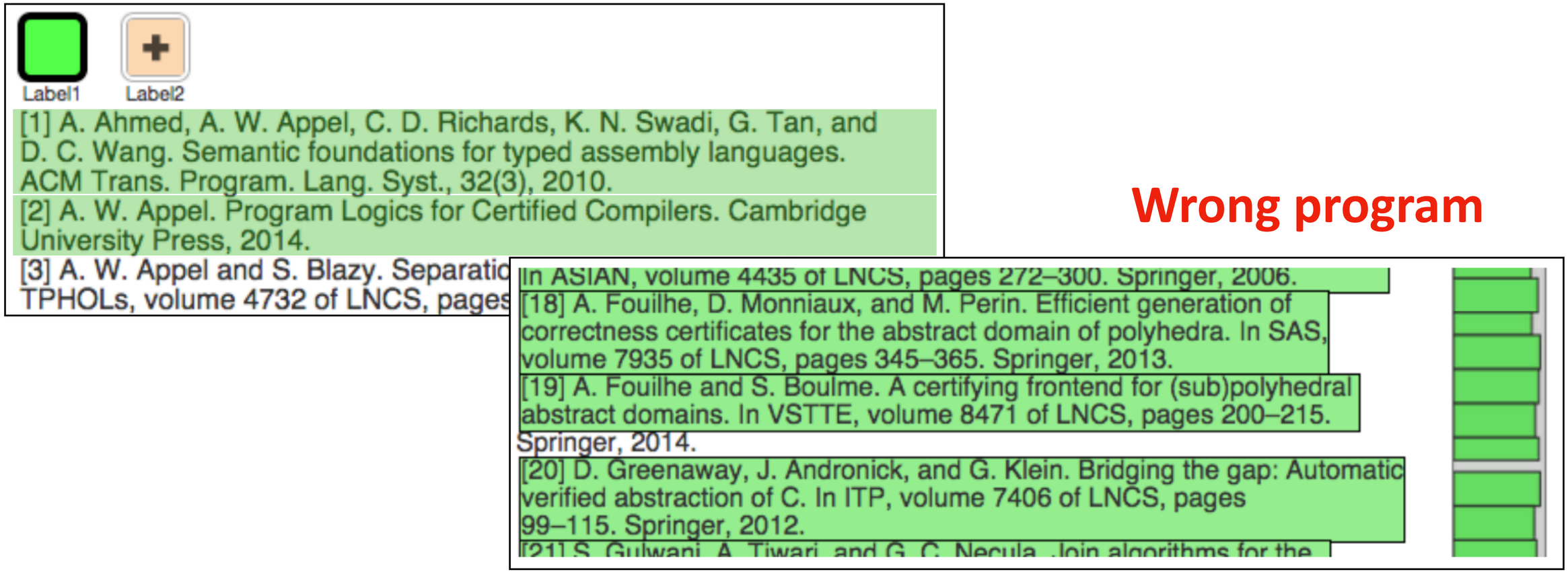
FlashProg System

- Consider task: given a list of papers, extract all authors
- FlashProg requires user to do this in steps
  - First, extract papers using examples



FlashProg System

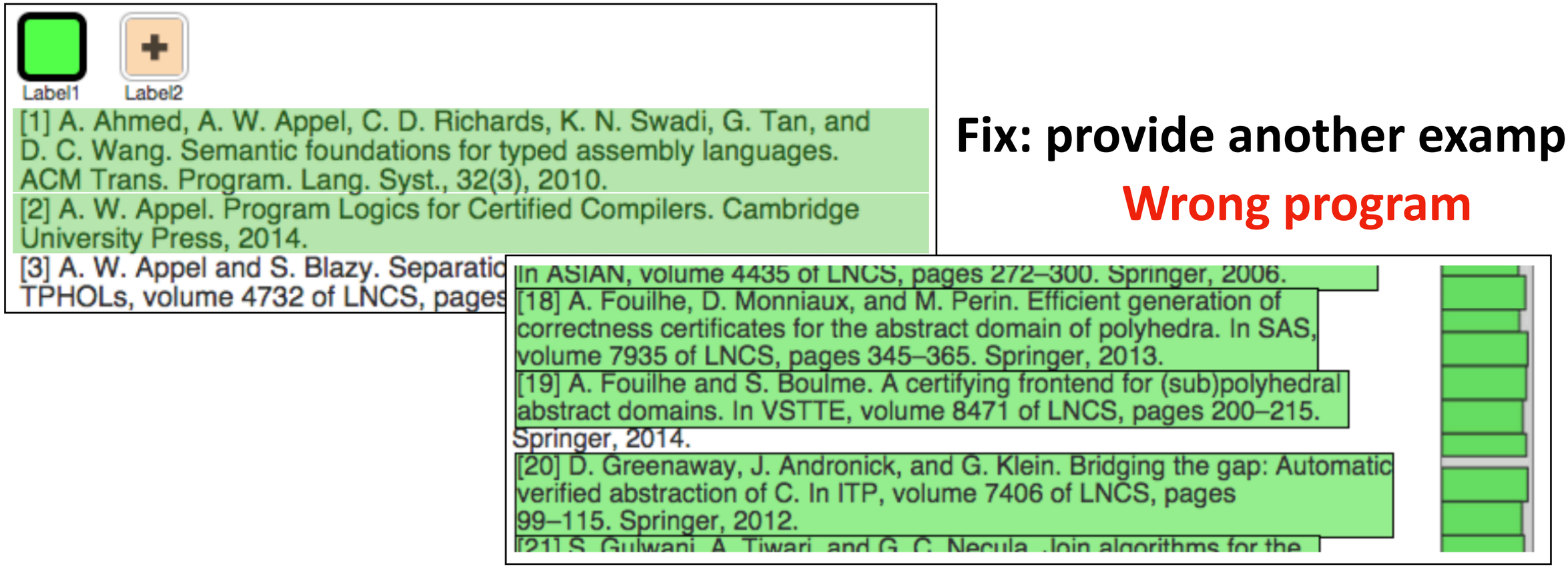
- Consider task: given a list of papers, extract all authors
- FlashProg requires user to do this in steps
  - First, extract papers using examples



Wrong program

FlashProg System

- Consider task: given a list of papers, extract all authors
- FlashProg requires user to do this in steps
  - First, extract papers using examples

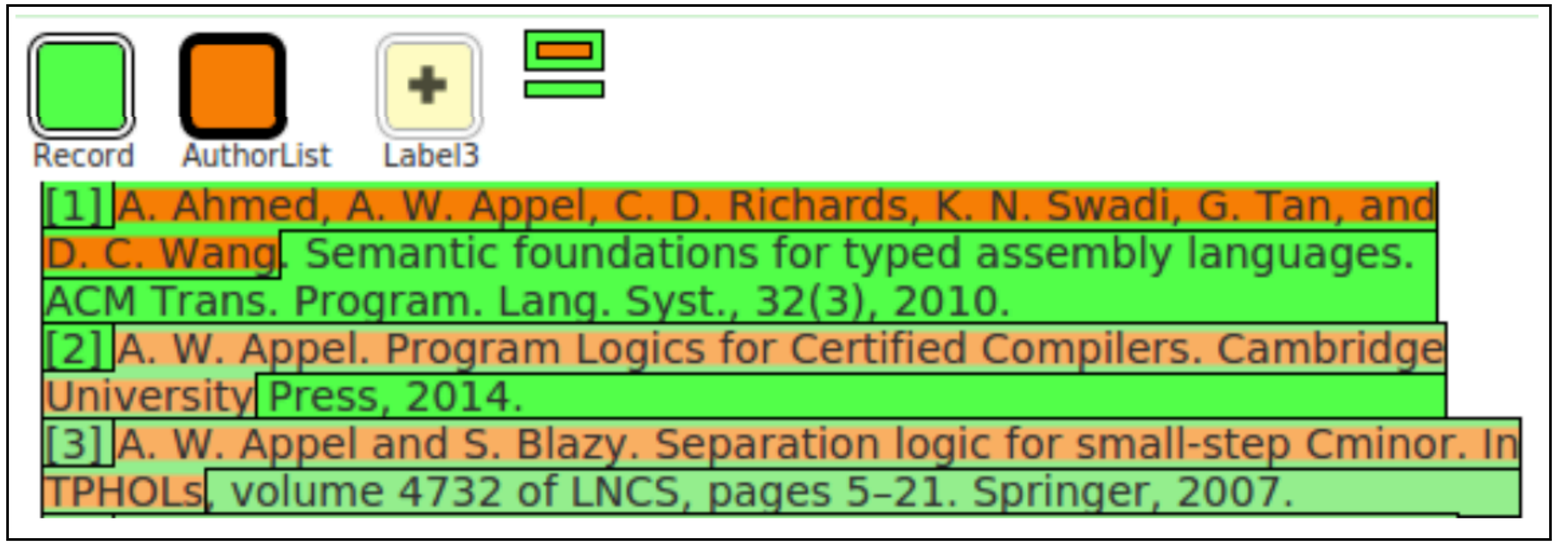


Fix: provide another example

Wrong program

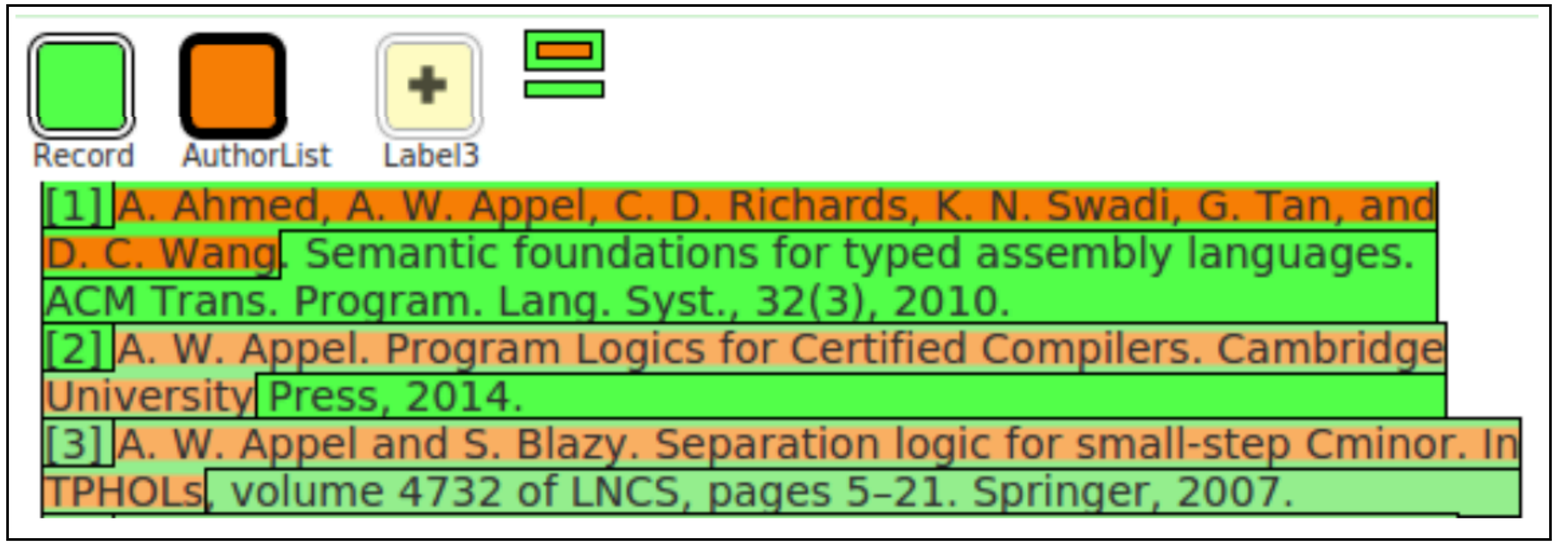
FlashProg System

- Consider task: given a list of papers, extract all authors
- FlashProg requires user to do this in steps
  - First, extract papers using examples
  - Next, extract author lists using examples



