

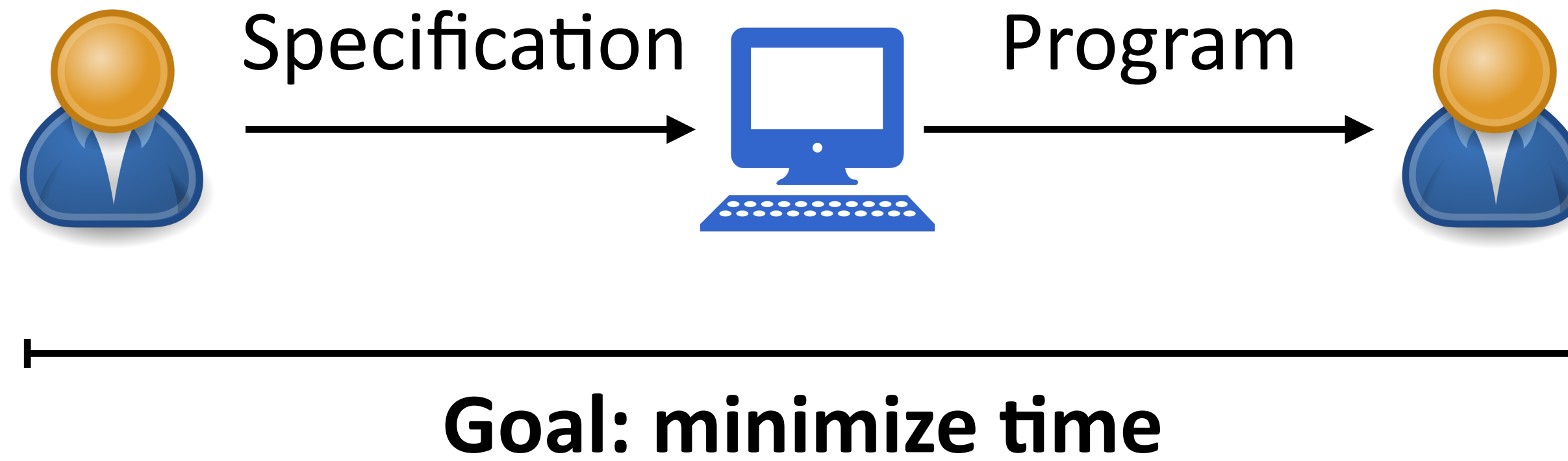
EECS 598-008 & EECS 498-008: Intelligent Programming Systems

Lecture 10

Announcements

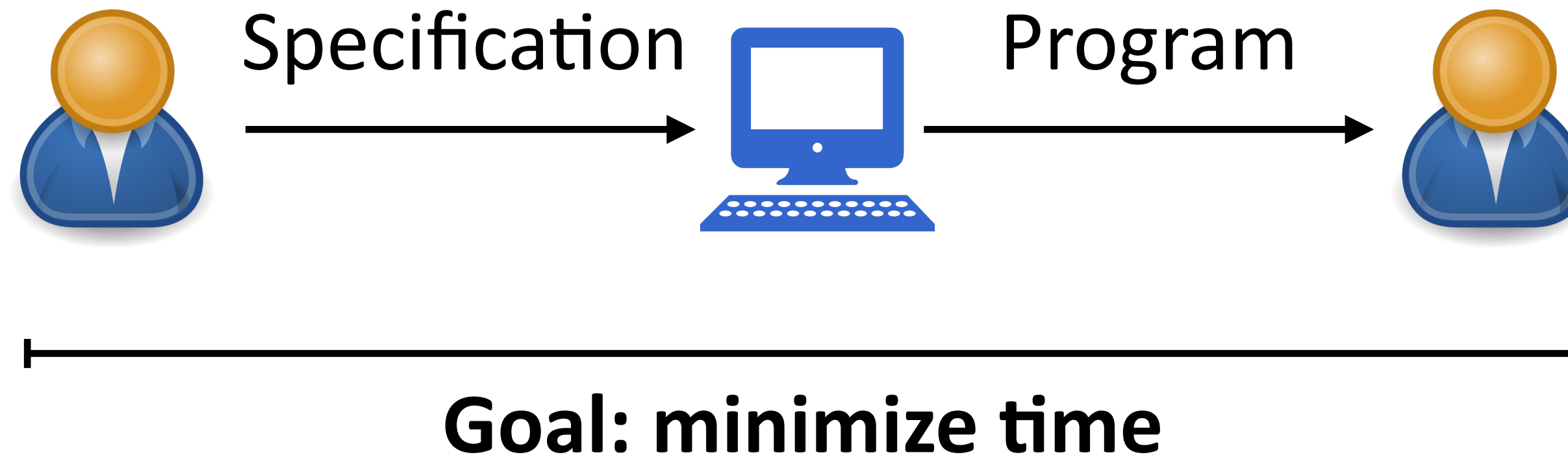
- **Live**, remote discussion 3-4pm Friday (tomorrow)
 - Zoom link on course website
 - Discuss A3
- Paper presentation assignment out by midnight today
 - 15 people submitted preferences
 - 12 slots

So Far...



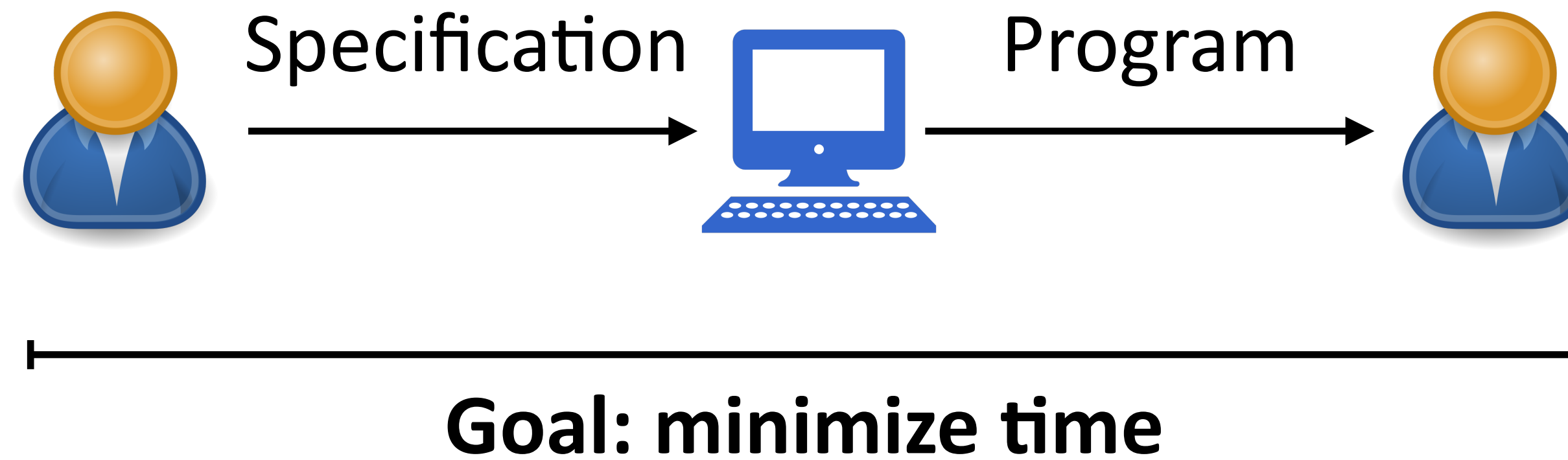
- Setup: given spec, find a program that satisfies spec

So Far...



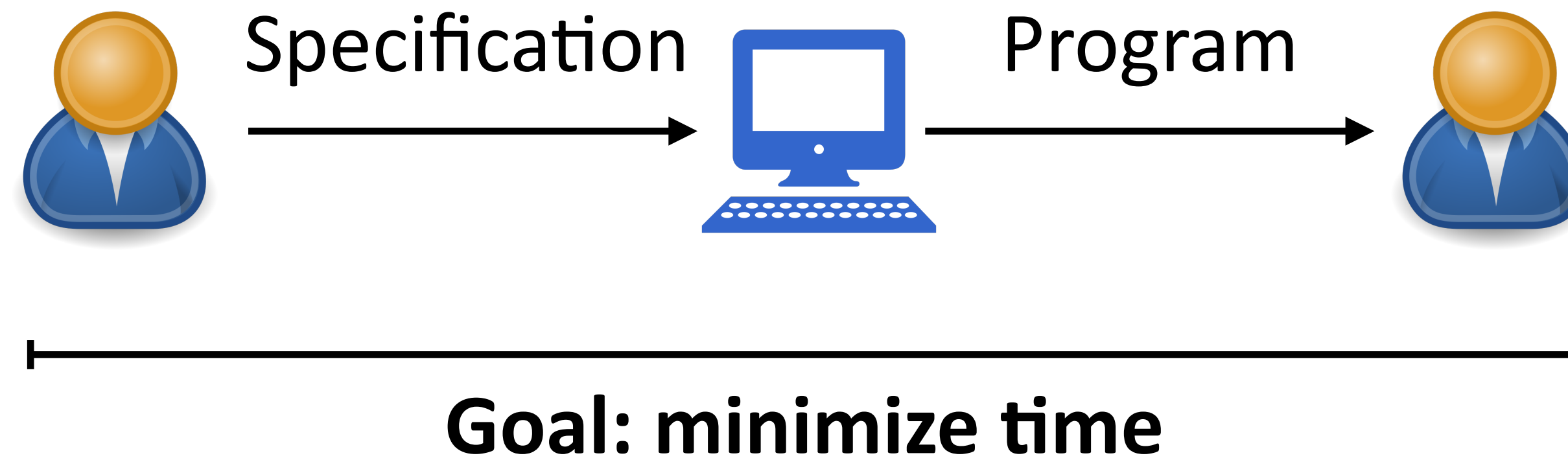
- Setup: given spec, find a program that satisfies spec
- Problems we've looked at:
 - Search space: how to define search space?
 - Search: how to find programs that satisfy specification?
 - Efficiency: How to efficiently search programs?

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 \neq 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

- Ranking (inductive bias)
- 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)
- **Interaction**
- 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.

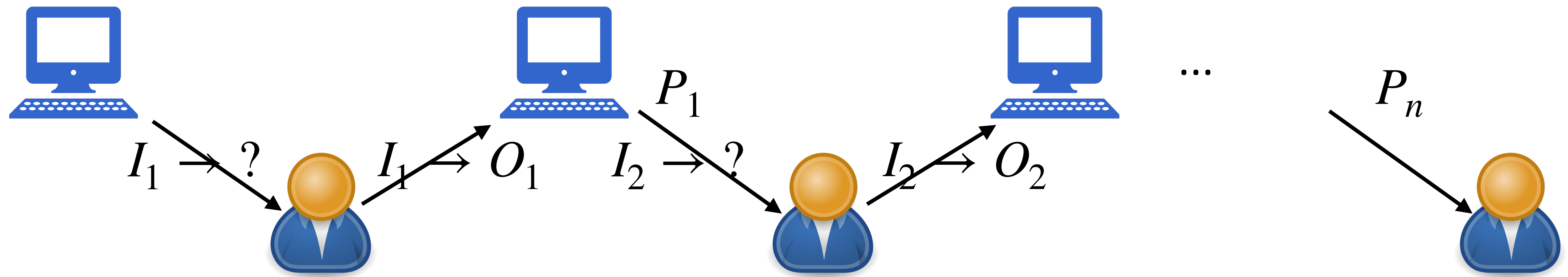
How To Pick “Good” Specifications?

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* **Gustavo Soares†** **Maxim Grechkin‡** **Vu Le§**
mikael.mayer@epfl.ch gsoares@dsc.ufcg.edu.br grechkin@cs.washington.edu vmle@ucdavis.edu

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

Mark Marron¶ **Oleksandr Polozov‡** **Rishabh Singh¶** **Benjamin Zorn¶** **Sumit Gulwani¶**
marron polozov@cs.washington.edu risin zorn 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

Add examples

The screenshot shows the FlashProg interface with the following components:

- Header:** "FlashProg" logo, "Interactive data extraction platform", and a yellow box with the number "1".
- Toolbar:** Buttons for "Add Input" (+), "Reset" (trash), "Undo" (left arrow), "Redo" (right arrow), and "Result" (down arrow).
- Left Panel:** A list of contact records with colored boxes highlighting specific fields. A yellow box with the number "2" is next to the record for Thomas Hardy.
- Right Panel:** Tabs for "Output", "Program viewer", and "Disambiguation". A yellow box with the number "3" is in the center of this panel.
- Legend:** A legend at the top left of the left panel shows colored boxes for "Name" (green), "City" (orange), "Phone number" (yellow), and "Label4" (pink).

