Test-Driven Synthesis

Daniel Perelman[1]

Sumit Gulwani[2] Dan Grossman[1] Peter Provost[3]

[1]University of Washington

[2]Microsoft Research

[3]Microsoft Corporation

June 11, 2014

TDD example[1]

Test cases:

Program:

```
wrap(string s, int len) {
// do nothing
     throw new
NotImplementedException();




}
```

[1] http://blog.8thlight.com/uncle-bob/2013/05/27/TheTransformationPriorityPremise.html

TDD example[1]

Test cases:

1. `wrap("word", 5) == "word"`

Program:

```
wrap(string s, int len) {
// return constant "word"
    return "word";



}
```

[1] http://blog.8thlight.com/uncle-bob/2013/05/27/TheTransformationPriorityPremise.html

TDD example[1]

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`

Program:

```
wrap(string s, int len) {
// return input string
    return s;



}
```

[1] http://blog.8thlight.com/uncle-bob/2013/05/27/TheTransformationPriorityPremise.html

TDD example[1]

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`

Program:

```
wrap(string s, int len) {
// return input if short
    if(s.Len ≤ len)
        return s;
    else
        return "Long\nWord";

}
```

[1] http://blog.8thlight.com/uncle-bob/2013/05/27/TheTransformationPriorityPremise.html

TDD example[1]

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`

Program:

```
wrap(string s, int len) {
// split string at len
    if(s.Len ≤ len)
        return s;
    else
        return s[0:len] +
"\n" + s[len:];
}
```

[1] http://blog.8thlight.com/uncle-bob/2013/05/27/TheTransformationPriorityPremise.html

TDD example[1]

Test cases:

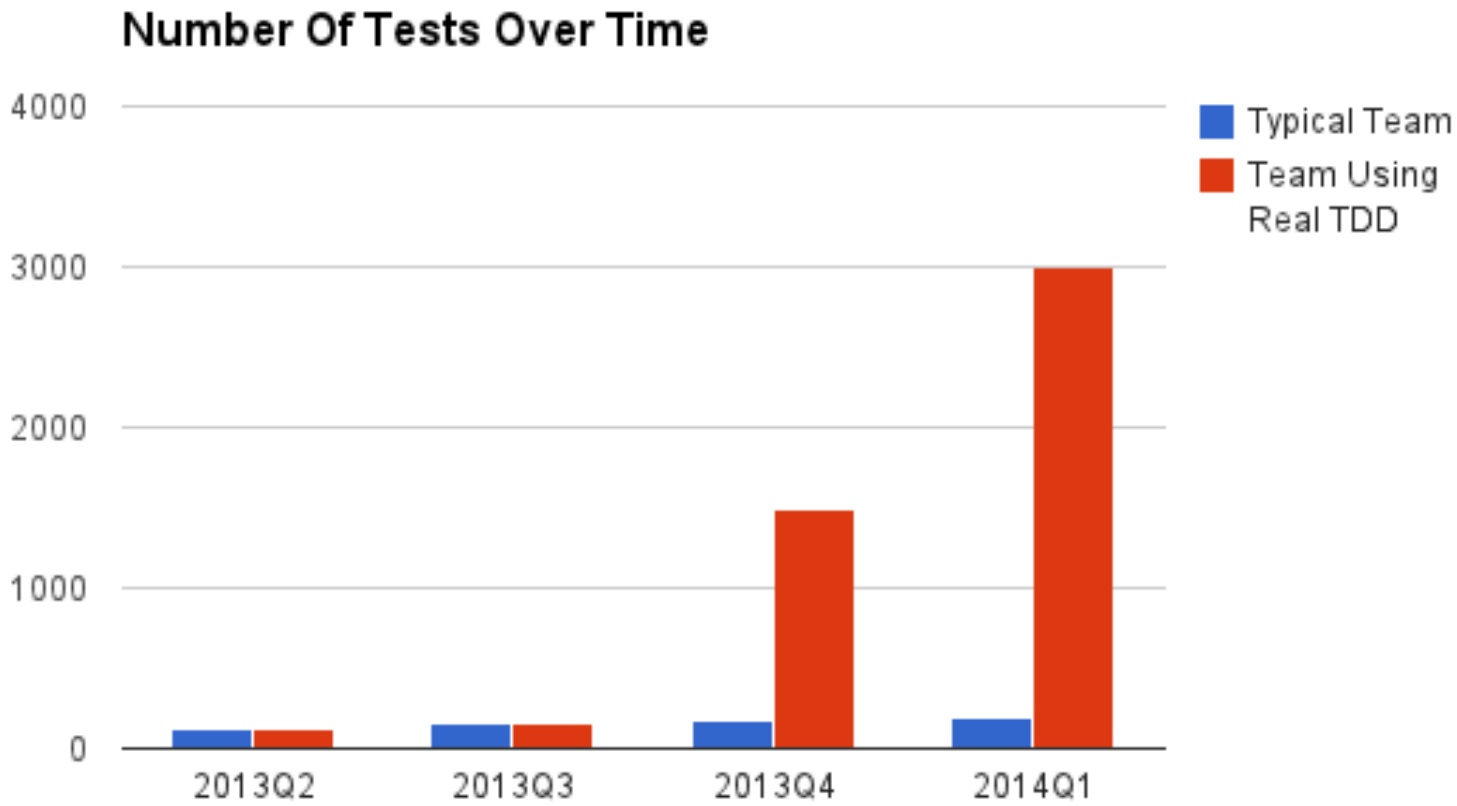
1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`
5. `wrap("LongerWord", 2) == "Lo\nng\ner\nWo\nrd"`