FlashProg System

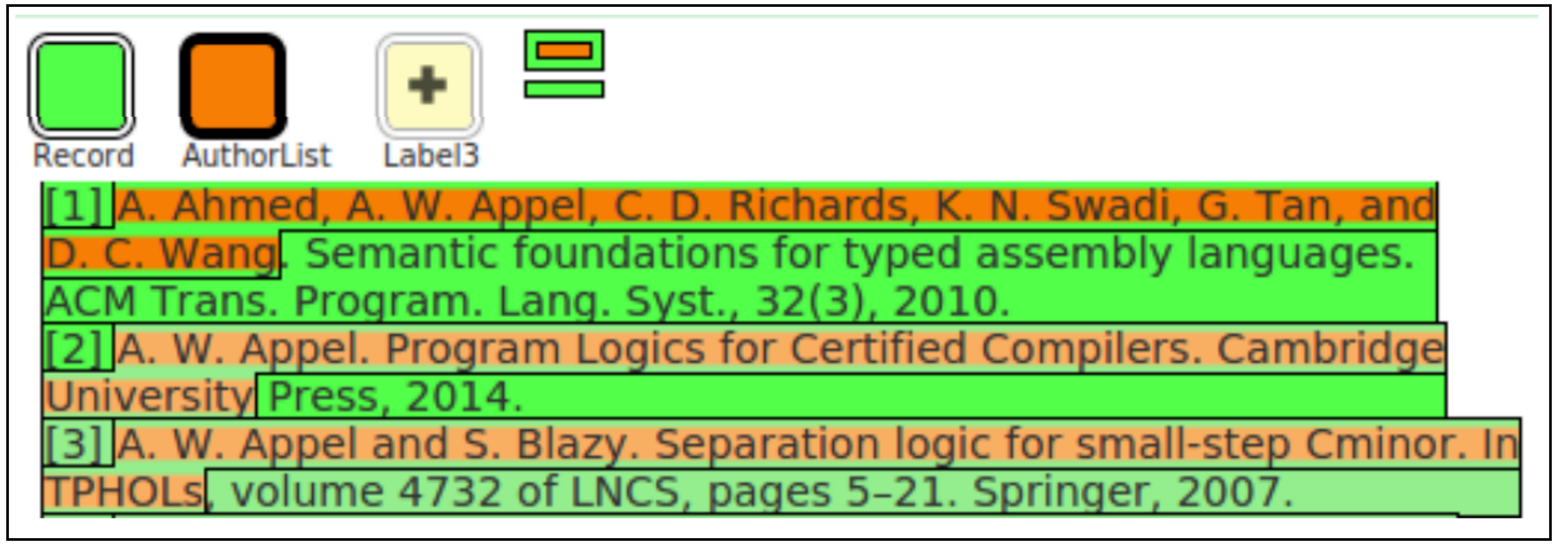
- Consider task: given a list of papers, extract all authors
- FlashProg requires user to do this in steps
  - First, extract papers using examples
  - Next, extract author lists using examples



Wrong program

FlashProg System

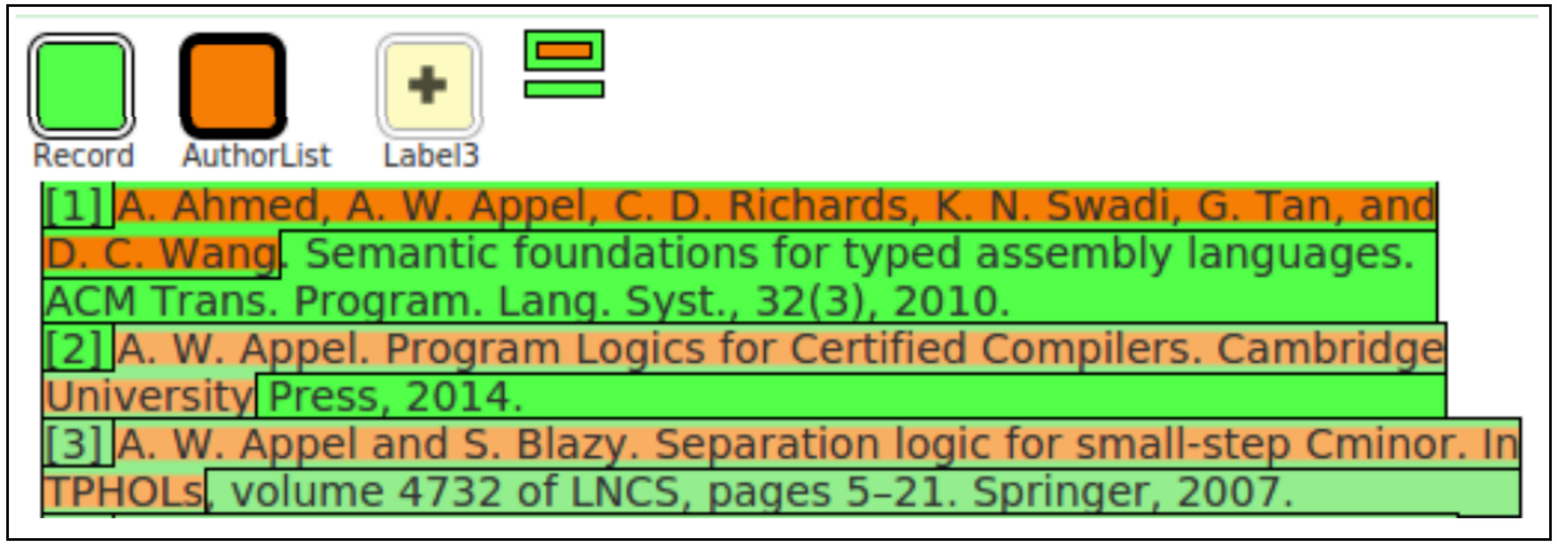
- Consider task: given a list of papers, extract all authors
- FlashProg requires user to do this in steps
  - First, extract papers using examples
  - Next, extract author lists using examples



Wrong program **Fix: provide another example?**

FlashProg System

- Consider task: given a list of papers, extract all authors
- FlashProg requires user to do this in steps
  - First, extract papers using examples
  - Next, extract author lists using examples



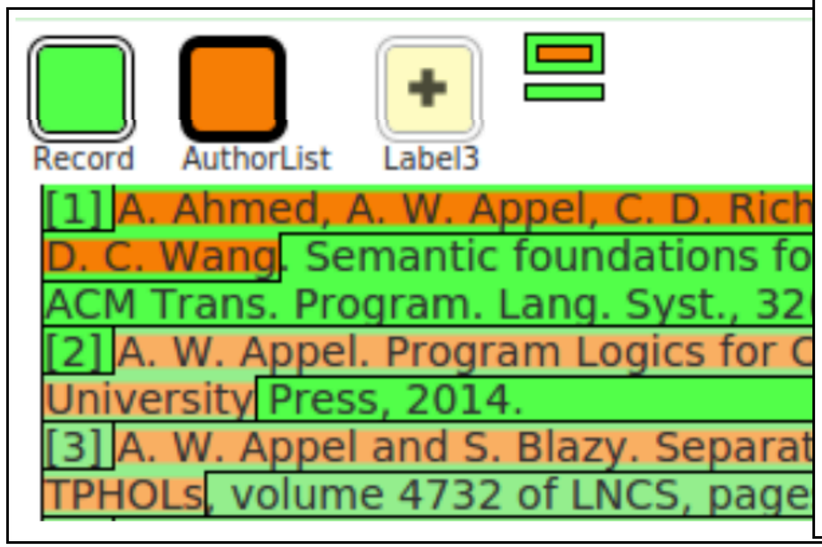
Wrong program

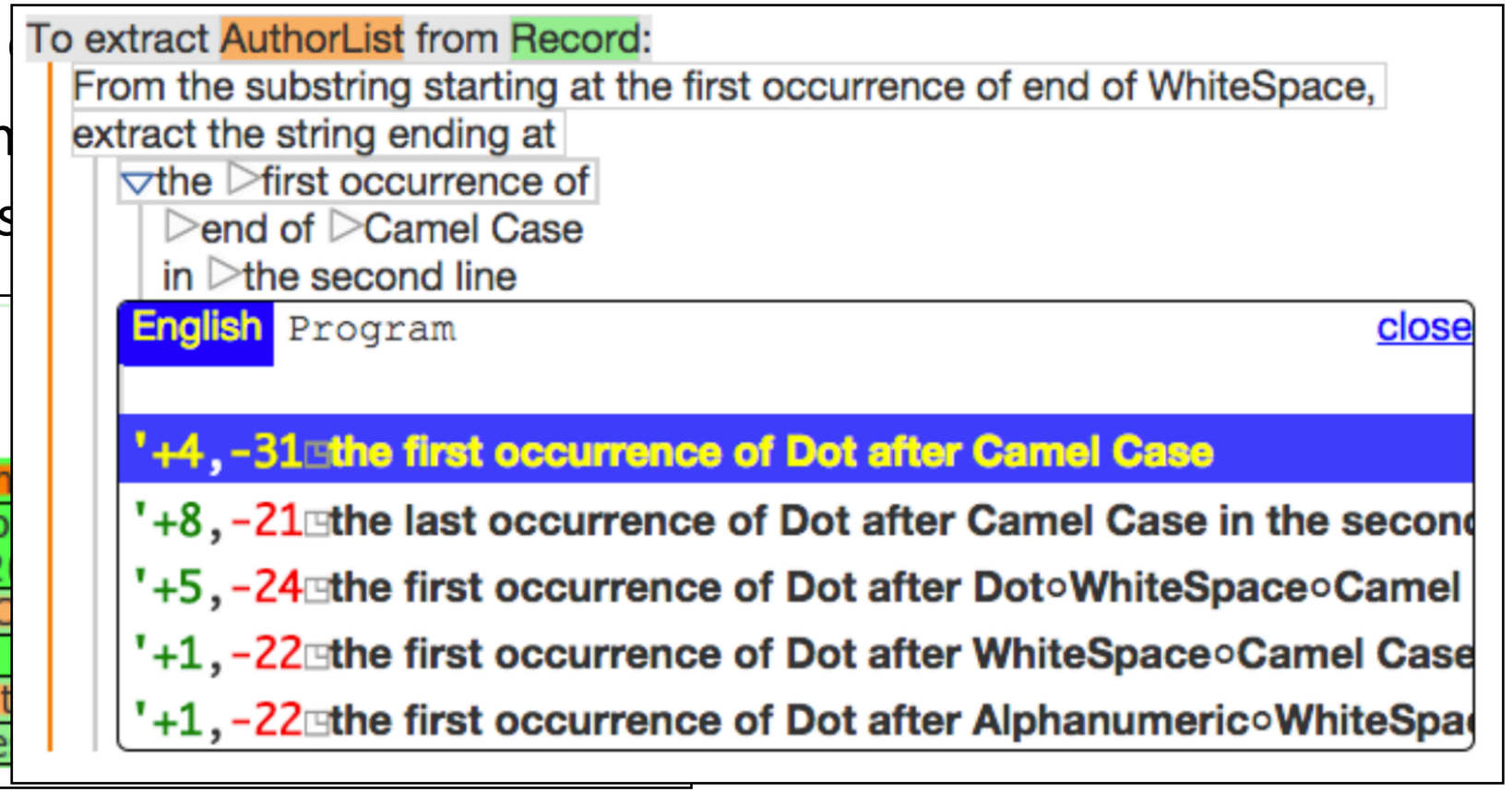
Fix: provide another example?

Or, choose a different program?