Record 1: Ana Trujillo (Name), 357 21th Place SE (Address), Redmond, WA (City), (757) 555-1634 (Phone number)

Record 2: Antonio Moreno (Name), 515 93th Lane (Address), Renton, WA (City), (411) 555-2786 (Phone number)

Record 3: Thomas Hardy (Name), 742 17th Street NE (Address), Seattle, WA (City), (412) 555-5719 (Phone number)

Record 4: Christina Berglund (Name), 475 22th Lane (Address), Redmond, WA (City), (443) 555-6774 (Phone number)

Record 5: Hanna Moos (Name), 785 45th Street NE (Address), Puvallun WA (City)

FlashProg System

Input text

The screenshot displays the FlashProg system interface. At the top, the logo "FlashProg" is followed by a yellow box containing the number "1". To the right are six icons: a plus sign (Add Input), a trash can (Reset), a left arrow (Undo), a right arrow (Redo), and a downward arrow (Result). Below the header, there are four colored boxes (green, orange, yellow, pink) labeled "Name", "City", "Phone number", and "Label4". A list of contact information is shown, with a yellow box containing the number "2" next to it. The list includes: Ana Trujillo (357 21th Place SE, Redmond, WA, (757) 555-1634), Antonio Moreno (515 93th Lane, Renton, WA, (411) 555-2786), Thomas Hardy (742 17th Street NE, Seattle, WA, (412) 555-5719), Christina Berglund (475 22th Lane, Redmond, WA, (443) 555-6774), and Hanna Moos (785 45th Street NE, Puvallun WA). To the right of the list is a vertical bar with horizontal lines. Further right, there are three tabs: "Output", "Program viewer", and "Disambiguation". A yellow box containing the number "3" is positioned in the center of the "Output" tab area.

FlashProg System

The screenshot displays the FlashProg system interface. At the top, the logo "FlashProg" is followed by a yellow box containing the number "1". Below the logo is the text "Interactive data extraction platform". To the right of the logo are six icons: a plus sign labeled "Add Input", a trash can labeled "Reset", a left arrow labeled "Undo", a right arrow labeled "Redo", and a downward arrow labeled "Result".

Below the top bar, there are three tabs: "Output", "Program viewer", and "Disambiguation". The "Output" tab is active. On the left side, there are four colored boxes (green, orange, yellow, pink) with labels "Name", "City", "Phone number", and "Label4" respectively. Below these are five rows of contact information, each with a corresponding colored bar to its right. A yellow box with the number "2" is positioned to the right of the second row.

On the right side, a yellow box with the number "3" is followed by the text "Extracted data".

Name	City	Phone number	Label4
Ana Trujillo	Redmond, WA	(757) 555-1634	
Antonio Moreno	Renton, WA	(411) 555-2786	
Thomas Hardy	Seattle, WA	(412) 555-5719	
Christina Berglund	Redmond, WA	(443) 555-6774	
Hanna Moos	Puyallup, WA		

FlashProg System

The screenshot displays the FlashProg system interface. At the top, the logo "FlashProg" is shown in blue and orange, with the tagline "Interactive data extraction platform" below it. To the right of the logo are five icons: a plus sign for "Add Input", a trash can for "Reset", a left arrow for "Undo", a right arrow for "Redo", and a downward arrow for "Result".

Below the header, there are three tabs: "Output" (selected), "Program viewer", and "Disambiguation".

On the left side, there are four colored boxes representing data fields: a green box for "Name", an orange box for "City", a yellow box for "Phone number", and a pink box with a plus sign for "Label4". Below these are five rows of data, each with a colored bar representing the field extraction. The data is as follows:

Name	City	Phone number
Ana Trujillo	Redmond, WA	(757) 555-1634
Antonio Moreno	Renton, WA	(411) 555-2786
Thomas Hardy	Seattle, WA	(412) 555-5719
Christina Berglund	Redmond, WA	(443) 555-6774
Hanna Moos	Seattle, WA	(412) 555-5719

On the right side, the "Output" tab is active, showing a table with 22 rows. The table has three columns: "Name", "City", and "Phone number". The data is as follows:

Name	City	Phone number
Ana Trujillo	Redmond	(757) 555-1634
Antonio Moreno	Renton	(411) 555-2786
Thomas Hardy	Seattle	(412) 555-5719
Christina Berglund	Redmond	(443) 555-6774
Hanna Moos	Puyallup	(376) 555-2462
Frederique Citeaux	Redmond	(689) 555-2770
Martin Sommer	Kent	(715) 555-5450
Laurence Lebihan	Redmond	(620) 555-2361
Elizabeth Lincoln	Renton	(851) 555-4561
Victoria Ashworth	Renton	(696) 555-6044
Patricio Simpson	Redmond	(179) 555-3265
Francisco Chang	Seattle	(272) 555-7434

Output Preview

FlashProg System

FlashProg
Interactive data extraction platform

Buttons: Add Input, Reset, Undo, Redo, Result

Legend: Name (Green), City (Orange), Phone number (Yellow), Label4 (Pink)

Output | **Program viewer**

Re-learn

To extract Name-struct from the whole text:
Take the boundaries of **Name** defined below.
To extract the sequence **Name** from the whole text:
Extract ▷ all lines starting with
Words/dots/hyphens◦WhiteSpace

To extract City-struct from Name-struct:
Take the boundaries of **City** defined below.
To extract **City** from Name-struct:
From the substring starting at the ▷
beginning of the third line, extract the
string ending at
| the first occurrence of Comma

To extract **Phone number** from City-struct:
extract the second line


Records:


- Ana Trujillo**
357 21th Place SE
Redmond, WA
(757) 555-1634
- Antonio Moreno**
515 93th Lane
Renton, WA
(411) 555-2786
- Thomas Hardy**
742 17th Street NE
Seattle, WA
(412) 555-5719
- Christina Berglund**
475 22th Lane
Redmond, WA
(443) 555-6774
- Hanna Moos**
785 45th Street NE

**Synthesized
Program**

FlashProg System

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


Label1


Label2



[1] A. Ahmed, A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan, and D. C. Wang. Semantic foundations for typed assembly languages. ACM Trans. Program. Lang. Syst., 32(3), 2010.

[2] A. W. Appel. Program Logics for Certified Compilers. Cambridge University Press, 2014.

[3] A. W. Appel and S. Blazy. Separation logic for small-step Cminor. In TPHOLs, volume 4732 of LNCS, pages 5–21. Springer, 2007.

FlashProg System

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



Label1 Label2



[1] A. Ahmed, A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan, and D. C. Wang. Semantic foundations for typed assembly languages. ACM Trans. Program. Lang. Syst., 32(3), 2010.

[2] A. W. Appel. Program Logics for Certified Compilers. Cambridge University Press, 2014.

[3] A. W. Appel and S. Blazy. Separation logic for small-step Cminor. In TPHOLs, volume 4732 of LNCS, pages 5–21. Springer, 2007.

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



Label1 Label2

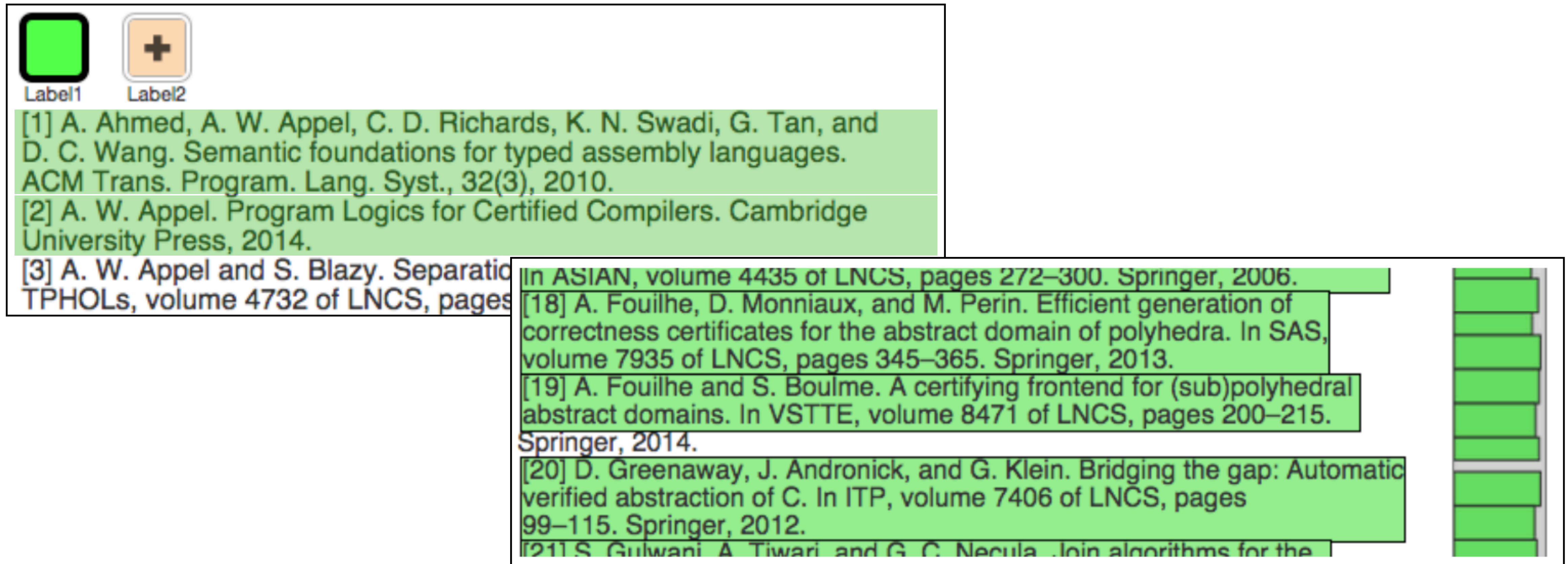
[1] A. Ahmed, A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan, and D. C. Wang. Semantic foundations for typed assembly languages. ACM Trans. Program. Lang. Syst., 32(3), 2010.

[2] A. W. Appel. Program Logics for Certified Compilers. Cambridge University Press, 2014.

[3] A. W. Appel and S. Blazy. Separation logic for small-step Cminor. In TPHOLs, volume 4732 of LNCS, pages 5–21. Springer, 2007.

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



The screenshot shows the FlashProg interface. At the top left, there are two icons: a green square labeled 'Label1' and an orange square with a plus sign labeled 'Label2'. Below these icons is a list of papers. The first three papers are highlighted in green, indicating they are examples. The remaining papers are not highlighted. On the right side of the interface, there is a vertical bar with several green segments, representing a progress or selection indicator.

Label1 Label2

[1] A. Ahmed, A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan, and D. C. Wang. Semantic foundations for typed assembly languages. ACM Trans. Program. Lang. Syst., 32(3), 2010.

[2] A. W. Appel. Program Logics for Certified Compilers. Cambridge University Press, 2014.

[3] A. W. Appel and S. Blazy. Separation logic for pointers. In ASIAC, volume 4435 of LNCS, pages 272–300. Springer, 2006.

[18] A. Foulhe, D. Monniaux, and M. Perin. Efficient generation of correctness certificates for the abstract domain of polyhedra. In SAS, volume 7935 of LNCS, pages 345–365. Springer, 2013.

[19] A. Foulhe and S. Boulme. A certifying frontend for (sub)polyhedral abstract domains. In VSTTE, volume 8471 of LNCS, pages 200–215. Springer, 2014.

[20] D. Greenaway, J. Andronick, and G. Klein. Bridging the gap: Automatic verified abstraction of C. In ITP, volume 7406 of LNCS, pages 99–115. Springer, 2012.

[21] S. Gulwani, A. Tiwari, and G. C. Necula. Join algorithms for the

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

The screenshot shows the FlashProg interface. At the top left, there are two icons: a green square labeled 'Label1' and an orange square with a plus sign labeled 'Label2'. Below these icons is a list of papers, each highlighted in green. The papers are:

- [1] A. Ahmed, A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan, and D. C. Wang. Semantic foundations for typed assembly languages. ACM Trans. Program. Lang. Syst., 32(3), 2010.
- [2] A. W. Appel. Program Logics for Certified Compilers. Cambridge University Press, 2014.
- [3] A. W. Appel and S. Blazy. Separation logic for mutable pointers. In ASIAC, volume 4435 of LNCS, pages 272–300. Springer, 2006.
- [18] A. Foulhe, D. Monniaux, and M. Perin. Efficient generation of correctness certificates for the abstract domain of polyhedra. In SAS, volume 7935 of LNCS, pages 345–365. Springer, 2013.
- [19] A. Foulhe and S. Boulme. A certifying frontend for (sub)polyhedral abstract domains. In VSTTE, volume 8471 of LNCS, pages 200–215. Springer, 2014.
- [20] D. Greenaway, J. Andronick, and G. Klein. Bridging the gap: Automatic verified abstraction of C. In ITP, volume 7406 of LNCS, pages 99–115. Springer, 2012.
- [21] S. Gulwani, A. Tiwari, and G. C. Necula. Join algorithms for the

To the right of the paper list, there is a vertical stack of green rectangular boxes, representing a list of authors. The text 'Wrong program' is written in red to the right of the interface, indicating an error.

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

Label1 Label2

[1] A. Ahmed, A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan, and D. C. Wang. Semantic foundations for typed assembly languages. ACM Trans. Program. Lang. Syst., 32(3), 2010.

[2] A. W. Appel. Program Logics for Certified Compilers. Cambridge University Press, 2014.