Program:

```
wrap(string s, int len) {
// wrap word to length len
    if(s.Len ≤ len)
        return s;
    else
        return s[0:len] +
"\n" + wrap(s[len:], len);
}
```

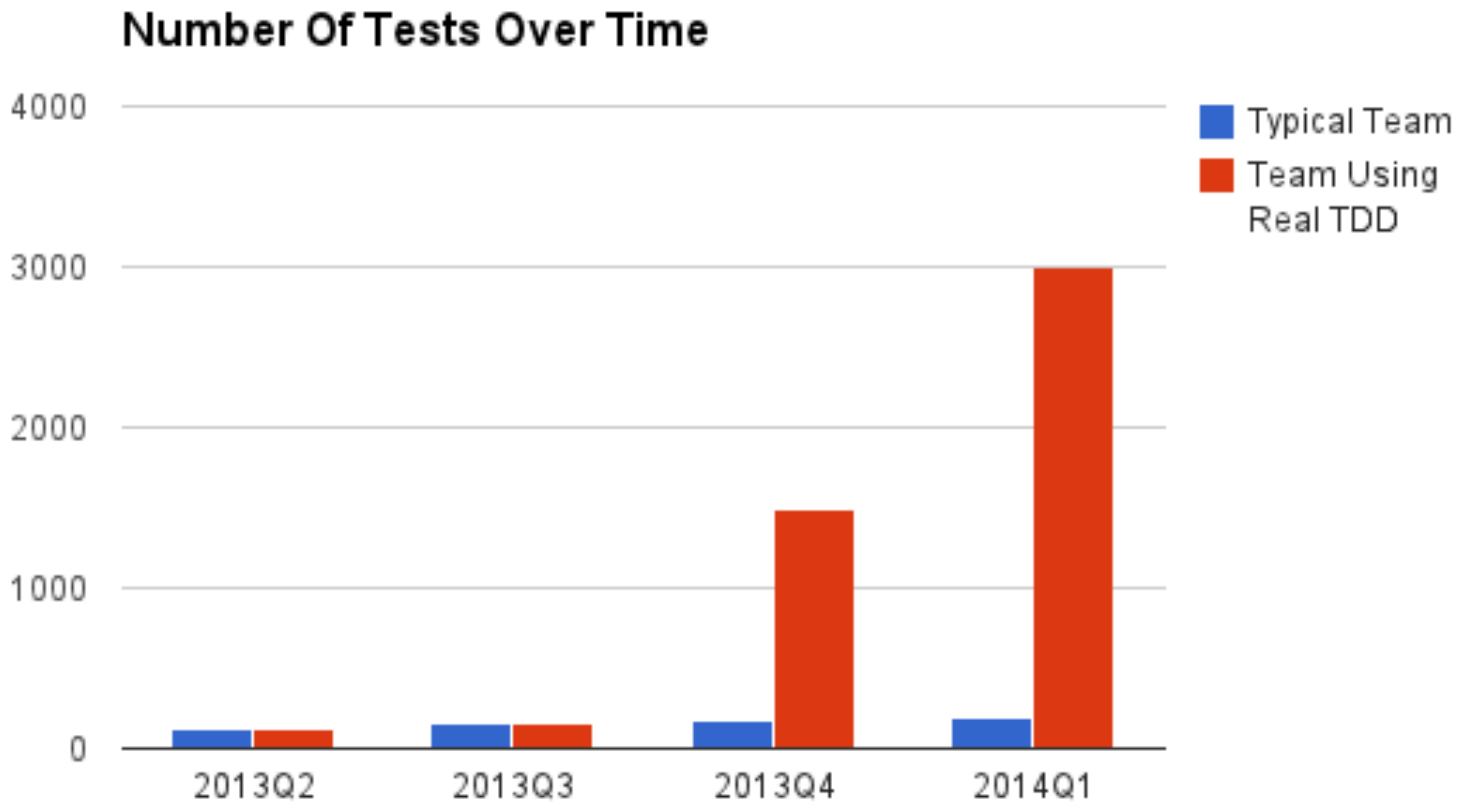
[1] http://blog.8thlight.com/uncle-bob/2013/05/27/TheTransformationPriorityPremise.html