FlashProg System

- Consider task: given a list of papers, extract all authors
- FlashProg requires user to
  - First, extract papers usin
  - Next, extract author lists





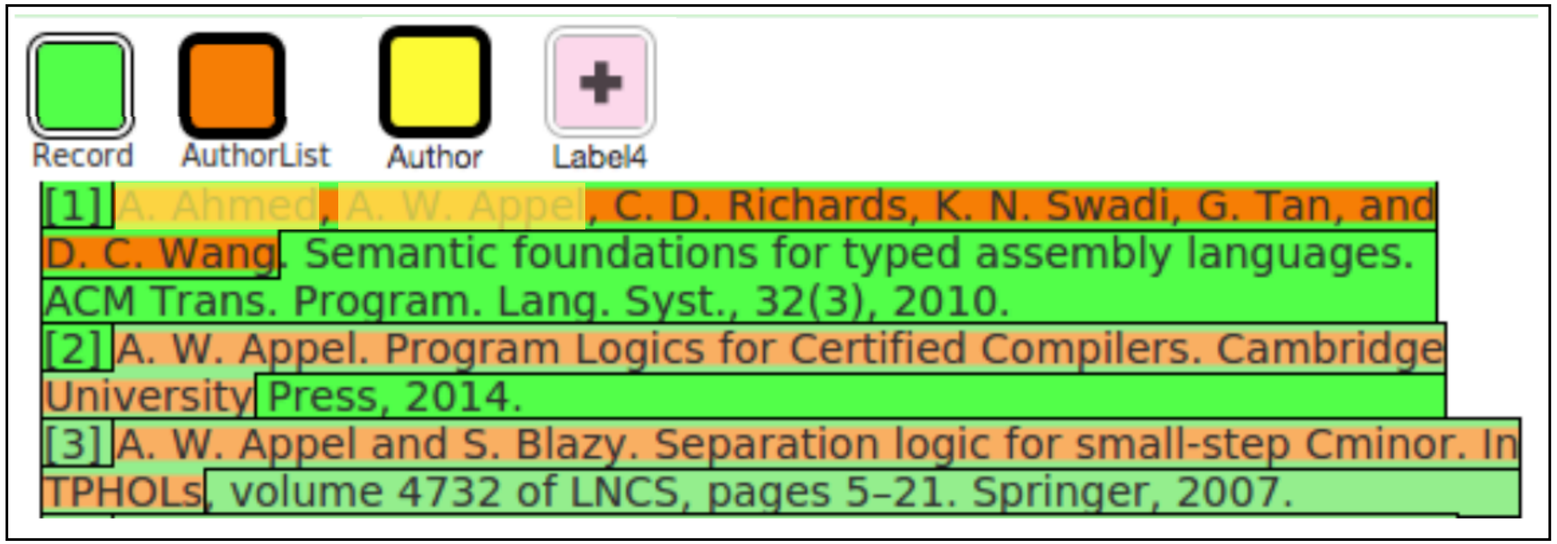
Wrong program

Fix: provide another example?

Or, choose a different program?

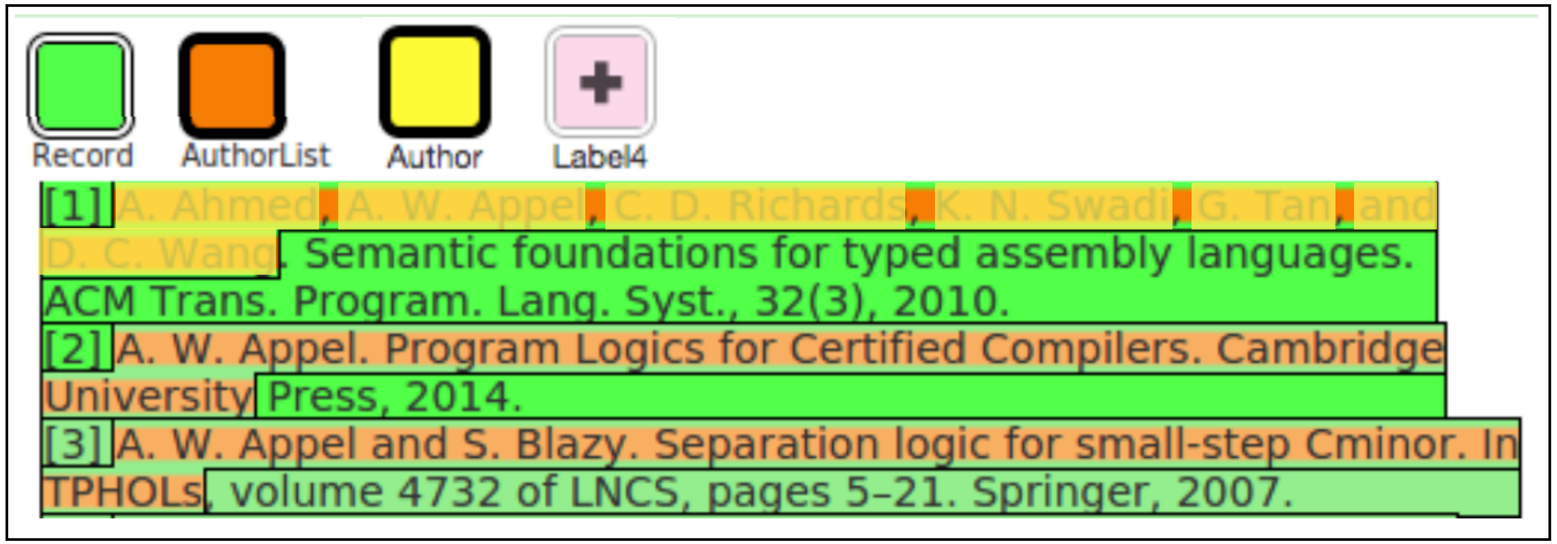
FlashProg System

- Consider task: given a list of papers, extract all authors
- FlashProg requires user to do this in steps
  - First, extract papers using examples
  - Next, extract author lists using examples
  - Finally, extract individual authors using examples



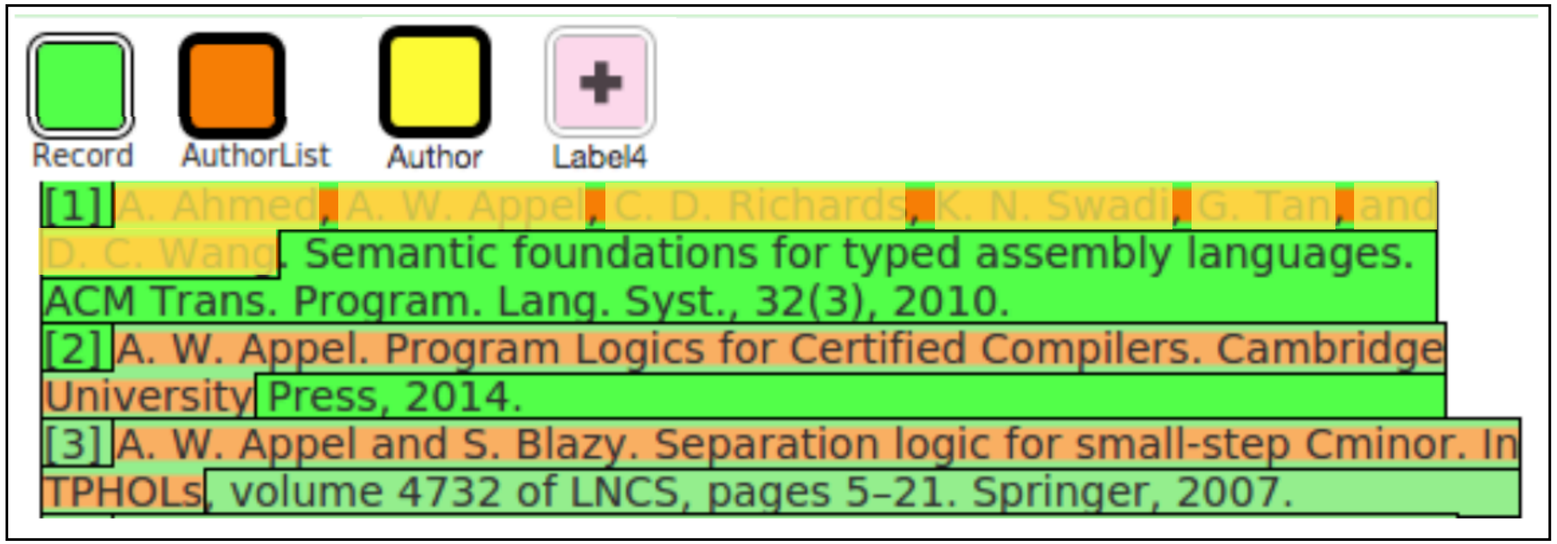
FlashProg System

- Consider task: given a list of papers, extract all authors
- FlashProg requires user to do this in steps
  - First, extract papers using examples
  - Next, extract author lists using examples
  - Finally, extract individual authors using examples



FlashProg System

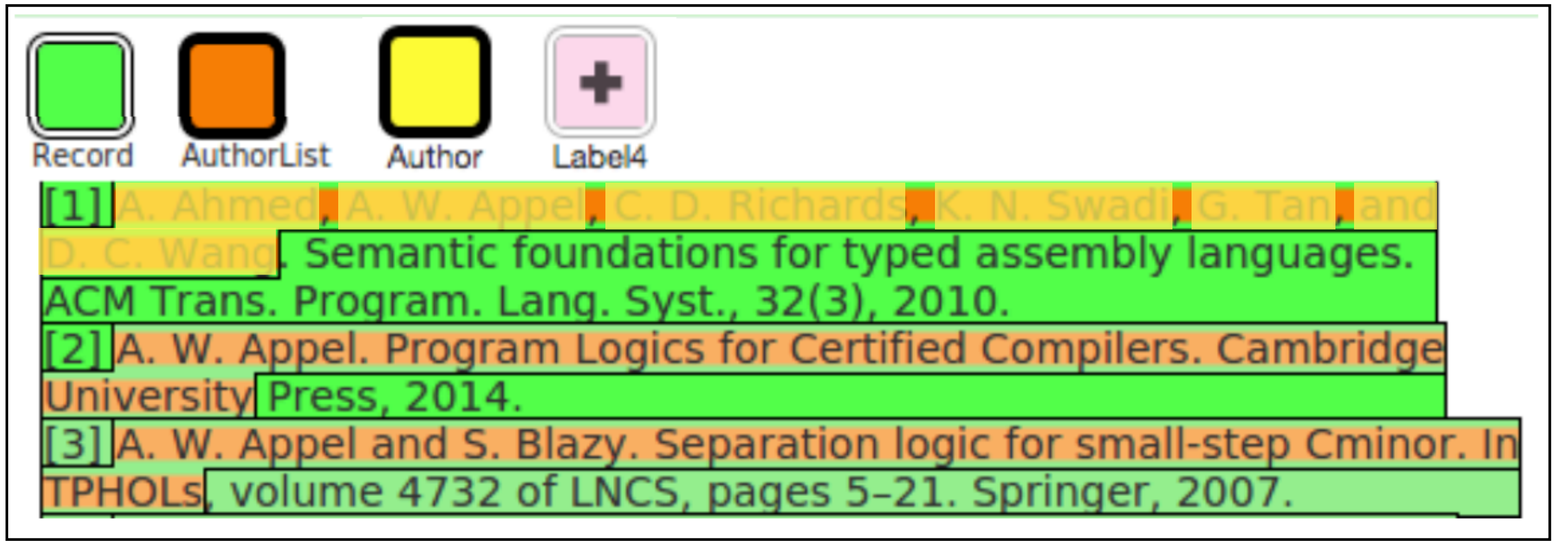
- Consider task: given a list of papers, extract all authors
- FlashProg requires user to do this in steps
  - First, extract papers using examples
  - Next, extract author lists using examples
  - Finally, extract individual authors using examples



Wrong program

FlashProg System

- Consider task: given a list of papers, extract all authors
- FlashProg requires user to do this in steps
  - First, extract papers using examples
  - Next, extract author lists using examples
  - Finally, extract individual authors using examples