[3] A. W. Appel and S. Blazy. Separation logic for mutable pointers. In ASIAN, volume 4435 of LNCS, pages 272–300. Springer, 2006.

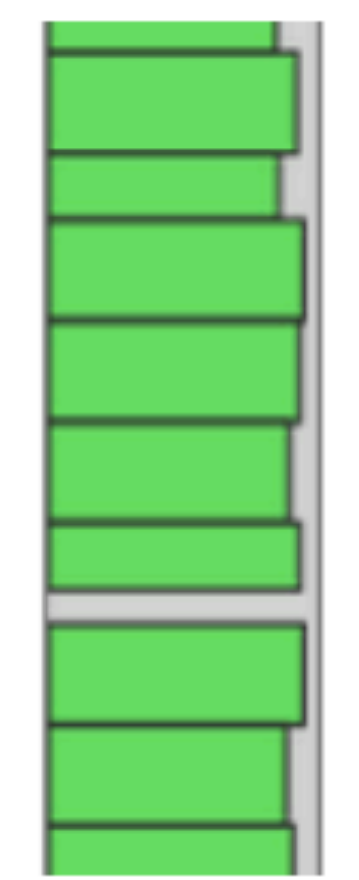
Fix: provide another example
Wrong program

[18] A. Foulhe, D. Monniaux, and M. Perin. Efficient generation of correctness certificates for the abstract domain of polyhedra. In SAS, volume 7935 of LNCS, pages 345–365. Springer, 2013.

[19] A. Foulhe and S. Boulme. A certifying frontend for (sub)polyhedral abstract domains. In VSTTE, volume 8471 of LNCS, pages 200–215. Springer, 2014.

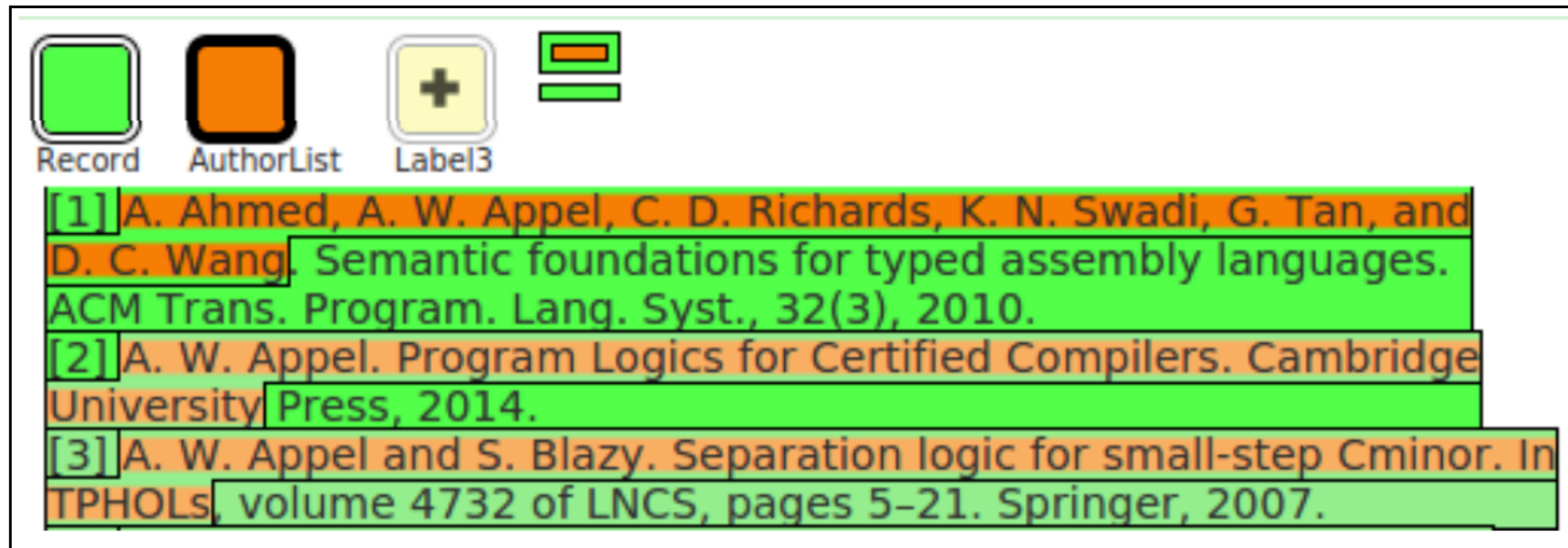
[20] D. Greenaway, J. Andronick, and G. Klein. Bridging the gap: Automatic verified abstraction of C. In ITP, volume 7406 of LNCS, pages 99–115. Springer, 2012.

[21] S. Gulwani, A. Tiwari, and G. C. Necula. Join algorithms for the



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



The screenshot shows the FlashProg interface. At the top, there is a toolbar with four icons: a green square labeled 'Record', an orange square labeled 'AuthorList', a yellow square with a plus sign labeled 'Label3', and a green square with a horizontal line. Below the toolbar, there is a list of three papers. The author names in each paper are highlighted in orange, and the rest of the text is highlighted in green.

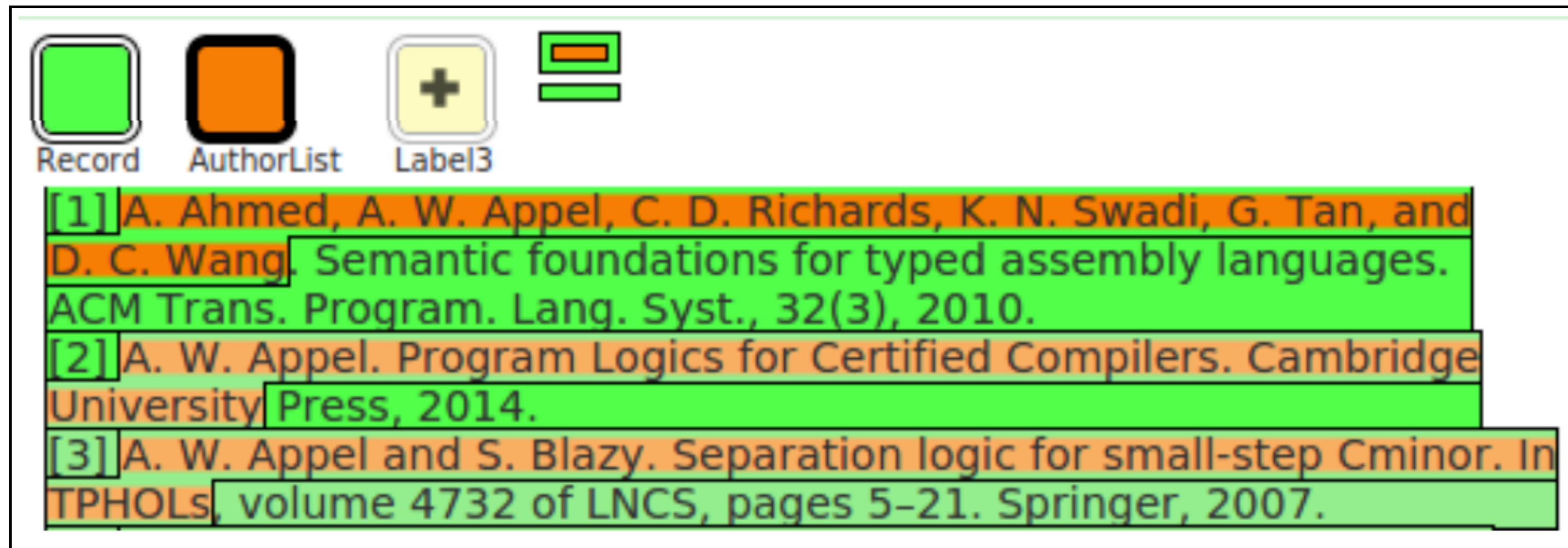
[1] A. Ahmed, A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan, and D. C. Wang. Semantic foundations for typed assembly languages. ACM Trans. Program. Lang. Syst., 32(3), 2010.

[2] A. W. Appel. Program Logics for Certified Compilers. Cambridge University Press, 2014.

[3] A. W. Appel and S. Blazy. Separation logic for small-step Cminor. In TPHOLs, volume 4732 of LNCS, pages 5–21. Springer, 2007.

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



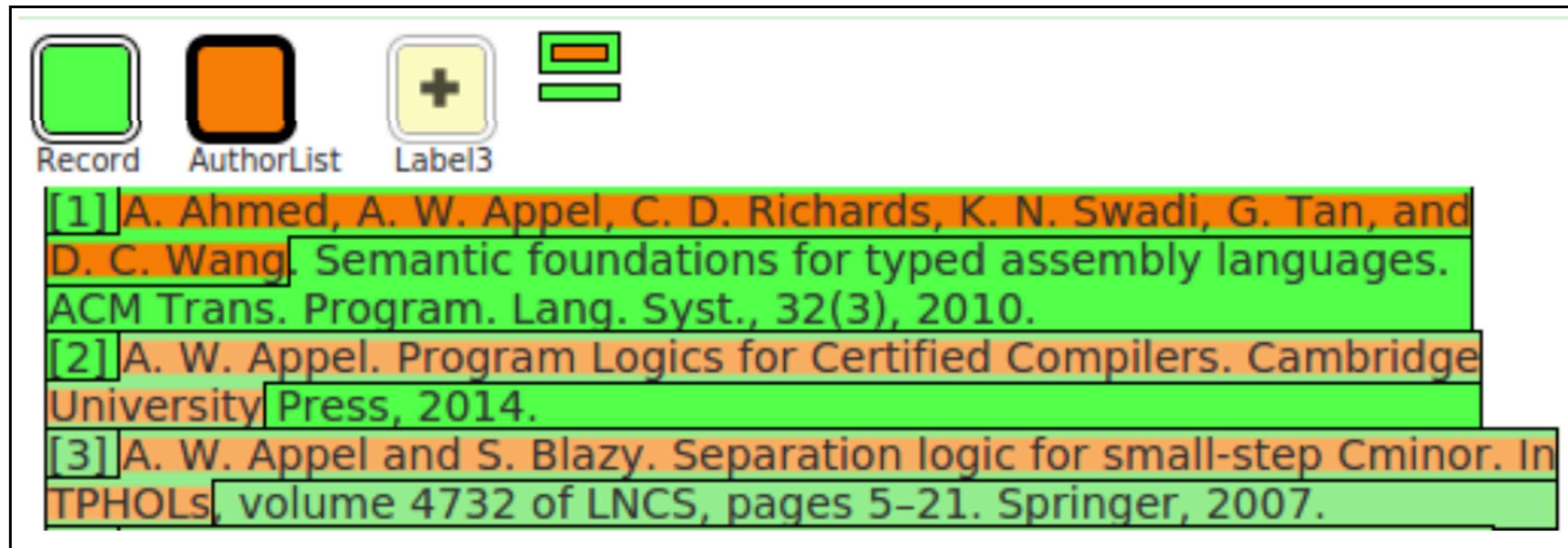
The screenshot shows the FlashProg interface with a toolbar at the top containing four icons: a green square labeled 'Record', an orange square labeled 'AuthorList', a yellow square with a plus sign labeled 'Label3', and a green rectangle with an orange bar. Below the toolbar, three paper entries are displayed. Each entry has its author list highlighted in orange and its title in green. The entries are:

- [1] A. Ahmed, A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan, and D. C. Wang. Semantic foundations for typed assembly languages. ACM Trans. Program. Lang. Syst., 32(3), 2010.
- [2] A. W. Appel. Program Logics for Certified Compilers. Cambridge University Press, 2014.
- [3] A. W. Appel and S. Blazy. Separation logic for small-step Cminor. In TPHOLS, volume 4732 of LNCS, pages 5–21. Springer, 2007.