Google's Real TDD[2]



[2] http://googletesting.blogspot.com/2014/04/the-real-test-driven-development.html

Google's Real TDD[2] (April Fools' Day joke)



[2] http://googletesting.blogspot.com/2014/04/the-real-test-driven-development.html

Example domain: string transformations

"Spreadsheet Data Manipulation using Examples,
CACM 2012,
Sumit Gulwani, William Harris, Rishabh Singh"

$\Downarrow$

"Spreadsheet Data Manipulation using Examples\n
Gulwani, S.; Harris, W.; Singh, R.\n
Communications of the ACM, 2012"

Example domain: table transformations

	Qual 1	Qual 2	Qual 3
Andrew	01.02.2003	27.06.2008	06.04.2007
Ben	31.08.2001		05.07.2004
Carl		18.04.2003	09.12.2009

Andrew	Qual 1	01.02.2003
Andrew	Qual 2	27.06.2008
Andrew	Qual 3	06.04.2007
Ben	Qual 1	31.08.2001
Ben	Qual 3	05.07.2004
Carl	Qual 2	18.04.2003
Carl	Qual 3	09.12.2009

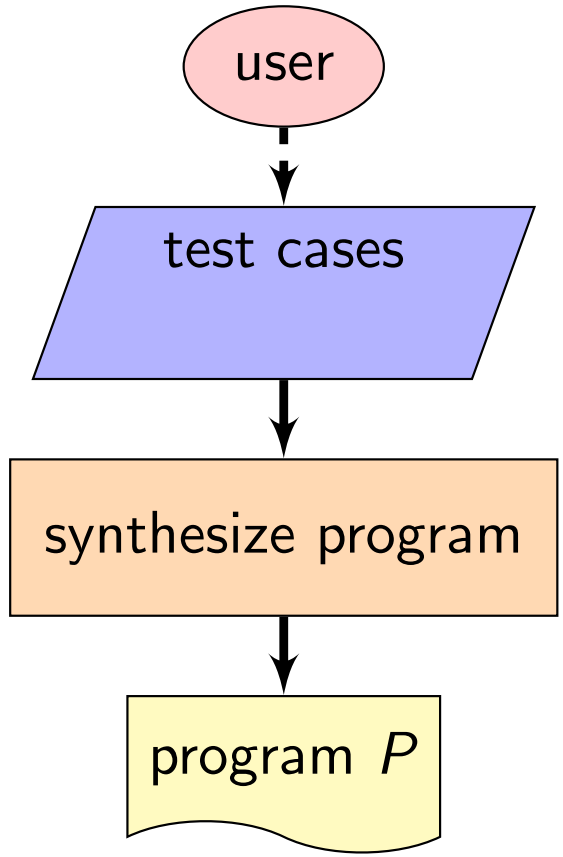
Example domain: XML transformations

```
<doc>
<p>1</p>
<p class='a'>2</p>
<p>3</p>
<p>4</p>
<p class='b'>5</p>
<p>6</p>
<p class='c'>7</p>
</doc>
```