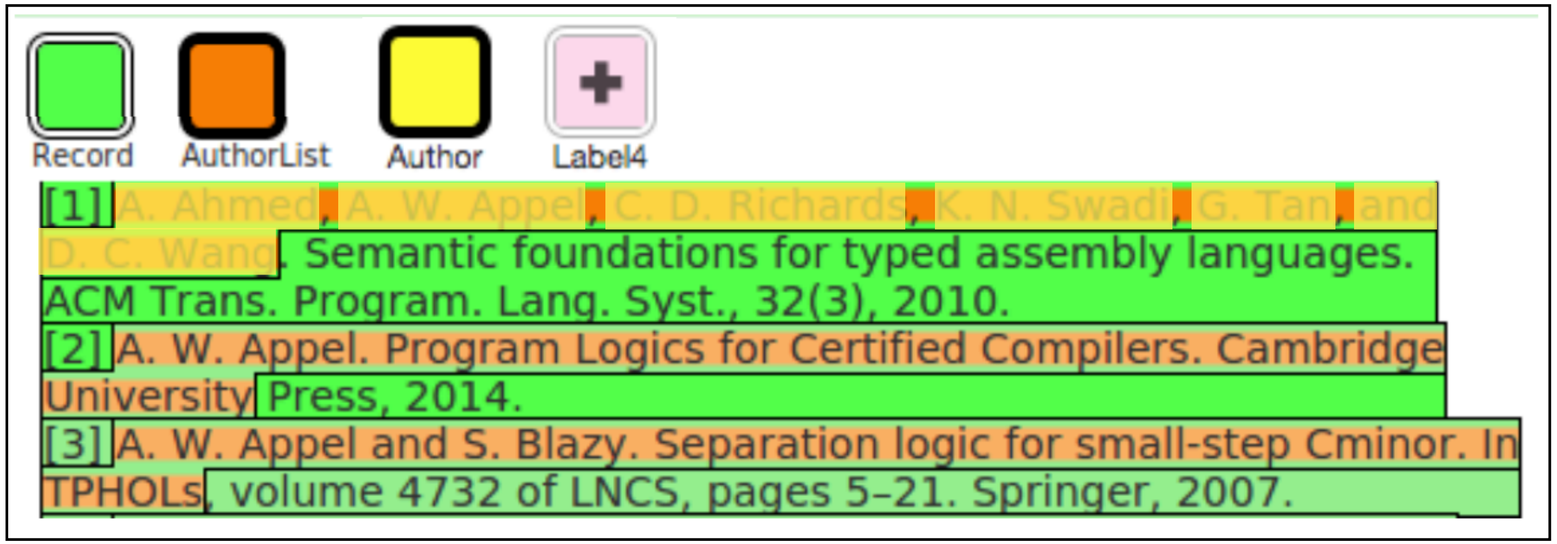
Wrong program

Fix: more examples?

Fix: choose another prog?

FlashProg System

- Consider task: given a list of papers, extract all authors
- FlashProg requires user to do this in steps
  - First, extract papers using examples
  - Next, extract author lists using examples
  - Finally, extract individual authors using examples



Wrong program

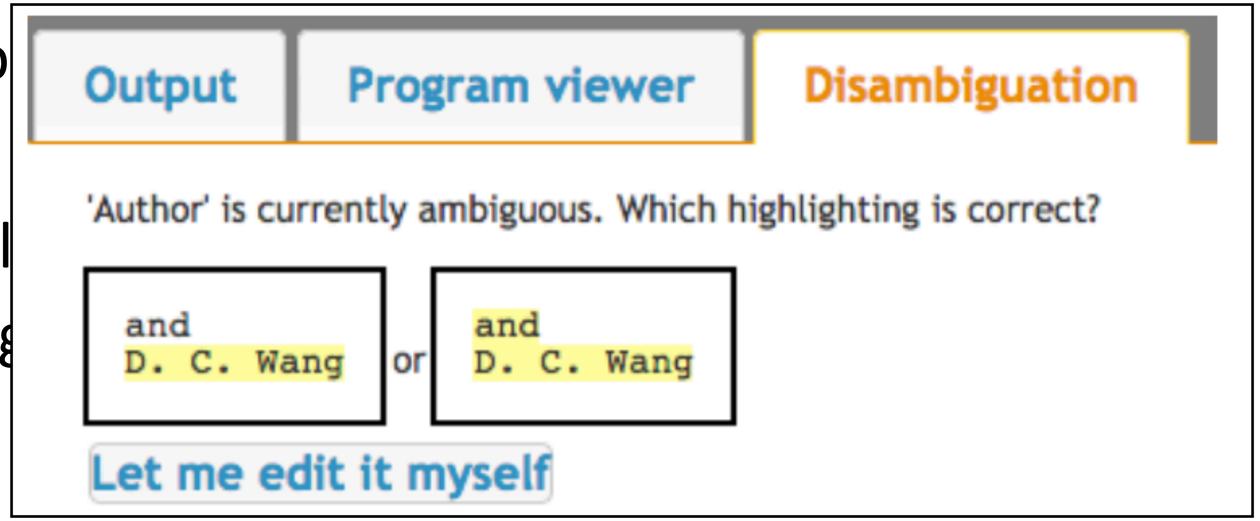
Fix: more examples?

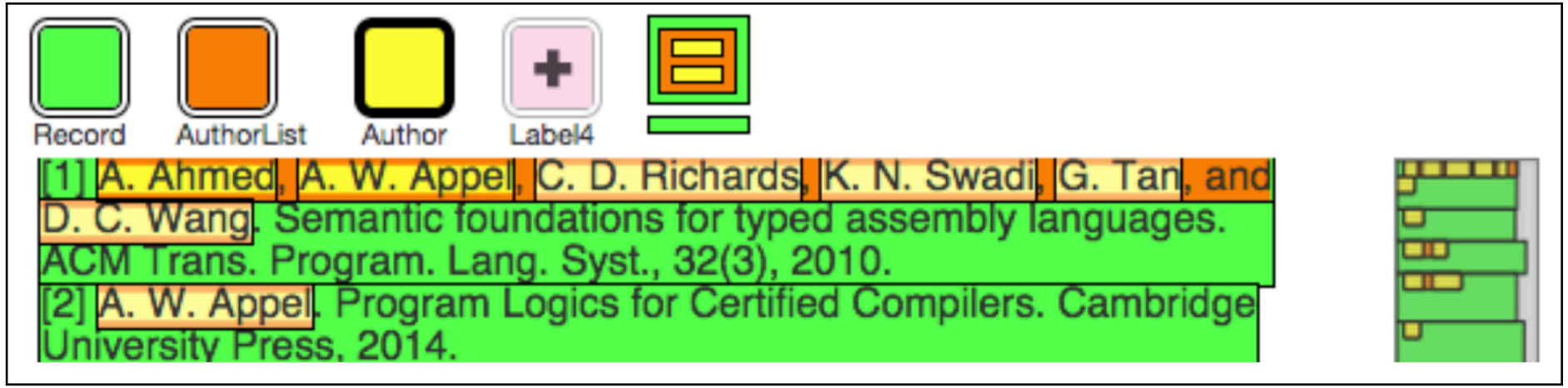
Fix: choose another prog?

Or, suggest prog?

FlashProg System

- Consider task: given a list of papers, extract all authors
- FlashProg requires user to do this in step
  - First, extract papers using examples
  - Next, extract author lists using exampl
  - Finally, extract individual authors using





FlashProg System

- When ambiguity occurs:

FlashProg System

- When ambiguity occurs:
  - User manually provides more examples

FlashProg System

- When ambiguity occurs:
  - User manually provides more examples
  - User manually selects a different program

FlashProg System

- When ambiguity occurs:
  - User manually provides more examples
  - User manually selects a different program
  - System automatically suggests input examples
    - User manually labels output examples
    - Effectively selects a different program

FlashProg System

- When ambiguity occurs:
  - User manually provides more examples
  - User manually selects a different program
  - System automatically suggests input examples
    - User manually labels output examples
    - Effectively selects a different program
- What’re some potential limitations?

Input Selection with Guarantees

- Prior work “randomly” selects input examples
- What guarantees can we provide?

Question Selection for Interactive Program Synthesis

Ruyi Ji
Key Lab of High Confidence Software Technologies, Ministry of Education
Department of Computer Science and Technology, EECS, Peking University
Beijing, China
jiruyi910387714@pku.edu.cn

Jingjing Liang
Key Lab of High Confidence Software Technologies, Ministry of Education
Department of Computer Science and Technology, EECS, Peking University
Beijing, China
jingjingliang@pku.edu.cn

Yingfei Xiong*
Key Lab of High Confidence Software Technologies, Ministry of Education
Department of Computer Science and Technology, EECS, Peking University
Beijing, China
xiongyf@pku.edu.cn

Lu Zhang
Key Lab of High Confidence Software Technologies, Ministry of Education
Department of Computer Science and Technology, EECS, Peking University
Beijing, China
zhanglucs@pku.edu.cn

Zhenjiang Hu
Key Lab of High Confidence Software Technologies, Ministry of Education
Department of Computer Science and Technology, EECS, Peking University
Beijing, China
huzj@pku.edu.cn

Abstract

In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '20)*

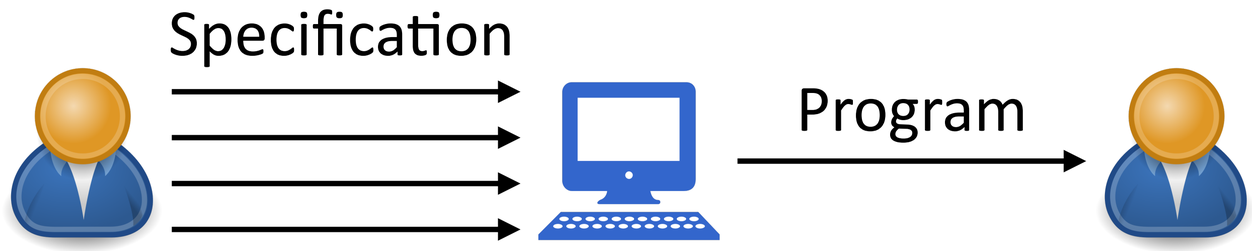
Question Selection for Interactive Program Synthesis

- Key idea: even with worst user answer, question would result in best reduction
- Technique: based on minimax branch
- Evaluation result: reduce number of questions by 2x

Improving Generalization

- Ranking (inductive bias)
- Interaction
- **<u>Multi-modality</u>**

Multi-Modal Program Synthesis



- Multi-modal specification:
  - Multiple **kinds** of specifications: examples, natural language, etc.

Multi-Modal Program Synthesis

- Multi-modal specification:
  - Multiple **kinds** of specifications: examples, natural language, etc.
- Related work
  - Multi-Modal Synthesis of Regular Expressions — examples + NL
  - Interactive Program Synthesis by Augmented Examples — examples + annotations
  - LooPy: Interactive Program Synthesis with Control Structures — examples + partial program
  - TF-Coder: Program Synthesis for Tensor Manipulations — examples + NL + constants
  - Etc.

Synthesis of Regular Expressions

- Classic problem dating back to 1980s
  - Seminal L* work by Angluin 1987

INFORMATION AND COMPUTATION 75, 87–106 (1987)

Learning Regular Sets from Queries and Counterexamples*

DANA ANGLUIN

Department of Computer Science, Yale University,
P.O. Box 2158, Yale Station, New Haven, Connecticut 06520

The problem of identifying an unknown regular set from examples of its members and nonmembers is addressed. It is assumed that the regular set is presented by a

L* Algorithm

- Problem: identify regular set from examples
  - Consider positive examples (members) and negative examples (nonmembers)

L* Algorithm

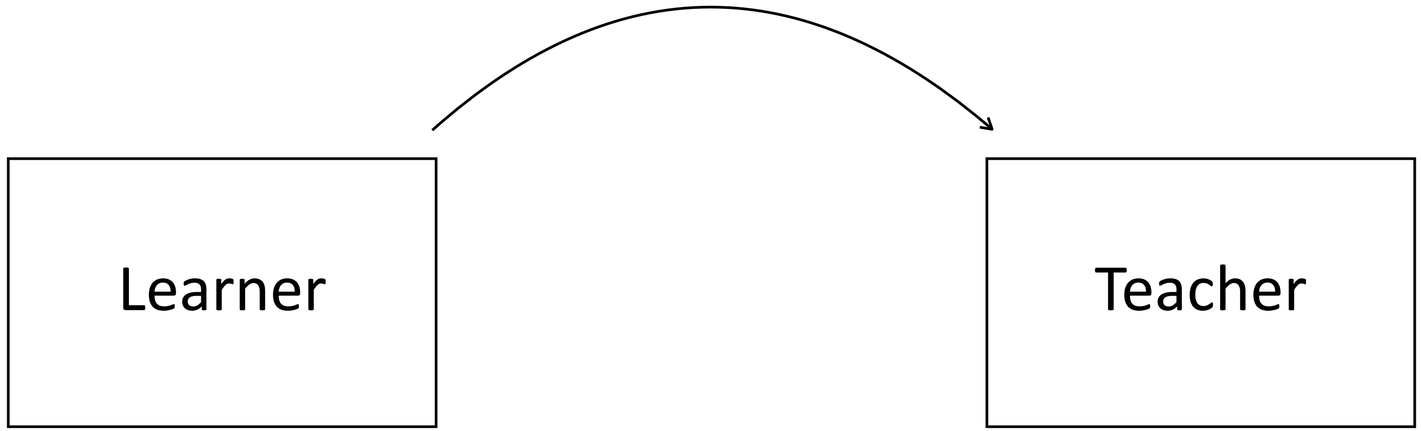
- Problem: identify regular set from examples
  - Consider positive examples (members) and negative examples (nonmembers)

L* Algorithm

- Problem: identify regular set from examples
  - Consider positive examples (members) and negative examples (nonmembers)

Query 1: is string s in the set?