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



The screenshot shows the FlashProg interface with a toolbar at the top containing four icons: a green square labeled 'Record', an orange square labeled 'AuthorList', a yellow square with a plus sign labeled 'Label3', and a green rectangle with an orange bar. Below the toolbar, a list of three papers is displayed. The author names in each paper are highlighted in orange, while the rest of the text is highlighted in green. The papers are:

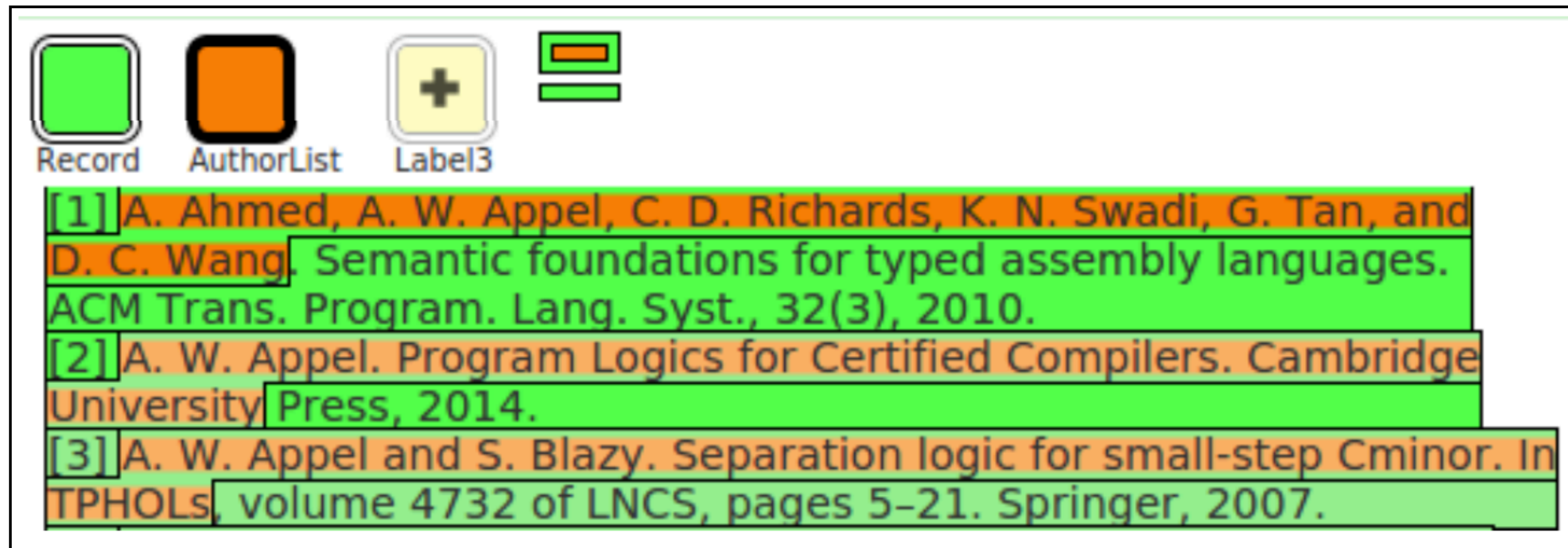
- [1] A. Ahmed, A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan, and D. C. Wang. Semantic foundations for typed assembly languages. ACM Trans. Program. Lang. Syst., 32(3), 2010.
- [2] A. W. Appel. Program Logics for Certified Compilers. Cambridge University Press, 2014.
- [3] A. W. Appel and S. Blazy. Separation logic for small-step Cminor. In TPHOLS, volume 4732 of LNCS, pages 5–21. Springer, 2007.

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



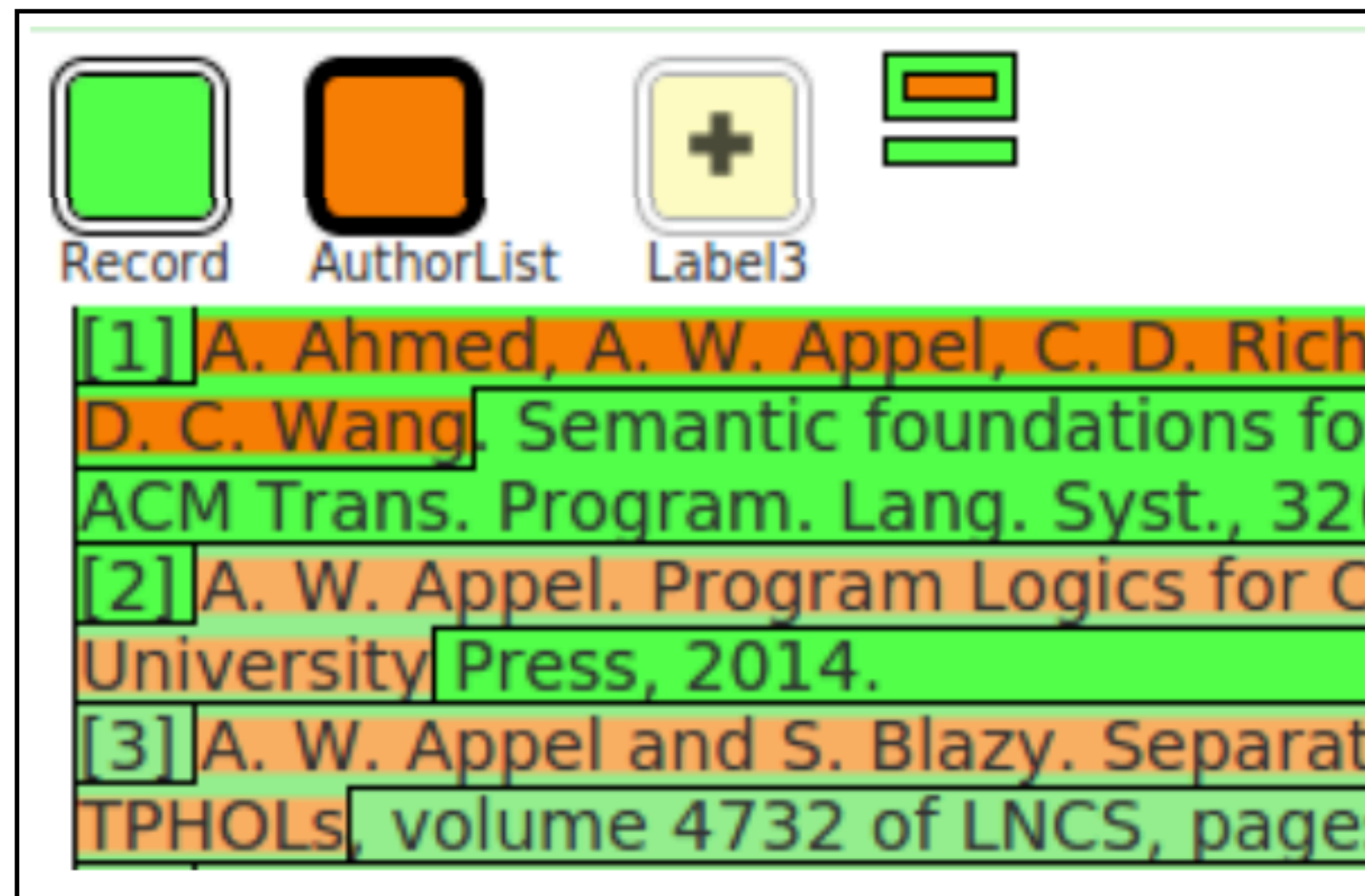
Wrong program

Fix: provide another example?

Or, choose a different program?

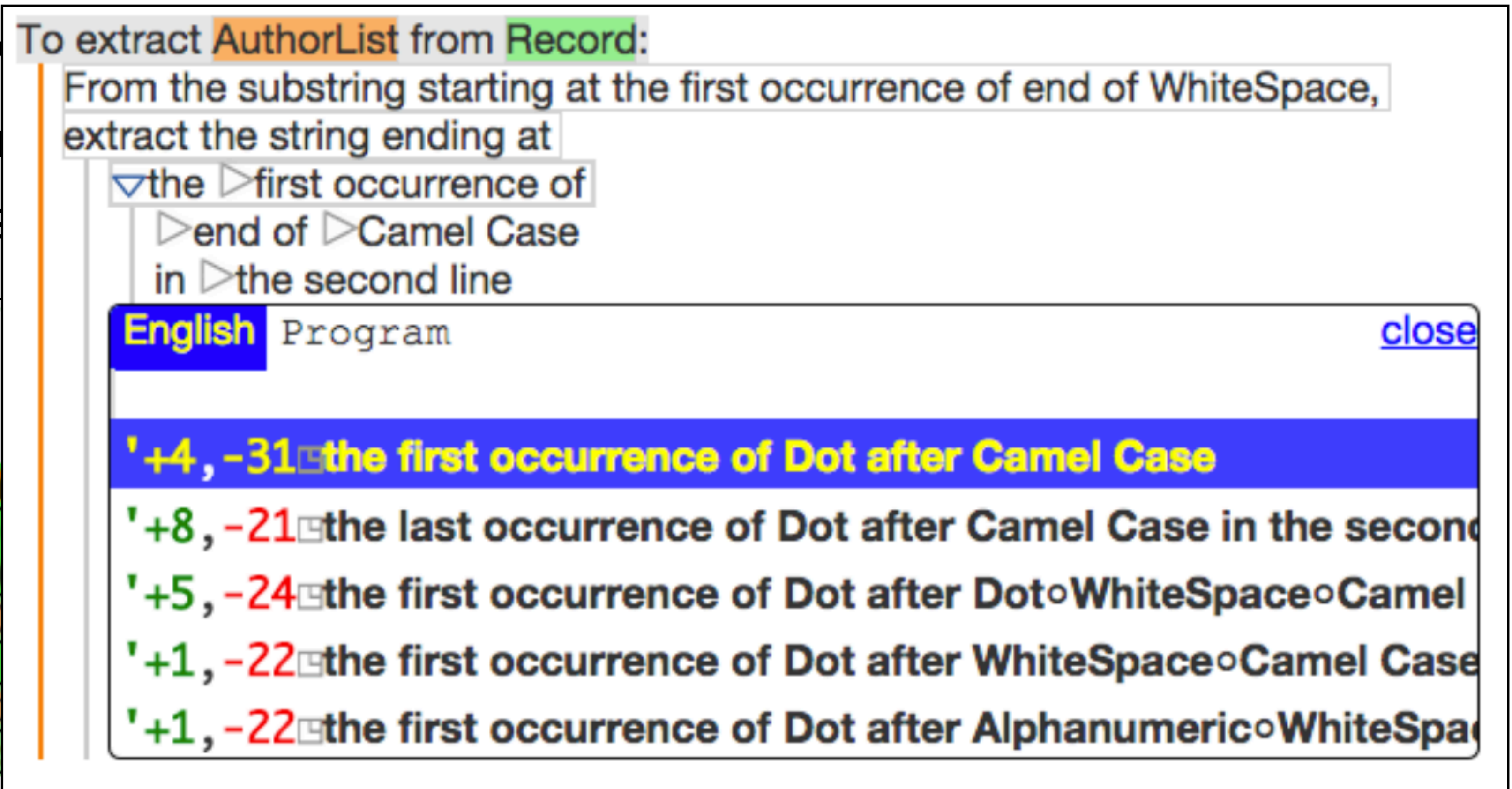
FlashProg System

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



The screenshot shows the FlashProg interface with a list of papers. The interface includes a toolbar with icons for Record (green square), AuthorList (orange square), Label3 (yellow square with a plus sign), and a green bar icon. The list of papers is as follows:

- [1] A. Ahmed, A. W. Appel, C. D. Rich, D. C. Wang. Semantic foundations for...
- [2] A. W. Appel. Program Logics for C...
- [3] A. W. Appel and S. Blazy. Separat...



The screenshot shows the 'English Program' editor with a rule for extracting author lists from a record. The rule is:

```
To extract AuthorList from Record:  
From the substring starting at the first occurrence of end of WhiteSpace,  
extract the string ending at  
  ▾the ▾first occurrence of  
    ▾end of ▾Camel Case  
    in ▾the second line
```

The editor also shows a list of rules for extracting author names:

- ' +4, -31 the first occurrence of Dot after Camel Case
- ' +8, -21 the last occurrence of Dot after Camel Case in the second
- ' +5, -24 the first occurrence of Dot after Dot◦WhiteSpace◦Camel
- ' +1, -22 the first occurrence of Dot after WhiteSpace◦Camel Case
- ' +1, -22 the first occurrence of Dot after Alphanumeric◦WhiteSpac