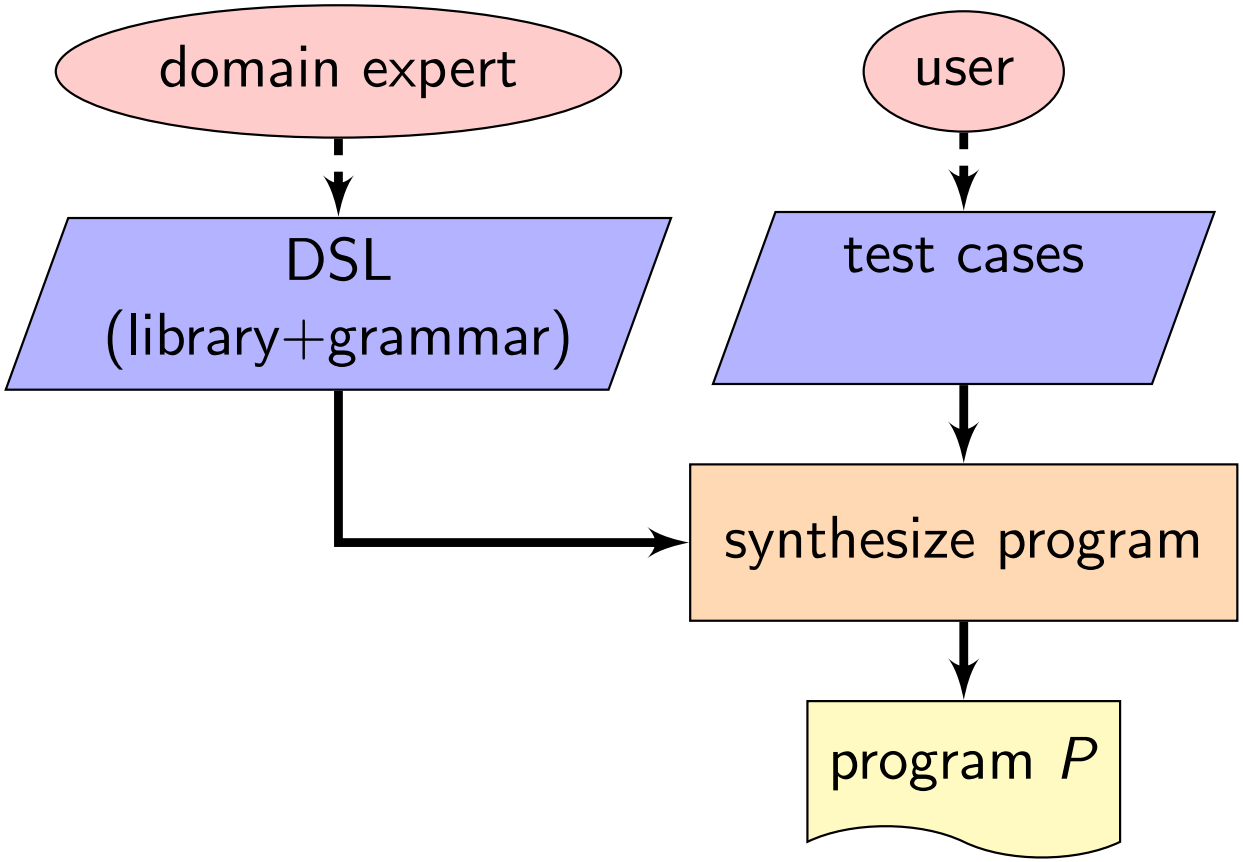
$\Rightarrow$

```
<doc>
<p>1</p>
<p class='a'>2</p>
<p class='a'>3</p>
<p class='a'>4</p>
<p class='b'>5</p>
<p class='b'>6</p>
<p class='c'>7</p>
</doc>
```