Query 2: is regex r equivalent to the desired regex?

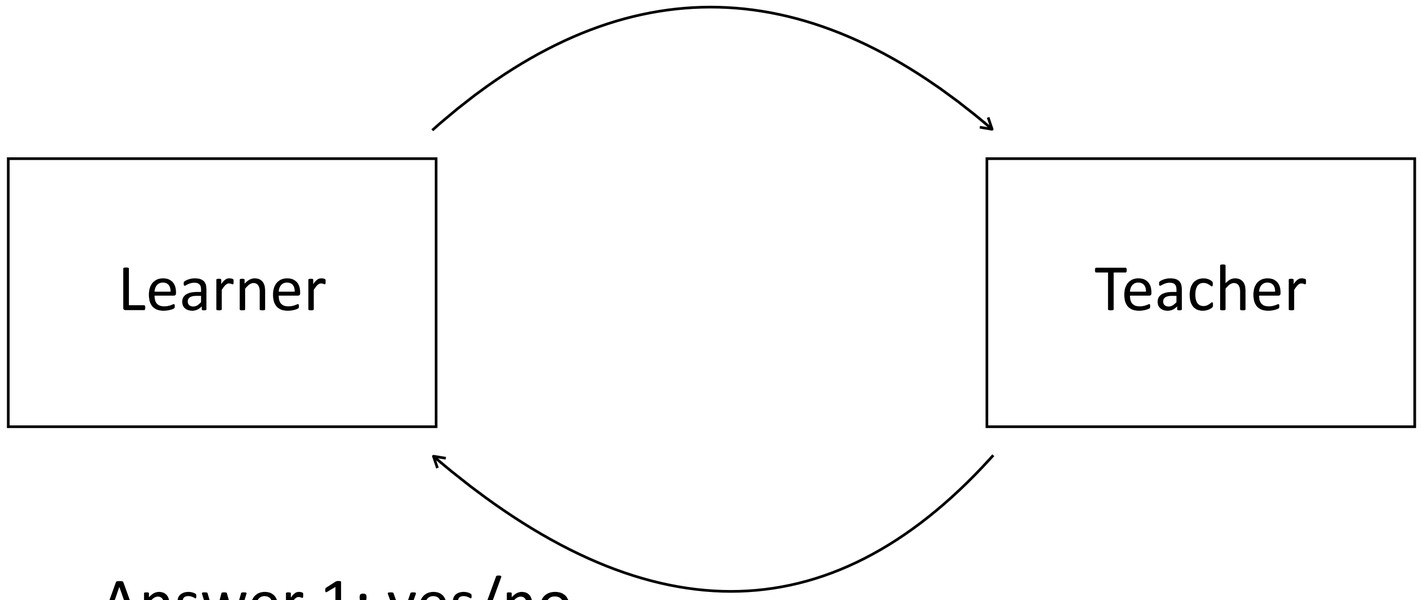


L* Algorithm

- Problem: identify regular set from examples
  - Consider positive examples (members) and negative examples (nonmembers)

Query 1: is string s in the set?

Query 2: is regex r equivalent to the desired regex?



Answer 1: yes/no

Answer 2: yes/no (and here is a counterexample s)

L* Algorithm

- Main result
  - Learn any regular set from examples
  - Time polynomial in number of states of the corresponding minimum DFA
  - Time polynomial in maximum length of any counterexample provided by Teacher

Regular Expression Synthesis Revisited

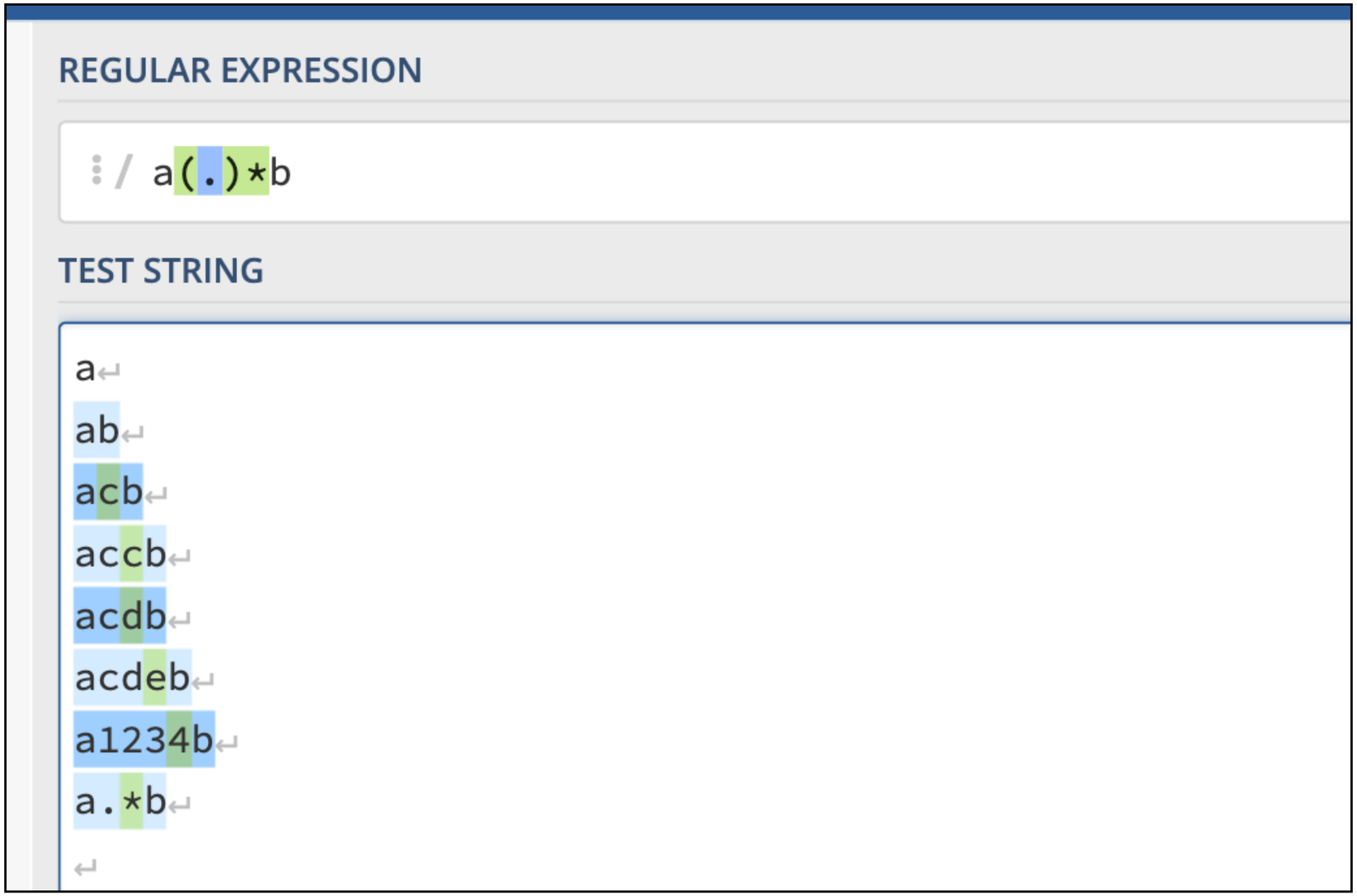
- A lot of work recently
  - Synthesizing Regular Expressions from Examples for Introductory Automata Assignments
  - Automatic Repair of Regular Expressions
  - Multi-Modal Synthesis of Regular Expressions
  - Sketch-Driven Regular Expression Generation from Natural Language and Examples
  - Automatic repair of vulnerable regular expressions
  - Interactive Program Synthesis by Augmented Examples
  - Optimal Neural Program Synthesis from Multimodal Specifications
  - Multi-modal Program Inference: a Marriage of Pre-trainedLanguage Models and Component-based Synthesis

Regular Expressions 101

- Regexes
  - Used for pattern matching strings
  - Given regex r and string s, r either matches s or r doesn't match s
  - https://regex101.com/

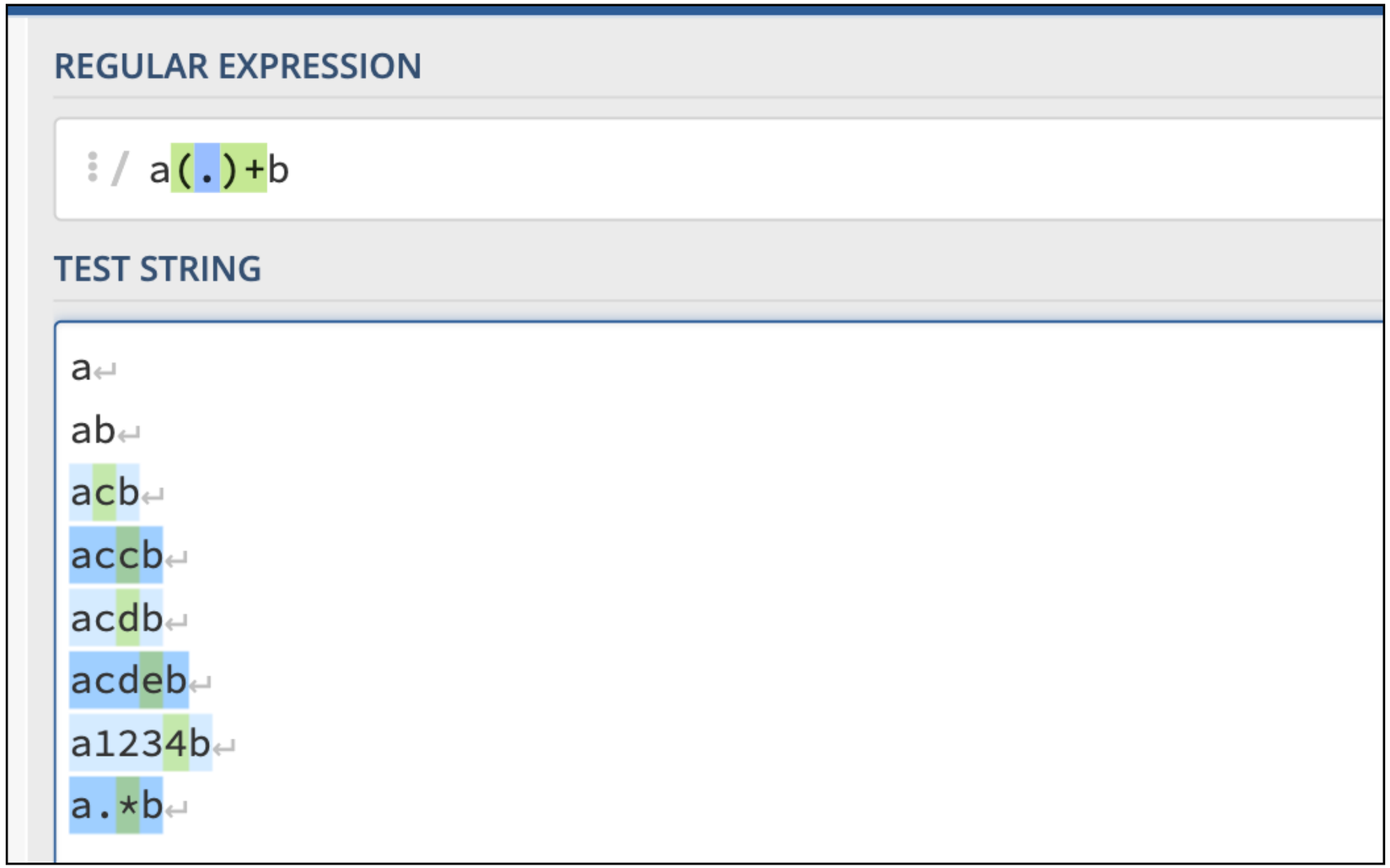
Regular Expressions 101

- **`a(.)*b`** matches strings that start with `a`, end with `b`, have **anything** in between