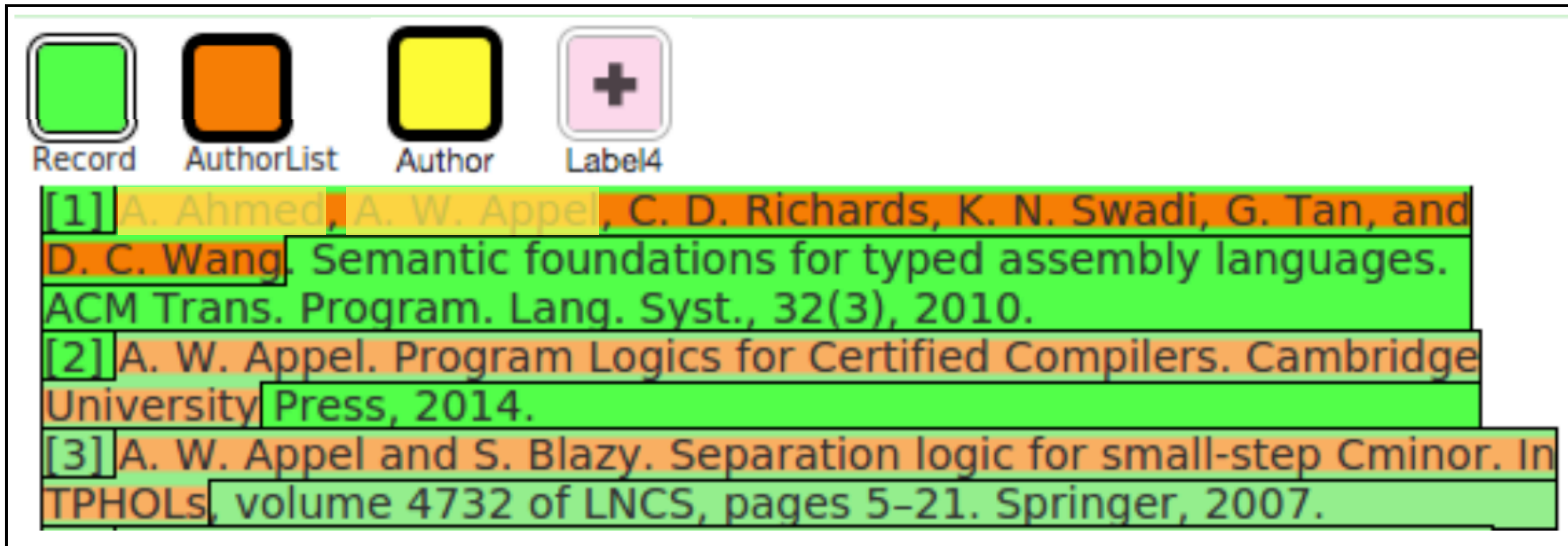
Wrong program

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

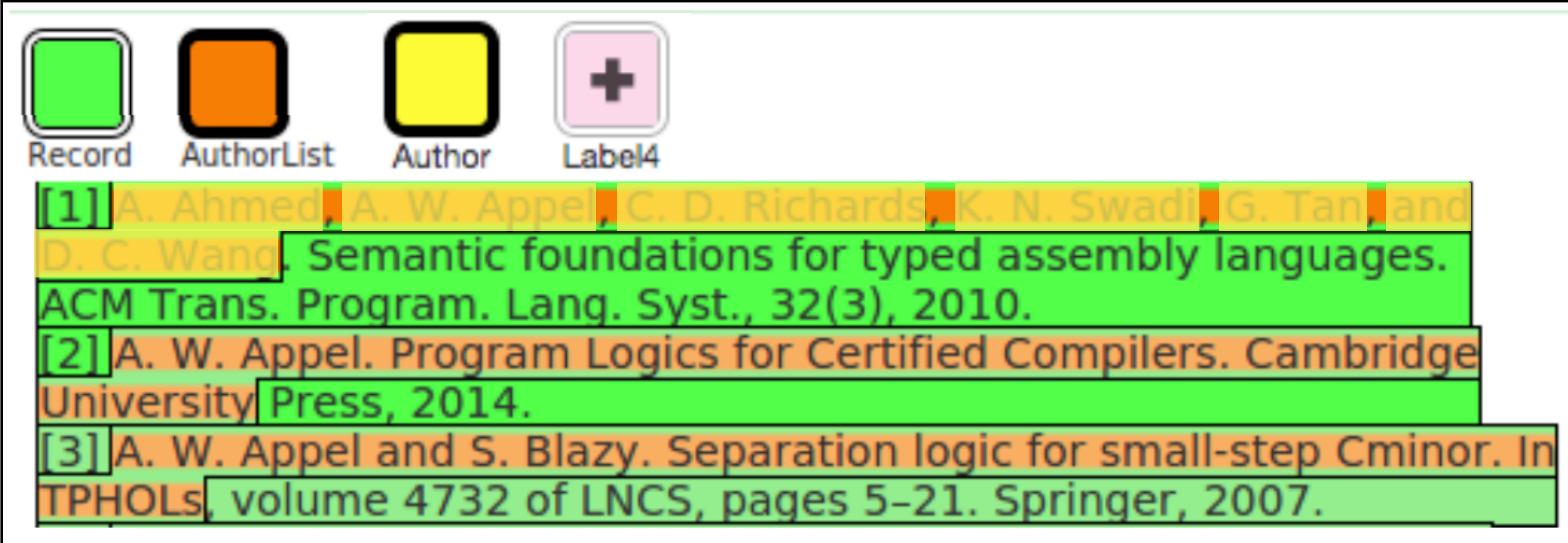


The screenshot displays the FlashProg interface. At the top, there is a legend with four colored boxes: a green box labeled 'Record', an orange box labeled 'AuthorList', a yellow box labeled 'Author', and a pink box with a plus sign labeled 'Label4'. Below the legend, three lines of text represent a list of papers. Each line is highlighted with a green background, and the author list portion of each line is highlighted with an orange background. The papers are:

- [1] A. Ahmed, A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan, and D. C. Wang. Semantic foundations for typed assembly languages. ACM Trans. Program. Lang. Syst., 32(3), 2010.
- [2] A. W. Appel. Program Logics for Certified Compilers. Cambridge University Press, 2014.
- [3] A. W. Appel and S. Blazy. Separation logic for small-step Cminor. In TPHOLs, volume 4732 of LNCS, pages 5-21. Springer, 2007.

FlashProg System

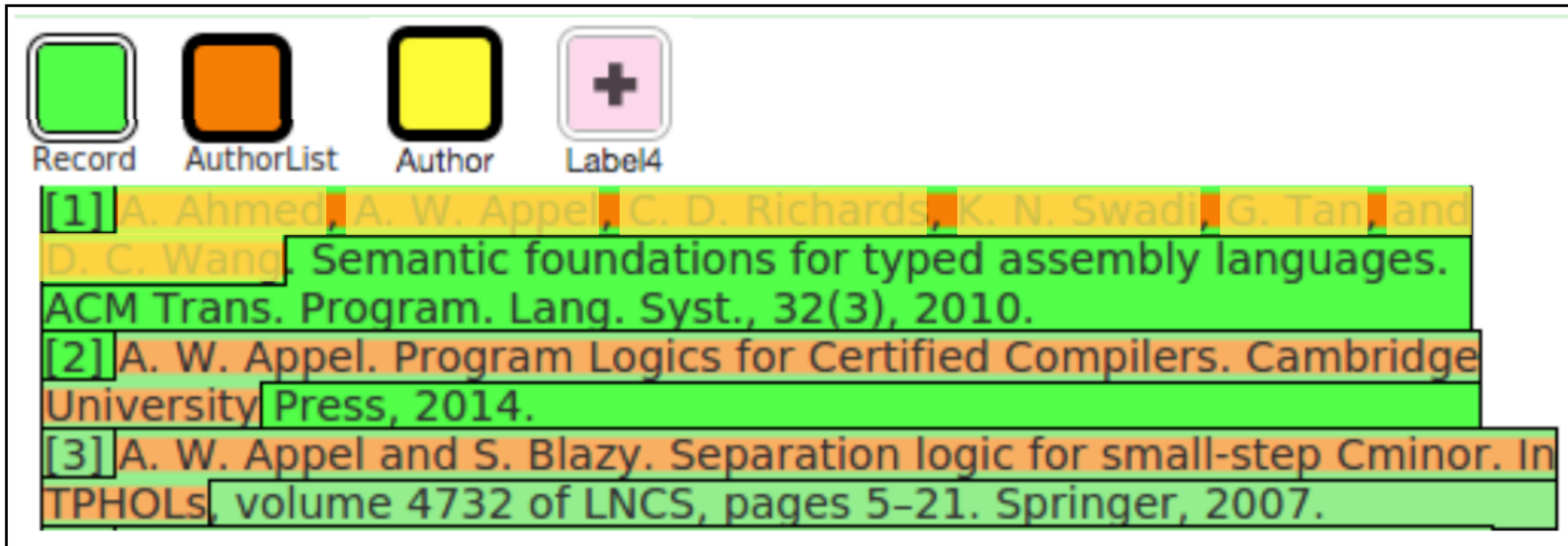
- Consider task: given a list of papers, extract all authors
- FlashProg requires user to do this in steps
 - First, extract papers using examples
 - Next, extract author lists using examples
 - Finally, extract individual authors using examples



The screenshot displays the FlashProg interface. At the top, there are four icons: a green square labeled 'Record', an orange square labeled 'AuthorList', a yellow square labeled 'Author', and a pink square with a plus sign labeled 'Label4'. Below these icons, three lines of text represent paper records. Each record is highlighted with a yellow background. The first record is '[1] A. Ahmed, A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan, and D. C. Wang. Semantic foundations for typed assembly languages. ACM Trans. Program. Lang. Syst., 32(3), 2010.' The second record is '[2] A. W. Appel. Program Logics for Certified Compilers. Cambridge University Press, 2014.' The third record is '[3] A. W. Appel and S. Blazy. Separation logic for small-step Cminor. In TPHOLs, volume 4732 of LNCS, pages 5-21. Springer, 2007.'

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

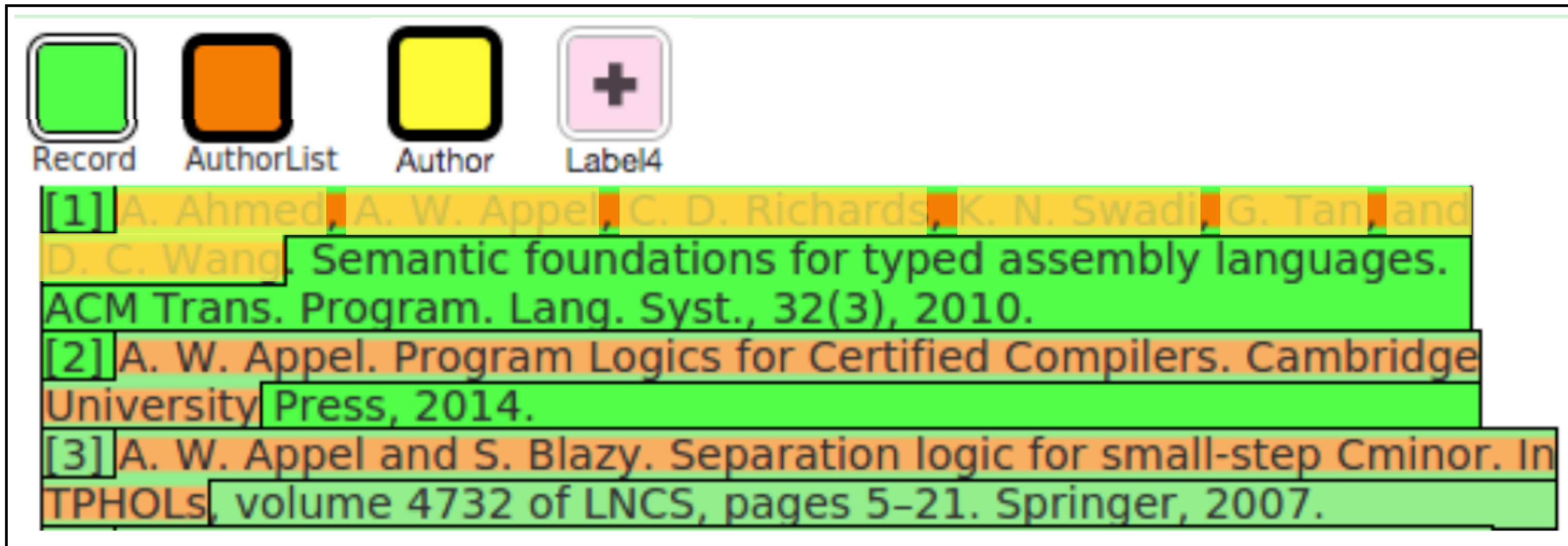


The screenshot shows the FlashProg interface. At the top, there is a legend with four colored boxes: a green box labeled 'Record', an orange box labeled 'AuthorList', a yellow box labeled 'Author', and a pink box with a plus sign labeled 'Label4'. Below the legend, there are three lines of text representing paper entries. Each entry is highlighted with a yellow background. The first entry is '[1] A. Ahmed, A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan, and D. C. Wang. Semantic foundations for typed assembly languages. ACM Trans. Program. Lang. Syst., 32(3), 2010.' The second entry is '[2] A. W. Appel. Program Logics for Certified Compilers. Cambridge University Press, 2014.' The third entry is '[3] A. W. Appel and S. Blazy. Separation logic for small-step Cminor. In TPHOLs, volume 4732 of LNCS, pages 5-21. Springer, 2007.'

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



The screenshot shows the FlashProg interface with a legend at the top and a list of three papers below. The legend consists of four colored squares with labels: a green square for 'Record', an orange square for 'AuthorList', a yellow square for 'Author', and a pink square with a plus sign for 'Label4'. The list of papers is as follows:

- [1] A. Ahmed, A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan, and D. C. Wang. Semantic foundations for typed assembly languages. ACM Trans. Program. Lang. Syst., 32(3), 2010.
- [2] A. W. Appel. Program Logics for Certified Compilers. Cambridge University Press, 2014.
- [3] A. W. Appel and S. Blazy. Separation logic for small-step Cminor. In TPHOLs, volume 4732 of LNCS, pages 5-21. Springer, 2007.