Regular Expressions 101

- **`a(.)+b`** matches strings that start with `a`, end with `b`, have **anything non-empty** in between



Regular Expressions 101

Anchors

^	Start of string, or start of line in multi-line pattern
\A	Start of string
$	End of string, or end of line in multi-line pattern
\Z	End of string
\b	Word boundary
\B	Not word boundary
\<	Start of word
\>	End of word

Character Classes

\c	Control character
\s	White space
\S	Not white space
\d	Digit
\D	Not digit
\w	Word
\W	Not word
\x	Hexadecimal digit
\O	Octal digit

Quantifiers

*	0 or more	{3}	Exactly 3
+	1 or more	{3,}	3 or more
?	0 or 1	{3,5}	3, 4 or 5

Add a ? to a quantifier to make it ungreedy.

Escape Sequences

\	Escape following character
\Q	Begin literal sequence
\E	End literal sequence

"Escaping" is a way of treating characters which have a special meaning in regular expressions literally, rather than as special characters.

Common Metacharacters

^	[	.	$
{	*	(	\
+	)	|	?
<	>		

The escape character is usually \

Special Characters

Groups and Ranges

.	Any character except new line (\n)
(a|b)	a or b
(...)	Group
(?:...)	Passive (non-capturing) group
[abc]	Range (a or b or c)
[^abc]	Not (a or b or c)
[a-q]	Lower case letter from a to q
[A-Q]	Upper case letter from A to Q
[0-7]	Digit from 0 to 7
\x	Group/subpattern number "x"

Ranges are inclusive.

Pattern Modifiers

g	Global match
i *	Case-insensitive
m *	Multiple lines
s *	Treat string as single line
x *	Allow comments and whitespace in pattern
e *	Evaluate replacement
U *	Ungreedy pattern

* PCRE modifier

Regular Expressions 101

Anchors

^	Start of string, or start of line in multi-line pattern
\A	Start of string
$	End of string, or end of line in multi-line pattern
\Z	End of string
\b	Word boundary
\B	Not word boundary
\<	Start of word
\>	End of word

Character Classes

\c	Control character
\s	White space
\S	Not white space
\d	Digit
\D	Not digit
\w	Word
\W	Not word
\x	Hexadecimal digit
\O	Octal digit

Quantifiers

*	0 or more	{3}	Exactly 3
+	1 or more	{3,}	3 or more
?	0 or 1	{3,5}	3, 4 or 5

Add a ? to a quantifier to make it ungreedy.

Escape Sequences

\	Escape following character
\Q	Begin literal sequence
\E	End literal sequence

"Escaping" is a way of treating characters which have a special meaning in regular expressions literally, rather than as special characters.

Common Metacharacters

^	[	.	$
{	*	(	\
+	)	|	?
<	>		

The escape character is usually \

Special Characters

Groups and Ranges

.	Any character except new line (\n)
(a|b)	a or b
(...)	Group
(?:...)	Passive (non-capturing) group
[abc]	Range (a or b or c)
[^abc]	Not (a or b or c)
[a-q]	Lower case letter from a to q
[A-Q]	Upper case letter from A to Q
[0-7]	Digit from 0 to 7
\x	Group/subpattern number "x"

Ranges are inclusive.

Pattern Modifiers

g	Global match
i *	Case-insensitive
m *	Multiple lines
s *	Treat string as single line
x *	Allow comments and whitespace in pattern
e *	Evaluate replacement
U *	Ungreedy pattern

* PCRE modifier

Regular Expressions 101

Anchors

^	Start of string, or start of line in multi-line pattern
\A	Start of string
$	End of string, or end of line in multi-line pattern
\Z	End of string
\b	Word boundary
\B	Not word boundary
\<	Start of word
\>	End of word

Character Classes

\c	Control character
\s	White space
\S	Not white space
\d	Digit
\D	Not digit
\w	Word
\W	Not word
\x	Hexadecimal digit
\O	Octal digit

Quantifiers

*	0 or more	{3}	Exactly 3
+	1 or more	{3,}	3 or more
?	0 or 1	{3,5}	3, 4 or 5

Add a ? to a quantifier to make it ungreedy.

Escape Sequences

\	Escape following character
\Q	Begin literal sequence
\E	End literal sequence

"Escaping" is a way of treating characters which have a special meaning in regular expressions literally, rather than as special characters.

Common Metacharacters

^	[	.	$
{	*	(	\
+	)	|	?
<	>		

The escape character is usually \

Special Characters

Groups and Ranges

.	Any character except new line (\n)
(a|b)	a or b
(...)	Group
(?:...)	Passive (non-capturing) group
[abc]	Range (a or b or c)
[^abc]	Not (a or b or c)
[a-q]	Lower case letter from a to q
[A-Q]	Upper case letter from A to Q
[0-7]	Digit from 0 to 7
\x	Group/subpattern number "x"

Ranges are inclusive.

Pattern Modifiers

g	Global match
i *	Case-insensitive
m *	Multiple lines
s *	Treat string as single line
x *	Allow comments and whitespace in pattern
e *	Evaluate replacement
U *	Ungreedy pattern

* PCRE modifier

Regular Expressions 101

- Different dialects/variants
  - Sometimes used to find strings

Regular Expressions 101

- Different dialects/variants
  - Sometimes used to find strings
- Common features
  - Character classes (e.g., \d, \w, \s)
  - Klenee start and quantifiers

Regular Expressions 101

- Different dialects/variants
  - Sometimes used to find strings
- Common features
  - Character classes (e.g., \d, \w, \s)
  - Klenee start and quantifiers
- Focus on a small (but still expressive) regex DSL

Regular Expressions 101

- Consider CFG (simplified):

r ::= cc | StartWith(r) | EndWith(r) | Contains(r) | Concat(r, r) | KleeneStar(r) | Repeat(r, k)

cc ::= <digit> | <char> | <any> | ..

k ::= 1 | 2 | ..

Regular Expressions 101

- Consider CFG (simplified):

  r ::= cc | StartWith(r) | EndWith(r) | Contains(r) | Concat(r, r) | KleeneStar(r) | Repeat(r, k)

  cc ::= <digit> | <char> | <any> | ..

  k ::= 1 | 2 | ..

- Regular expressions in this DSL:
  - StartWith(<digit>)

Regular Expressions 101

- Consider CFG (simplified):

  r ::= cc | StartWith(r) | EndWith(r) | Contains(r) | Concat(r, r) | KleeneStar(r) | Repeat(r, k)

  cc ::= <digit> | <char> | <any> | ..

  k ::= 1 | 2 | ..

- Regular expressions in this DSL:
  - StartWith(<digit>)
  - Concat(a, Concat(KleeneStar(<any>), b))

Regular Expressions 101

- Consider CFG (simplified):

  r ::= cc | StartWith(r) | EndWith(r) | Contains(r) | Concat(r, r) | KleeneStar(r) | Repeat(r, k)

  cc ::= <digit> | <char> | <any> | ..

  k ::= 1 | 2 | ..

- Regular expressions in this DSL:
  - StartWith(<digit>)
  - Concat(a, Concat(KleeneStar(<any>), b))
  - Repeat(<char>, 5)

How To Explain A Regex?

r ::= cc | StartWith(r) | EndWith(r) | Contains(r) | Concat(r, r) | KleeneStar(r) | Repeat(r, k)

cc ::= <digit> | <char> | <any> | ..

k ::= 1 | 2 | ..

- Given positive and negative strings, find regex that matches **all** positive strings and **none** of negative strings

How To Explain A Regex?

r ::= cc | StartWith(r) | EndWith(r) | Contains(r) | Concat(r, r) | KleeneStar(r) | Repeat(r, k)

cc ::= <digit> | <char> | <any> | ..

k ::= 1 | 2 | ..

- Given positive and negative strings, find regex that matches **all** positive strings and **none** of negative strings
  - { “a”+ }, what’s a regex?

How To Explain A Regex?

r ::= cc | StartWith(r) | EndWith(r) | Contains(r) | Concat(r, r) | KleeneStar(r) | Repeat(r, k)

cc ::= <digit> | <char> | <any> | ..

k ::= 1 | 2 | ..

- Given positive and negative strings, find regex that matches **all** positive strings and **none** of negative strings
  - { “a”+ }, what’s a regex?
  - { “a”+, “b”- }, what’s a regex?

How To Explain A Regex?

r ::= cc | StartWith(r) | EndWith(r) | Contains(r) | Concat(r, r) | KleeneStar(r) | Repeat(r, k)

cc ::= <digit> | <char> | <any> | ..

k ::= 1 | 2 | ..

- Given positive and negative strings, find regex that matches **all** positive strings and **none** of negative strings
  - { “a”+ }, what’s a regex?
  - { “a”+, “b”- }, what’s a regex?
- Take-away: examples are not sufficient (under-constrained)

How To Explain A Regex?

r ::= cc | StartWith(r) | EndWith(r) | Contains(r) | Concat(r, r) | KleeneStar(r) | Repeat(r, k)

cc ::= <digit> | <char> | <any> | ..

k ::= 1 | 2 | ..

- Or, we can use natural language to describe our intent
  - Write a regex that matches only capital letters

How To Explain A Regex?

r ::= cc | StartWith(r) | EndWith(r) | Contains(r) | Concat(r, r) | KleeneStar(r) | Repeat(r, k)

cc ::= <digit> | <char> | <any> | ..

k ::= 1 | 2 | ..

- Or, we can use natural language to describe our intent
  - Write a regex that matches only capital letters

- Take-away: NL is ambiguous

How To Explain A Regex?

- Examples + NL

I need a regular expression that can validate `Decimal (18,3)`, meaning a precision of 18, and a scale of 3.

It would need to pass the following criteria:

- Max number of digits before comma is `15`
- Max number of digits after the comma is `3`

Valid Examples:

```
123456789.123
123456789123456.12
12345.1
123456789123456
```

Not Valid Examples:

```
1234567891234567
123.1234
1.12345
.1234
```

How can I achieve this?

Regel System

- Key insight: use both examples and natural language as specification

Multi-modal Synthesis of Regular Expressions

Qiaochu Chen
University of Texas at Austin
Austin, Texas, USA
qchen@cs.utexas.edu

Xinyu Wang
University of Michigan, Ann Arbor
Ann Arbor, Michigan, USA
xwangsd@umich.edu

Xi Ye
University of Texas at Austin
Austin, Texas, USA
xiye@cs.utexas.edu

Greg Durrett
University of Texas at Austin
Austin, Texas, USA
gdurrett@cs.utexas.edu

Isil Dillig
University of Texas at Austin
Austin, Texas, USA
isil@cs.utexas.edu

Abstract

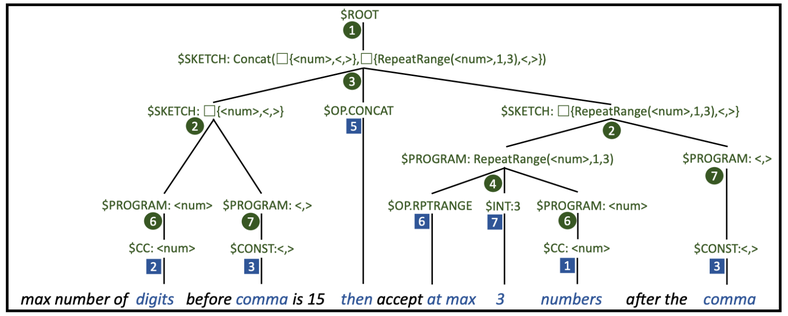
In this paper, we propose a multi-modal synthesis technique

Keywords: Program Synthesis, Programming by Natural Languages, Programming by Example, Regular Expression

Regel System

NL

Examples



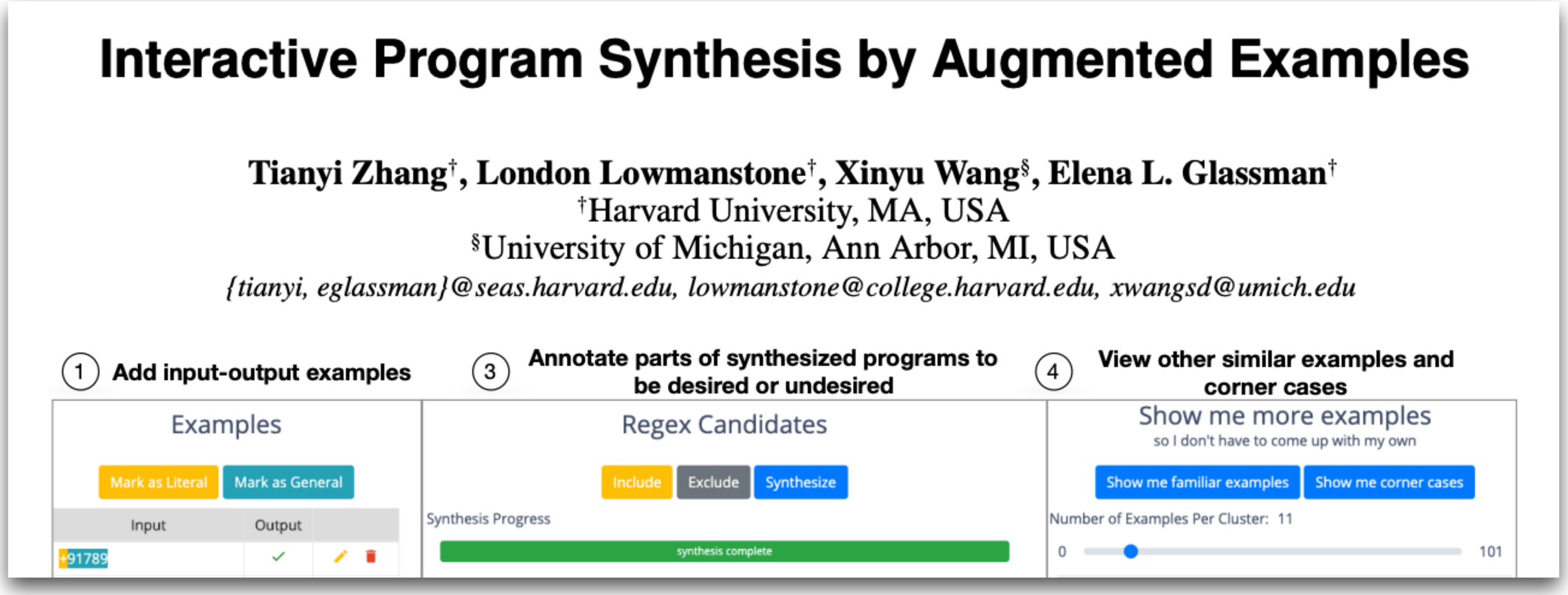
Sketch

```
procedure SYNTHESIZE(S, E+, E−)
input: an h-sketch S, positive and negative examples E+, E−
output: a regex consistent with S, E+ and E−, or ⊥
  P0 := (v0, ∅, [v0 ◂ S]);  worklist := {P0};
  while worklist ≠ ∅ do
    P := worklist.remove();
    if IsConcrete(P) then
      if IsCorrect(P, E+, E−) then return P;
    else if IsSymbolic(P) then
      worklist := worklist ∪ INFERCONSTANTS(P, E+, E−);
    else
      (v, S) := SelectOpenNode(P);
      worklist′ := EXPAND(P, v, S);
      for all P′ ∈ worklist′ do
        if INFEASIBLE(P′, E+, E−) then
          worklist′.remove(P′);
      worklist := worklist ∪ worklist′;
  return ⊥;
```

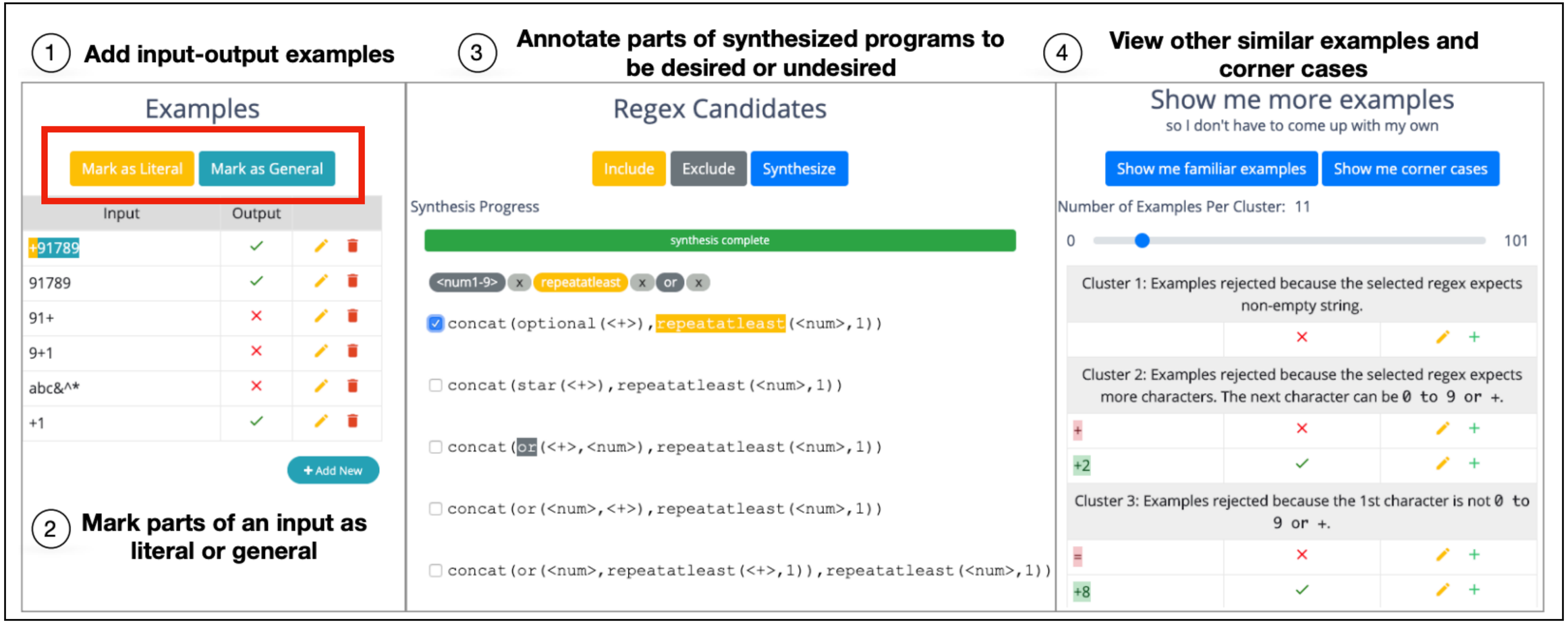
Regular Expression

Regae System

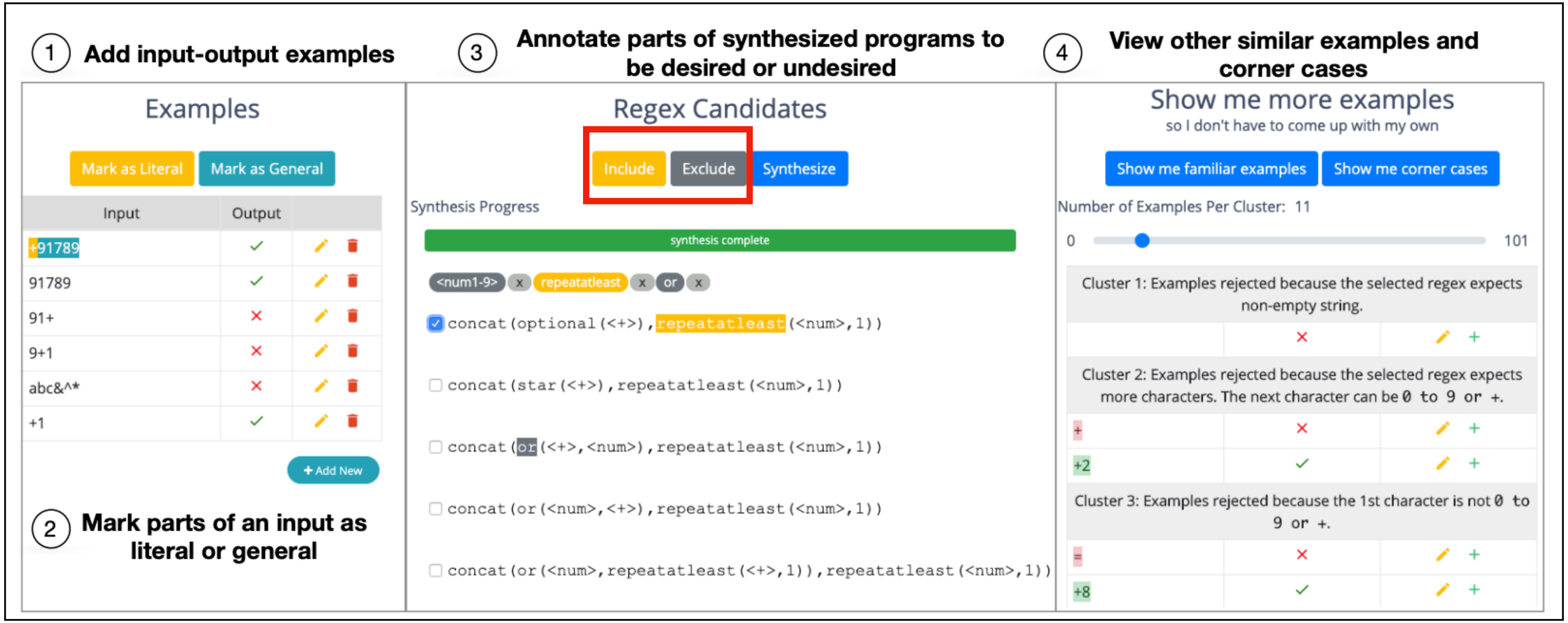
- Key insight: even richer specification modalities through UI