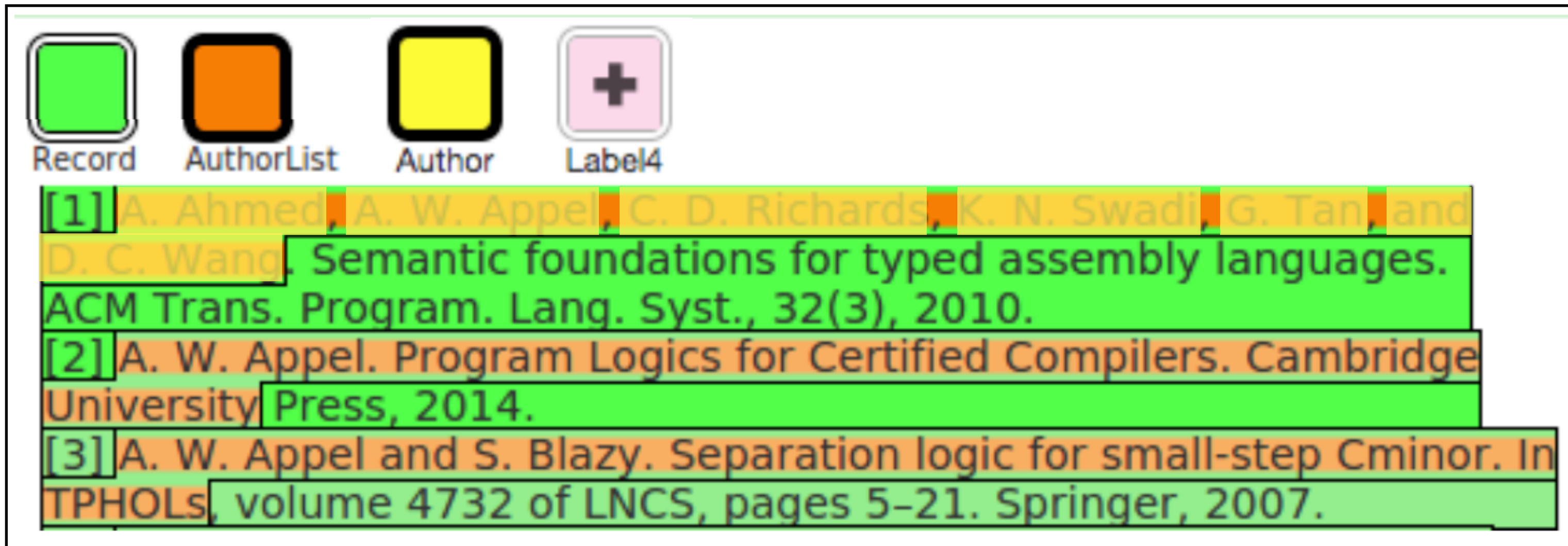
Wrong program

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



The screenshot shows the FlashProg interface with a toolbar at the top containing four icons: a green square labeled 'Record', an orange square labeled 'AuthorList', a yellow square labeled 'Author', and a pink square with a plus sign labeled 'Label4'. Below the toolbar, three paper entries are displayed, each with a numbered label in a green box and a highlighted author list in an orange box. The first entry is labeled '[1]' and has authors 'A. Ahmed, A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan, and D. C. Wang'. The second entry is labeled '[2]' and has author 'A. W. Appel'. The third entry is labeled '[3]' and has authors 'A. W. Appel and S. Blazy'. The text of the papers is highlighted in green.

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 steps
 - First, extract papers using examples
 - Next, extract author lists using examples
 - Finally, extract individual authors using examples

Output Program viewer **Disambiguation**

'Author' is currently ambiguous. Which highlighting is correct?

and D. C. Wang or and D. C. Wang

Let me edit it myself

Record AuthorList Author Label4

[1] A. Ahmed, A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan, and D. C. Wang. Semantic foundations for typed assembly languages. ACM Trans. Program. Lang. Syst., 32(3), 2010.

[2] A. W. Appel. Program Logics for Certified Compilers. Cambridge University Press, 2014.

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)*

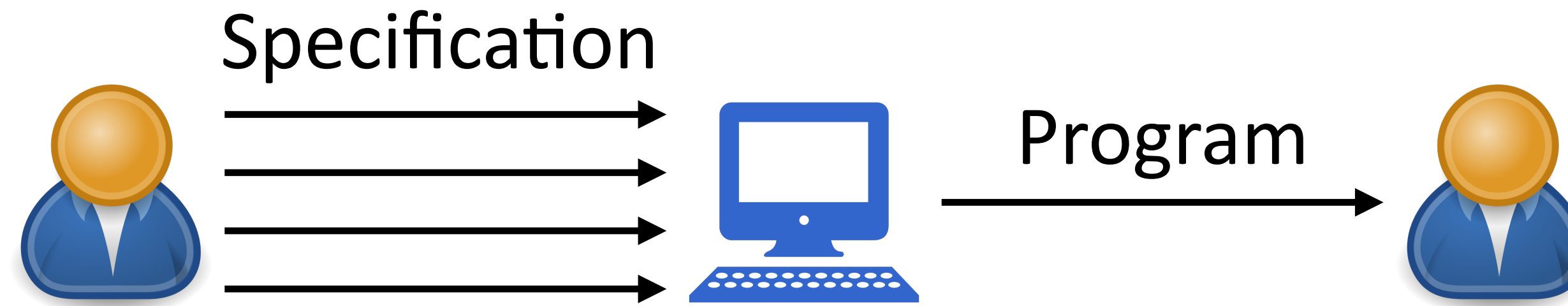
Question Selection for Interactive Program Synthesis

- 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
- **Multi-modality**

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)

Learner

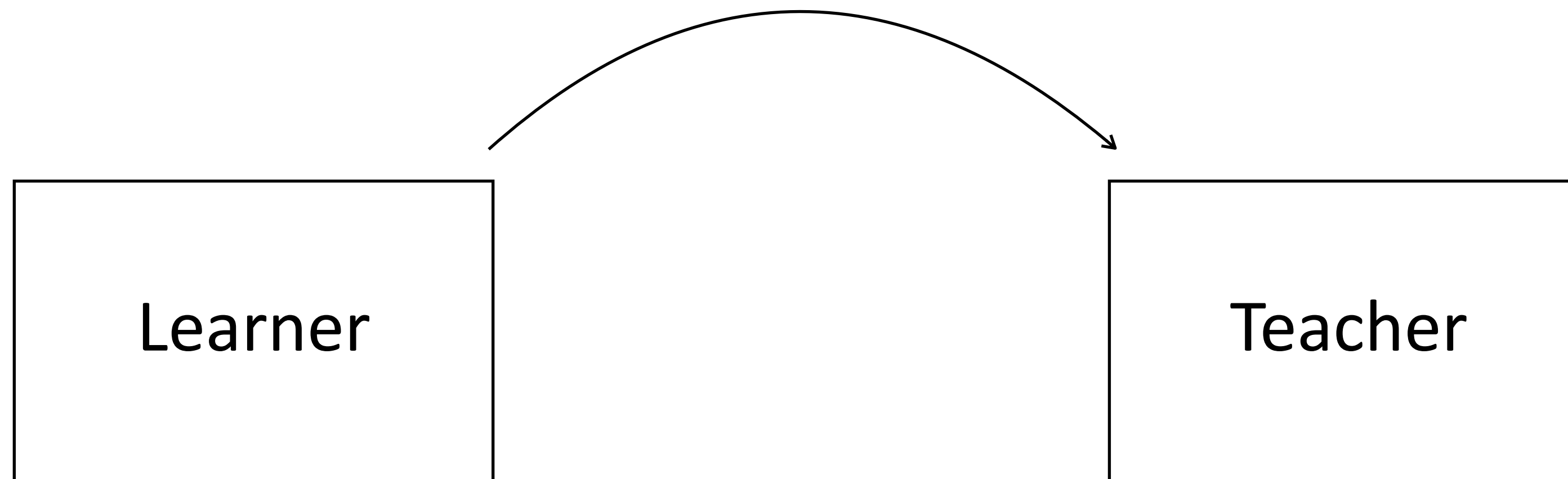
Teacher

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?

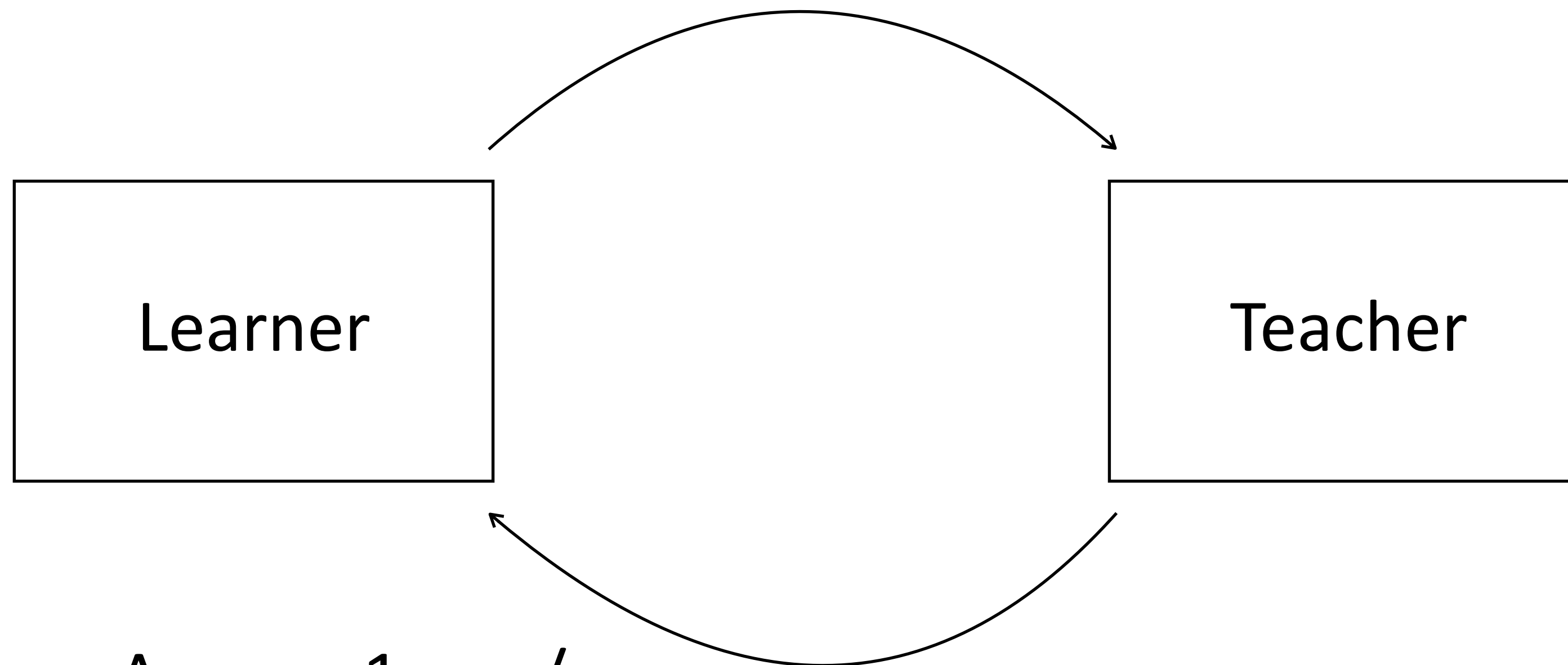


L* Algorithm

- 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-trained Language 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

The screenshot shows a web-based regular expression testing interface. At the top, the text "REGULAR EXPRESSION" is displayed above a text input field containing the pattern `a(.)*b`. Below this, the text "TEST STRING" is displayed above a list of test strings. Each string is shown with a blue highlight under the 'a' and a green highlight under the 'b', indicating a successful match. The test strings are: `a`, `ab`, `acb`, `accb`, `acdb`, `acdeb`, `a1234b`, and `a.*b`. A cursor is visible at the end of each string.

Regular Expressions 101

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

The screenshot shows a web-based regular expression testing interface. At the top, the text "REGULAR EXPRESSION" is displayed above a text input field containing the pattern `a(.)+b`. Below this, the text "TEST STRING" is displayed above a list of test strings. Each string is followed by a cursor icon. The strings and their visual highlights are: "a" (no highlight), "ab" (no highlight), "acb" (the 'c' is highlighted in green), "accb" (the 'c' is highlighted in green), "acdb" (the 'c' is highlighted in green), "acdeb" (the 'c' is highlighted in green), "a1234b" (the "1234" is highlighted in green), and "a.*b" (the "." is highlighted in green).

Regular Expressions 101

Anchors

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

Character Classes

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

Quantifiers

<code>*</code>	0 or more	<code>{3}</code>	Exactly 3
<code>+</code>	1 or more	<code>{3,}</code>	3 or more
<code>?</code>	0 or 1	<code>{3,5}</code>	3, 4 or 5

Add a `?` to a quantifier to make it ungreedy.

Escape Sequences

<code>\</code>	Escape following character
<code>\Q</code>	Begin literal sequence
<code>\E</code>	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

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

The escape character is usually `\`

Special Characters

Groups and Ranges

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

Ranges are inclusive.

Pattern Modifiers

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

* PCRE modifier

Regular Expressions 101

Anchors

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

Character Classes

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

Quantifiers

<code>*</code>	0 or more	<code>{3}</code>	Exactly 3
<code>+</code>	1 or more	<code>{3,}</code>	3 or more
<code>?</code>	0 or 1	<code>{3,5}</code>	3, 4 or 5

Add a `?` to a quantifier to make it ungreedy.