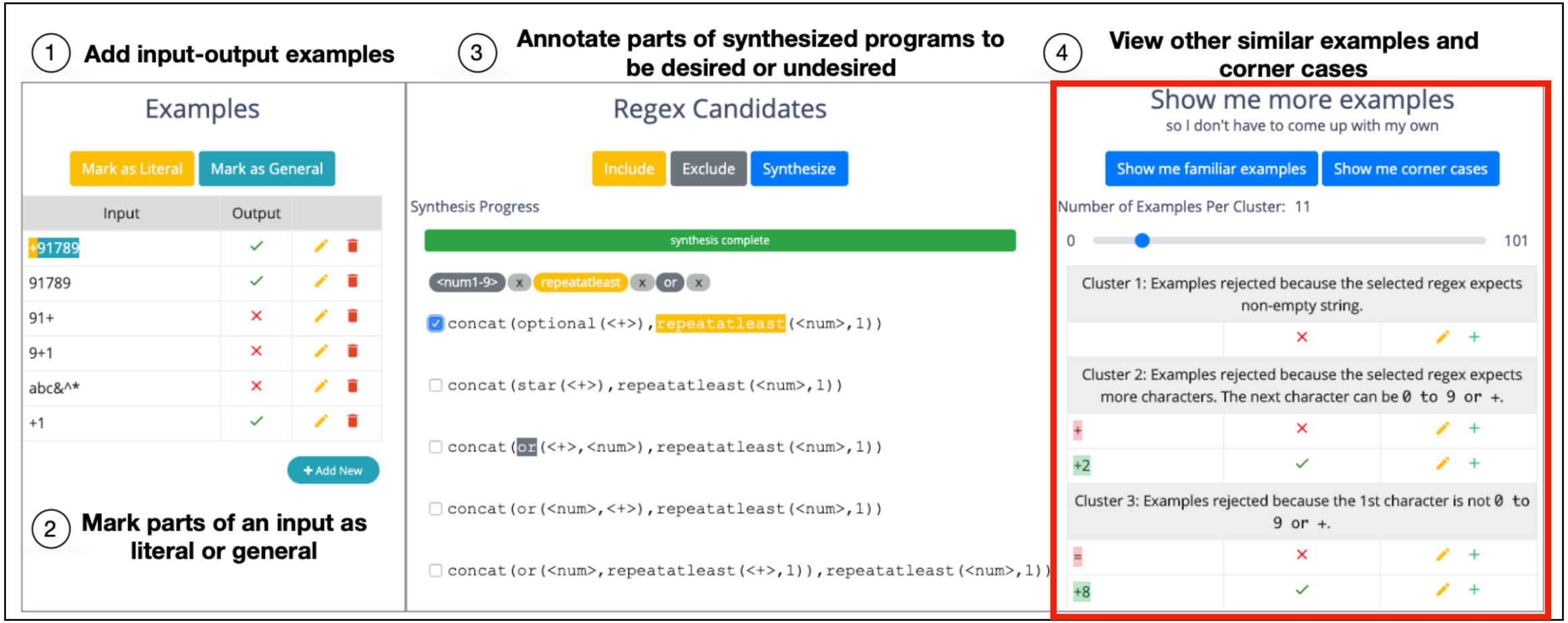
Regae System



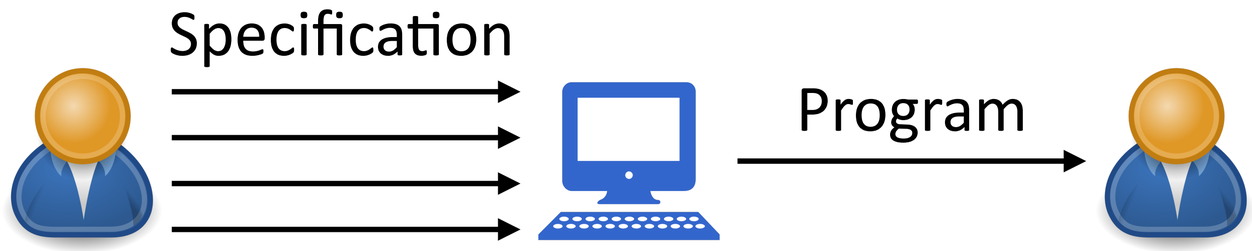
Regae System



Regae System

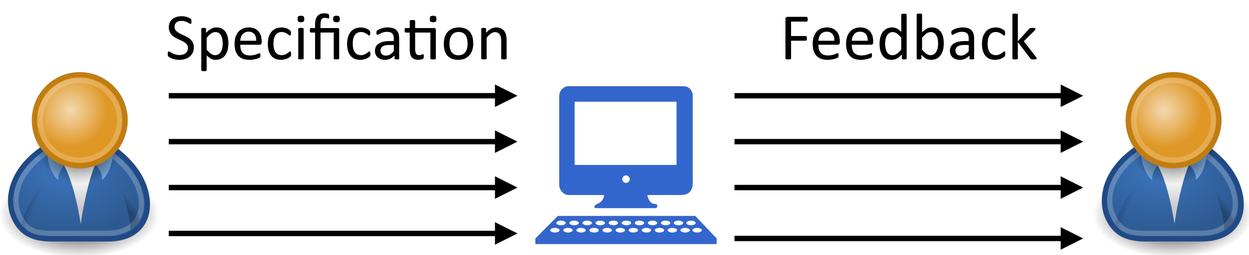


Multi-Modal Program Synthesis



- Multi-modal specification:
  - Multiple **kinds** of specifications: examples, natural language, etc.

Multi-Modal Program Synthesis



- Multi-modal specification:
  - Multiple **kinds** of specifications: examples, natural language, etc.
  - Multiple kinds of **feedback**: synthesized programs, examples, etc.

Multi-Modal Program Synthesis

- Not yet well explored

Interpretable Program Synthesis

Tianyi Zhang
Harvard University
Cambridge, MA, USA
tianyi@harvard.seas.edu

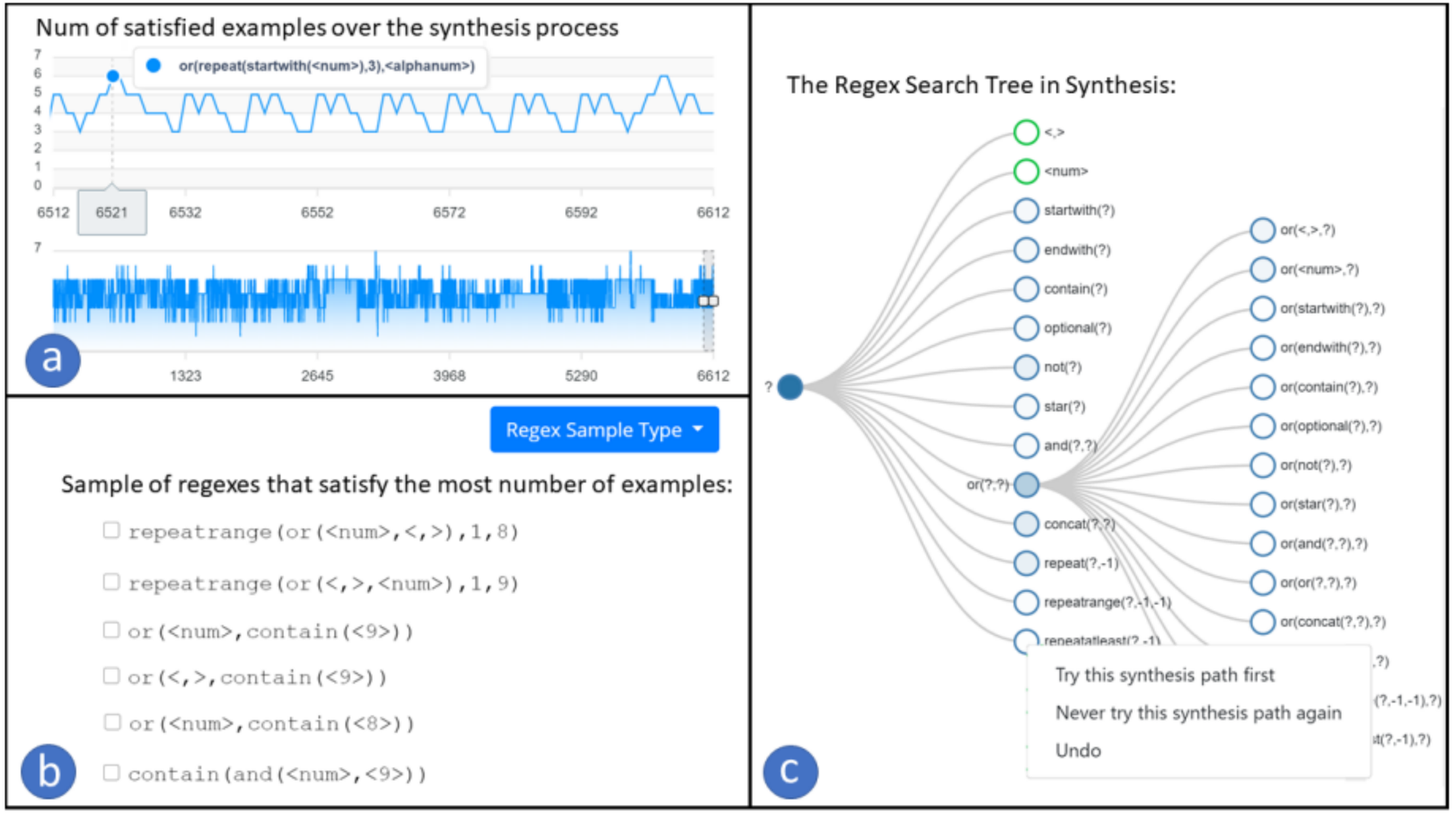
Zhiyang Chen
University of Michigan
Ann Arbor, MI, USA
zhiychen@umich.edu

Yuanli Zhu
University of Michigan
Ann Arbor, MI, USA
leozhu@umich.edu

Priyan Vaithilingam
Harvard University
Cambridge, MA, USA
pvaithilingam@g.harvard.edu

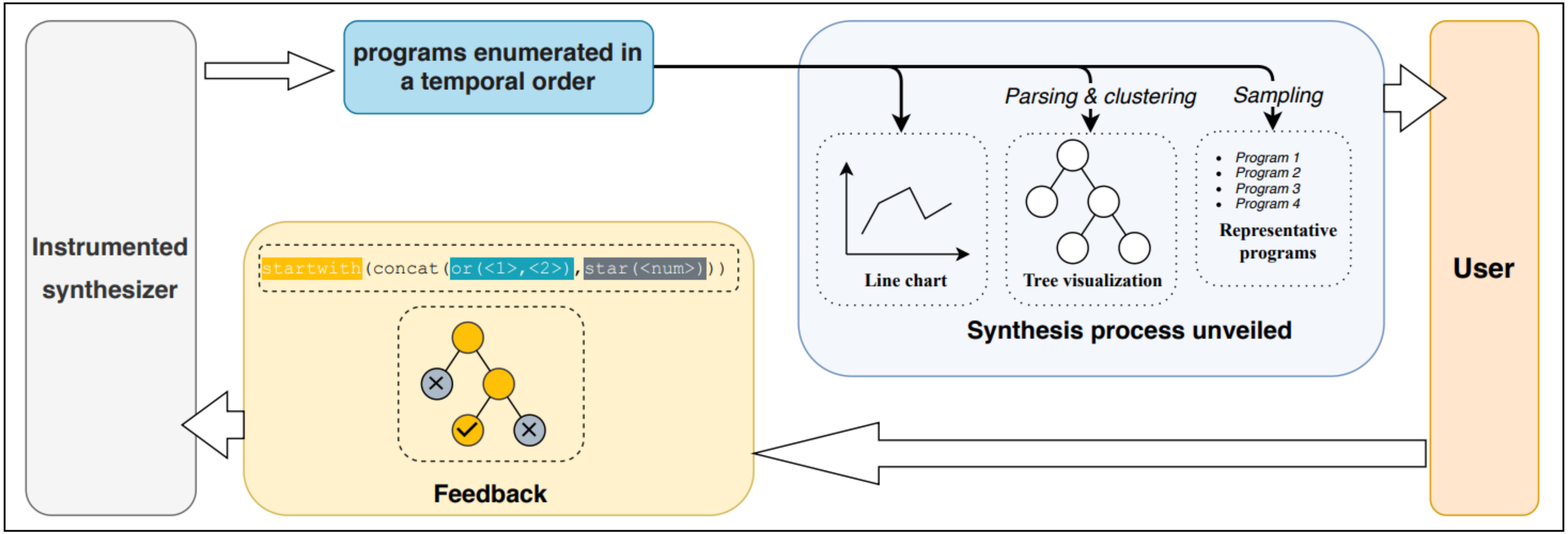
Xinyu Wang
University of Michigan
Ann Arbor, MI, USA
xwangsd@umich.edu

Elena L. Glassman
Harvard University
Cambridge, MA, USA
glassman@seas.harvard.edu



Interpretable Program Synthesis

- Key idea: visualize synthesis process to users



Interactive, Multi-Modal Program Synthesis