Escape Sequences

<code>\</code>	Escape following character
<code>\Q</code>	Begin literal sequence
<code>\E</code>	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

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

The escape character is usually `\`

Special Characters

Groups and Ranges

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

Ranges are inclusive.

Pattern Modifiers

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

* PCRE modifier

Regular Expressions 101

Anchors

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

Character Classes

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

Quantifiers

<code>*</code>	0 or more	<code>{3}</code>	Exactly 3
<code>+</code>	1 or more	<code>{3,}</code>	3 or more
<code>?</code>	0 or 1	<code>{3,5}</code>	3, 4 or 5

Add a `?` to a quantifier to make it ungreedy.

Escape Sequences

<code>\</code>	Escape following character
<code>\Q</code>	Begin literal sequence
<code>\E</code>	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

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

The escape character is usually `\`

Special Characters

Groups and Ranges

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

Ranges are inclusive.

Pattern Modifiers

<code>g</code>	Global match
<code>i *</code>	Case-insensitive
<code>m *</code>	Multiple lines
<code>s *</code>	Treat string as single line
<code>x *</code>	Allow comments and whitespace in pattern
<code>e *</code>	Evaluate replacement
<code>U *</code>	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`)
 - Kleene star and quantifiers

Regular Expressions 101

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

Regular Expressions 101

- Consider CFG (simplified):

$r ::= cc \mid \text{StartWith}(r) \mid \text{EndWith}(r) \mid \text{Contains}(r) \mid \text{Concat}(r, r) \mid \text{KleeneStar}(r) \mid \text{Repeat}(r, k)$

$cc ::= \langle \text{digit} \rangle \mid \langle \text{char} \rangle \mid \langle \text{any} \rangle \mid ..$

$k ::= 1 \mid 2 \mid ..$

Regular Expressions 101

- Consider CFG (simplified):

$r ::= cc \mid \text{StartWith}(r) \mid \text{EndWith}(r) \mid \text{Contains}(r) \mid \text{Concat}(r, r) \mid \text{KleeneStar}(r) \mid \text{Repeat}(r, k)$

$cc ::= \langle \text{digit} \rangle \mid \langle \text{char} \rangle \mid \langle \text{any} \rangle \mid ..$

$k ::= 1 \mid 2 \mid ..$

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

Regular Expressions 101

- Consider CFG (simplified):

$r ::= cc \mid \text{StartWith}(r) \mid \text{EndWith}(r) \mid \text{Contains}(r) \mid \text{Concat}(r, r) \mid \text{KleeneStar}(r) \mid \text{Repeat}(r, k)$

$cc ::= \langle \text{digit} \rangle \mid \langle \text{char} \rangle \mid \langle \text{any} \rangle \mid ..$

$k ::= 1 \mid 2 \mid ..$

- Regular expressions in this DSL:
 - $\text{StartWith}(\langle \text{digit} \rangle)$
 - $\text{Concat}(a, \text{Concat}(\text{KleeneStar}(\langle \text{any} \rangle), b))$

Regular Expressions 101

- Consider CFG (simplified):

$r ::= cc \mid \text{StartWith}(r) \mid \text{EndWith}(r) \mid \text{Contains}(r) \mid \text{Concat}(r, r) \mid \text{KleeneStar}(r) \mid \text{Repeat}(r, k)$

$cc ::= \langle \text{digit} \rangle \mid \langle \text{char} \rangle \mid \langle \text{any} \rangle \mid ..$

$k ::= 1 \mid 2 \mid ..$

- Regular expressions in this DSL:
 - $\text{StartWith}(\langle \text{digit} \rangle)$
 - $\text{Concat}(a, \text{Concat}(\text{KleeneStar}(\langle \text{any} \rangle), b))$
 - $\text{Repeat}(\langle \text{char} \rangle, 5)$

How To Explain A Regex?

$r ::= cc \mid \text{StartWith}(r) \mid \text{EndWith}(r) \mid \text{Contains}(r) \mid \text{Concat}(r, r) \mid \text{KleeneStar}(r) \mid \text{Repeat}(r, k)$

$cc ::= \langle \text{digit} \rangle \mid \langle \text{char} \rangle \mid \langle \text{any} \rangle \mid \dots$

$k ::= 1 \mid 2 \mid \dots$

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

How To Explain A Regex?

$r ::= cc \mid \text{StartWith}(r) \mid \text{EndWith}(r) \mid \text{Contains}(r) \mid \text{Concat}(r, r) \mid \text{KleeneStar}(r) \mid \text{Repeat}(r, k)$

$cc ::= \langle \text{digit} \rangle \mid \langle \text{char} \rangle \mid \langle \text{any} \rangle \mid \dots$

$k ::= 1 \mid 2 \mid \dots$

- 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?

$r ::= cc \mid \text{StartWith}(r) \mid \text{EndWith}(r) \mid \text{Contains}(r) \mid \text{Concat}(r, r) \mid \text{KleeneStar}(r) \mid \text{Repeat}(r, k)$

$cc ::= \langle \text{digit} \rangle \mid \langle \text{char} \rangle \mid \langle \text{any} \rangle \mid \dots$

$k ::= 1 \mid 2 \mid \dots$

- 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?

$r ::= cc \mid \text{StartWith}(r) \mid \text{EndWith}(r) \mid \text{Contains}(r) \mid \text{Concat}(r, r) \mid \text{KleeneStar}(r) \mid \text{Repeat}(r, k)$

$cc ::= \langle \text{digit} \rangle \mid \langle \text{char} \rangle \mid \langle \text{any} \rangle \mid \dots$

$k ::= 1 \mid 2 \mid \dots$

- 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?

$r ::= cc \mid \text{StartWith}(r) \mid \text{EndWith}(r) \mid \text{Contains}(r) \mid \text{Concat}(r, r) \mid \text{KleeneStar}(r) \mid \text{Repeat}(r, k)$

$cc ::= \langle \text{digit} \rangle \mid \langle \text{char} \rangle \mid \langle \text{any} \rangle \mid \dots$

$k ::= 1 \mid 2 \mid \dots$

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

How To Explain A Regex?

$r ::= cc \mid \text{StartWith}(r) \mid \text{EndWith}(r) \mid \text{Contains}(r) \mid \text{Concat}(r, r) \mid \text{KleeneStar}(r) \mid \text{Repeat}(r, k)$

$cc ::= \langle \text{digit} \rangle \mid \langle \text{char} \rangle \mid \langle \text{any} \rangle \mid \dots$

$k ::= 1 \mid 2 \mid \dots$

- 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

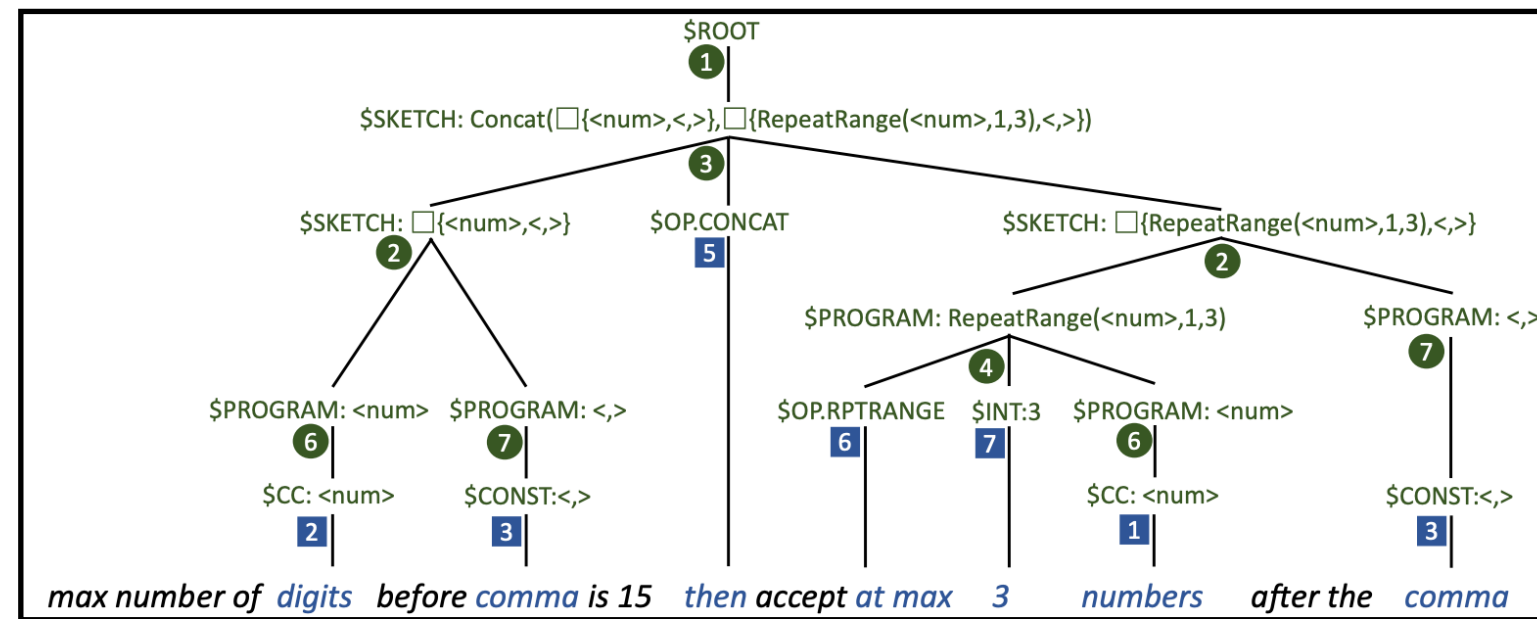
In this paper, we propose a multi-modal synthesis technique

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

Regel System

NL

Examples



Sketch

```

1: procedure SYNTHESIZE( $S, \mathcal{E}^+, \mathcal{E}^-$ )
input: an h-sketch  $S$ , positive and negative examples  $\mathcal{E}^+, \mathcal{E}^-$ 
output: a regex consistent with  $S, \mathcal{E}^+$  and  $\mathcal{E}^-$ , or  $\perp$ 
2:  $P_0 := (v_0, \emptyset, [v_0 \leftarrow S]); \text{worklist} := \{P_0\};$ 
3: while worklist  $\neq \emptyset$  do
4:    $P := \text{worklist.remove}();$ 
5:   if IsConcrete( $P$ ) then
6:     if IsCorrect( $P, \mathcal{E}^+, \mathcal{E}^-$ ) then return  $P$ ;
7:   else if IsSymbolic( $P$ ) then
8:     worklist := worklist  $\cup$  INFERCONSTANTS( $P, \mathcal{E}^+, \mathcal{E}^-$ );
9:   else
10:     $(v, S) := \text{SelectOpenNode}(P);$ 
11:    worklist' := EXPAND( $P, v, S$ );
12:    for all  $P' \in \text{worklist}'$  do
13:      if INFEASIBLE( $P', \mathcal{E}^+, \mathcal{E}^-$ ) then
14:        worklist'.remove( $P'$ );
15:    worklist := worklist  $\cup$  worklist';
16: return  $\perp$ ;

```

Regular Expression

Regae System

- Key insight: even richer specification modalities through UI

Interactive Program Synthesis by Augmented Examples

Tianyi Zhang[†], London Lowmanstone[†], Xinyu Wang[§], Elena L. Glassman[†]

[†]Harvard University, MA, USA



[§]University of Michigan, Ann Arbor, MI, USA

{tianyi, eglassman}@seas.harvard.edu, lowmanstone@college.harvard.edu, xwangsd@umich.edu

① Add input-output examples

Examples

Mark as Literal Mark as General

Input	Output	
+91789	✓	 

③ Annotate parts of synthesized programs to be desired or undesired

Regex Candidates

Include Exclude Synthesize

Synthesis Progress

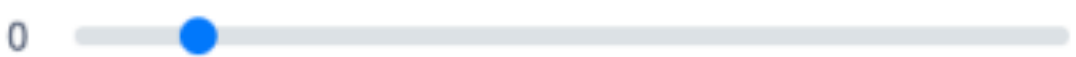
synthesis complete

④ View other similar examples and corner cases

Show me more examples
so I don't have to come up with my own

Show me familiar examples Show me corner cases

Number of Examples Per Cluster: 11

0  101

Regae System

1 Add input-output examples

2 Mark parts of an input as literal or general

Examples

Mark as Literal Mark as General

Input	Output		
+91789	✓		
91789	✓		
91+	✗		
9+1	✗		
abc&^*	✗		
+1	✓		

+ Add New

3 Annotate parts of synthesized programs to be desired or undesired

Regex Candidates

Include Exclude Synthesize

Synthesis Progress

synthesis complete

<num1-9> x repeatatleast x or x

- concat (optional (<+>), repeatatleast (<num>, 1))
- concat (star (<+>), repeatatleast (<num>, 1))
- concat (or (<+>, <num>), repeatatleast (<num>, 1))
- concat (or (<num>, <+>), repeatatleast (<num>, 1))
- concat (or (<num>, repeatatleast (<+>, 1)), repeatatleast (<num>, 1))

4 View other similar examples and corner cases

Show me more examples

so I don't have to come up with my own

Show me familiar examples Show me corner cases

Number of Examples Per Cluster: 11

0 101

Cluster 1: Examples rejected because the selected regex expects non-empty string.

	✗		+
--	---	--	---

Cluster 2: Examples rejected because the selected regex expects more characters. The next character can be 0 to 9 or +.

	✗		+
+2	✓		+

Cluster 3: Examples rejected because the 1st character is not 0 to 9 or +.

	✗		+
+8	✓		+

Regae System

1 Add input-output examples

2 Mark parts of an input as literal or general

3 Annotate parts of synthesized programs to be desired or undesired

4 View other similar examples and corner cases

Examples

Mark as Literal Mark as General

Input	Output		
+91789	✓		
91789	✓		
91+	✗		
9+1	✗		
abc&^*	✗		
+1	✓		

+ Add New

Regex Candidates

Include Exclude Synthesize

Synthesis Progress

synthesis complete

<num1-9> x repeatatleast x or x

- concat (optional (<+>), repeatatleast (<num>, 1))
- concat (star (<+>), repeatatleast (<num>, 1))
- concat (or (<+>, <num>), repeatatleast (<num>, 1))
- concat (or (<num>, <+>), repeatatleast (<num>, 1))
- concat (or (<num>, repeatatleast (<+>, 1)), repeatatleast (<num>, 1))

Show me more examples

so I don't have to come up with my own

Show me familiar examples Show me corner cases

Number of Examples Per Cluster: 11

0 101

Cluster 1: Examples rejected because the selected regex expects non-empty string.

	✗		+
--	---	--	---

Cluster 2: Examples rejected because the selected regex expects more characters. The next character can be 0 to 9 or +.

	✗		+
+2	✓		+

Cluster 3: Examples rejected because the 1st character is not 0 to 9 or +.

	✗		+
+8	✓		+

Regae System

1 Add input-output examples

2 Mark parts of an input as literal or general

Examples

Mark as Literal Mark as General

Input	Output	
+91789	✓	
91789	✓	
91+	✗	
9+1	✗	
abc&^*	✗	
+1	✓	

+ Add New

3 Annotate parts of synthesized programs to be desired or undesired

Regex Candidates

Include Exclude Synthesize

Synthesis Progress

synthesis complete

<num1-9> x repeatatleast x or x

- concat (optional (<+>), repeatatleast (<num>, 1))
- concat (star (<+>), repeatatleast (<num>, 1))
- concat (or (<+>, <num>), repeatatleast (<num>, 1))
- concat (or (<num>, <+>), repeatatleast (<num>, 1))
- concat (or (<num>, repeatatleast (<+>, 1)), repeatatleast (<num>, 1))

4 View other similar examples and corner cases

Show me more examples

so I don't have to come up with my own

Show me familiar examples Show me corner cases

Number of Examples Per Cluster: 11

0 101

Cluster 1: Examples rejected because the selected regex expects non-empty string.

	✗	
--	---	--

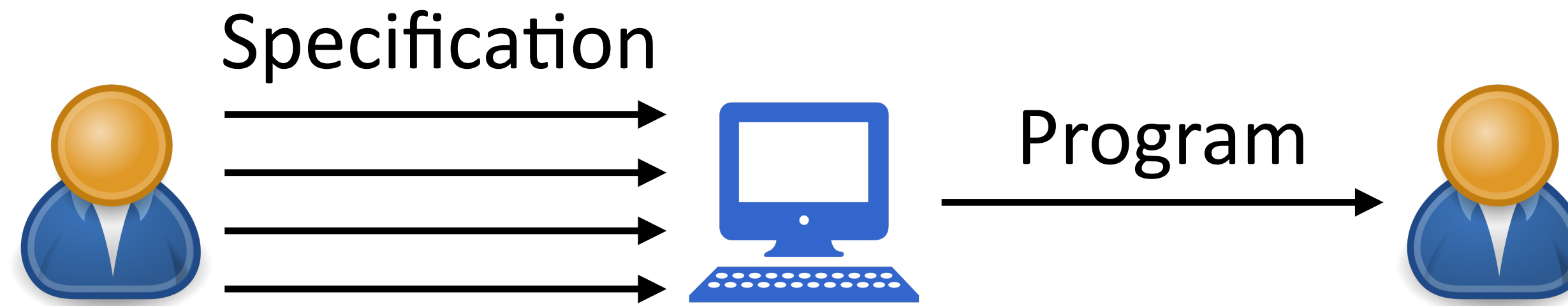
Cluster 2: Examples rejected because the selected regex expects more characters. The next character can be 0 to 9 or +.

	✗	
+2	✓	

Cluster 3: Examples rejected because the 1st character is not 0 to 9 or +.

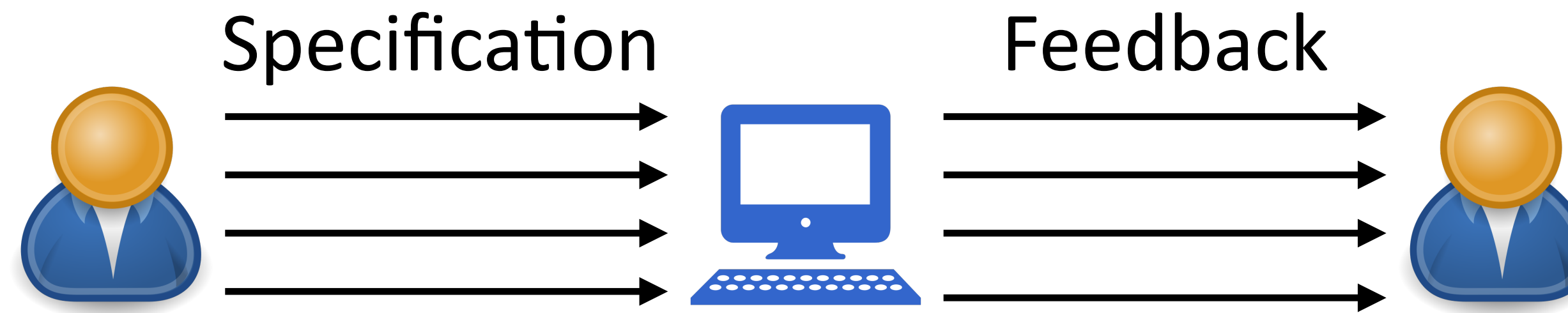
	✗	
+8	✓	

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

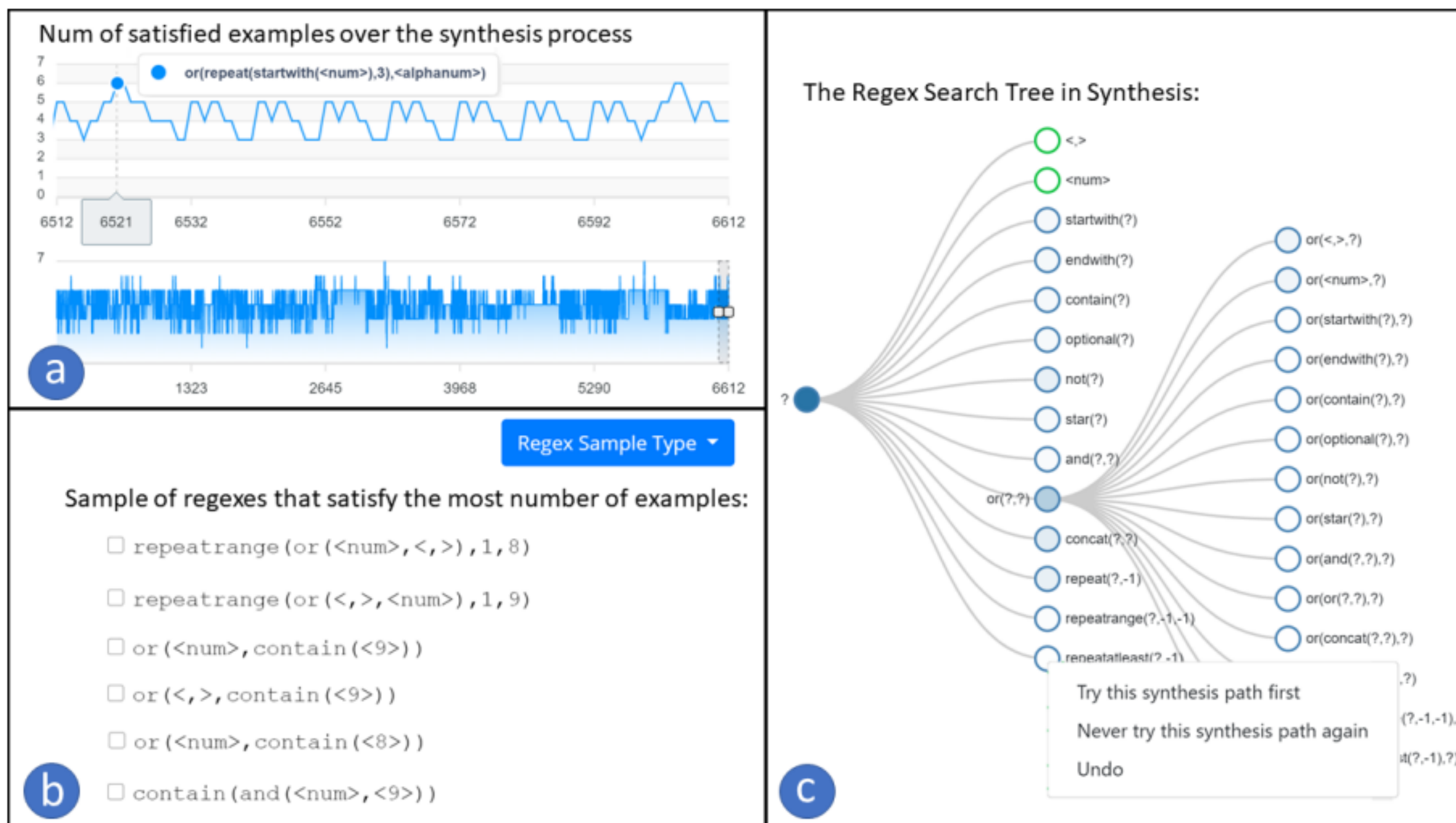
Zhiyang Chen
University of Michigan
Ann Arbor, MI, USA
zhiychen@umich.edu

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

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

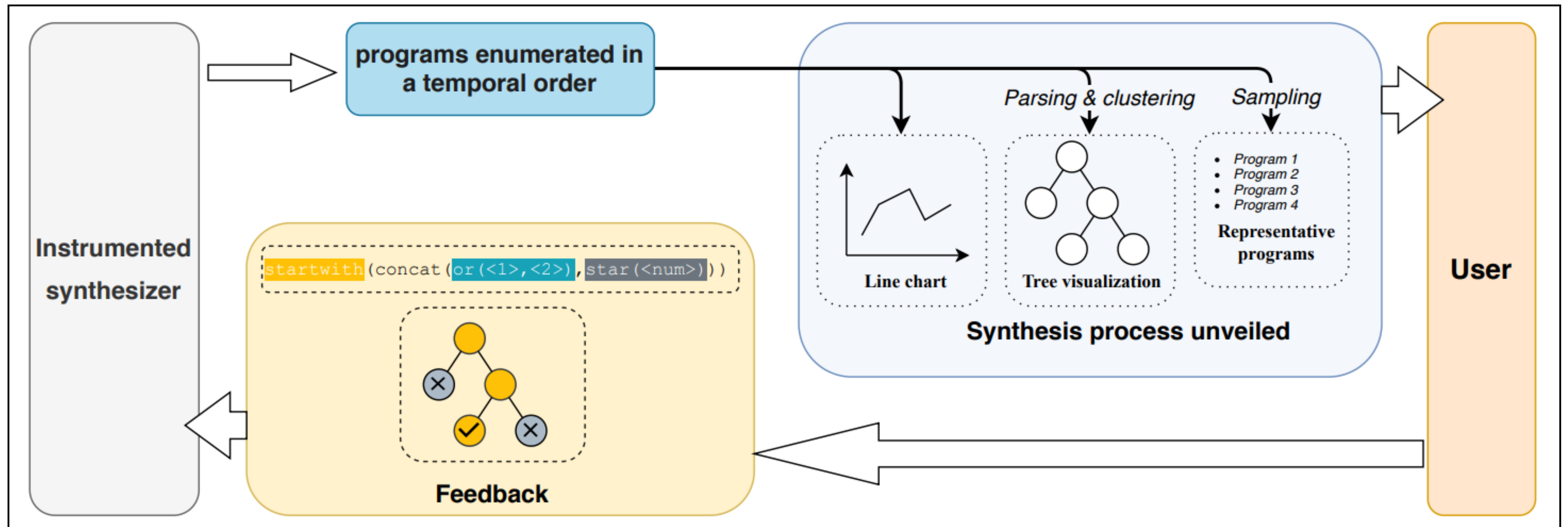
Xinyu Wang
University of Michigan
Ann Arbor, MI, USA
xwangsd@umich.edu

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