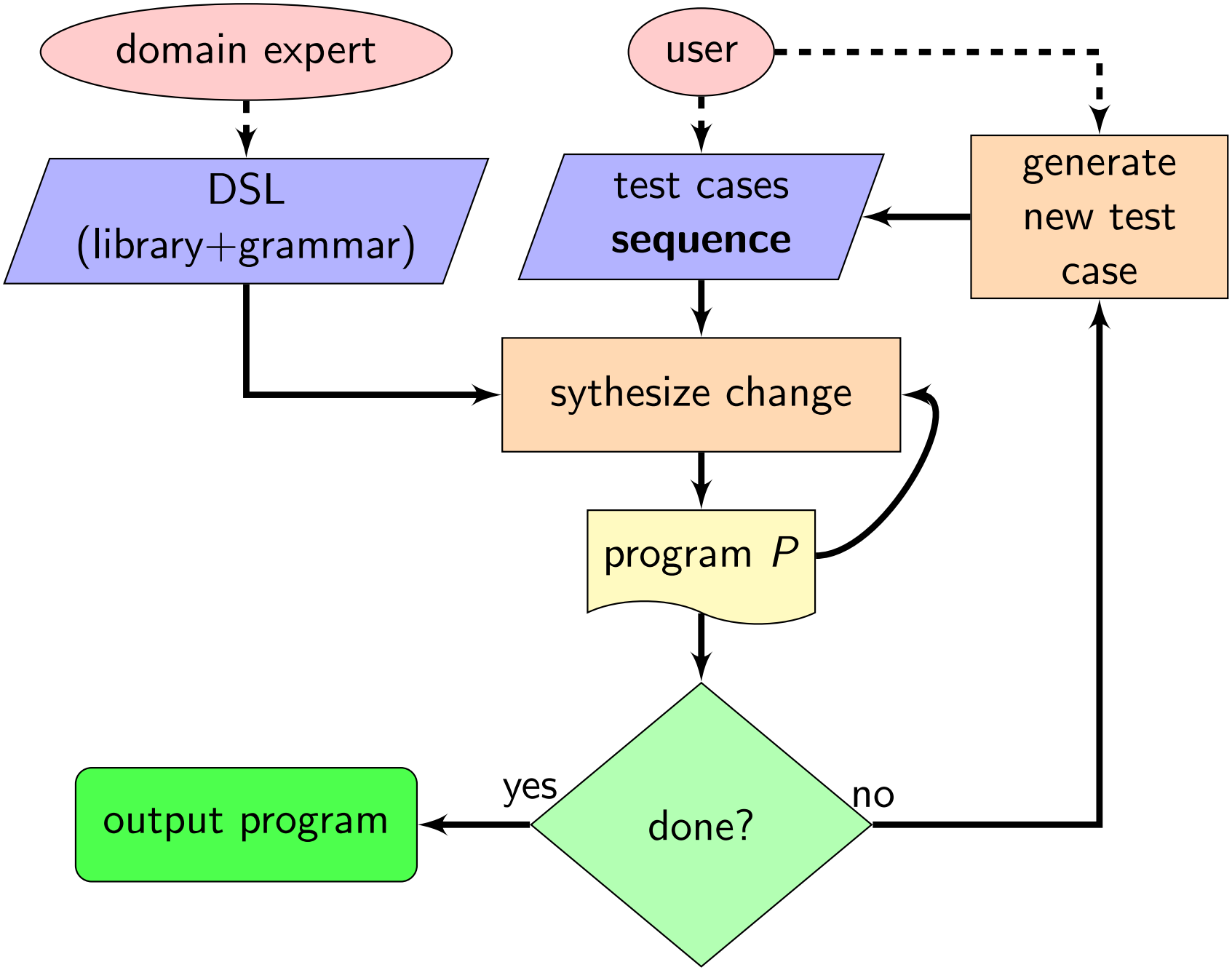
Workflow



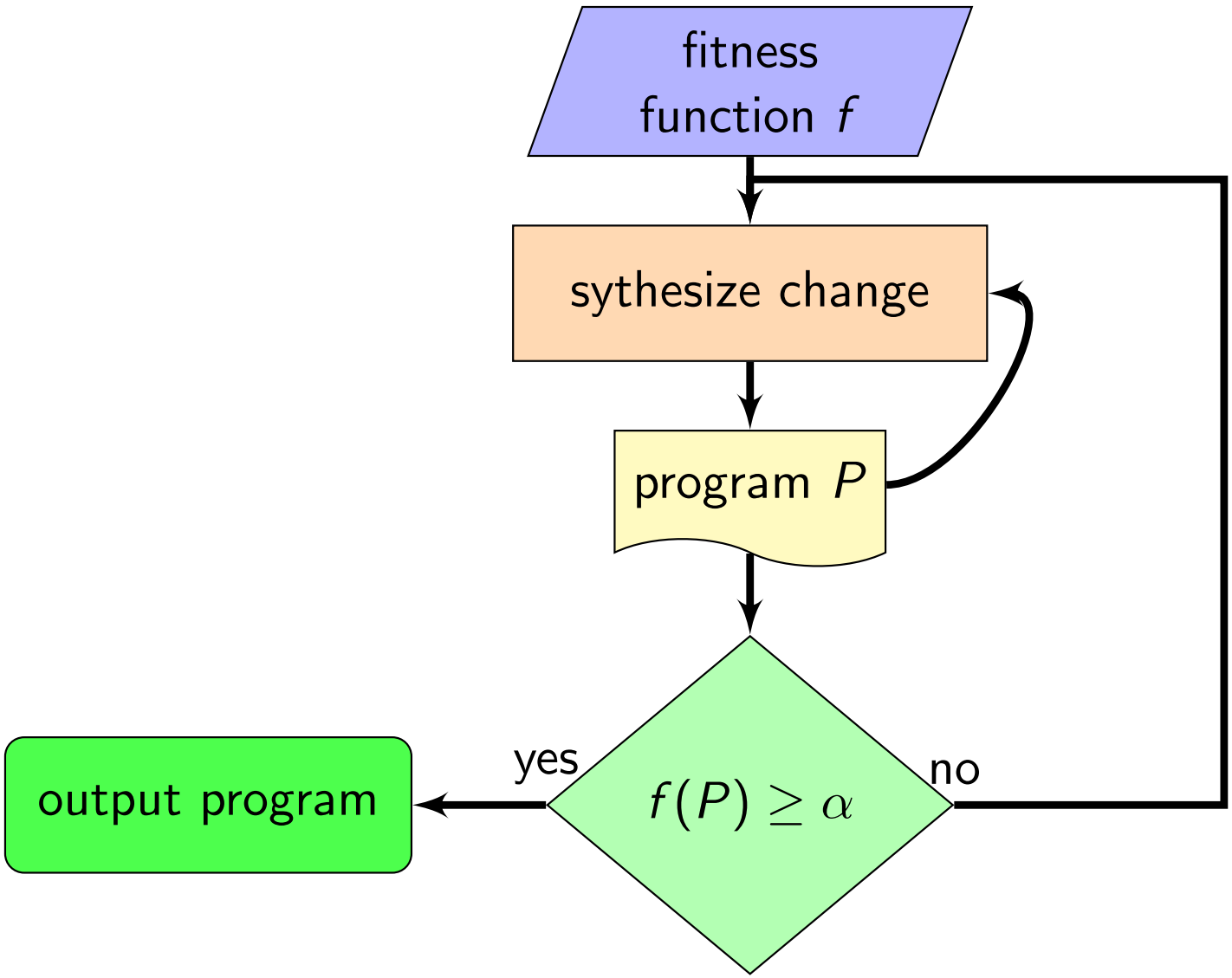
Workflow


domain expert
user
DSL
(library+grammar)
test cases
synthesize program
program P

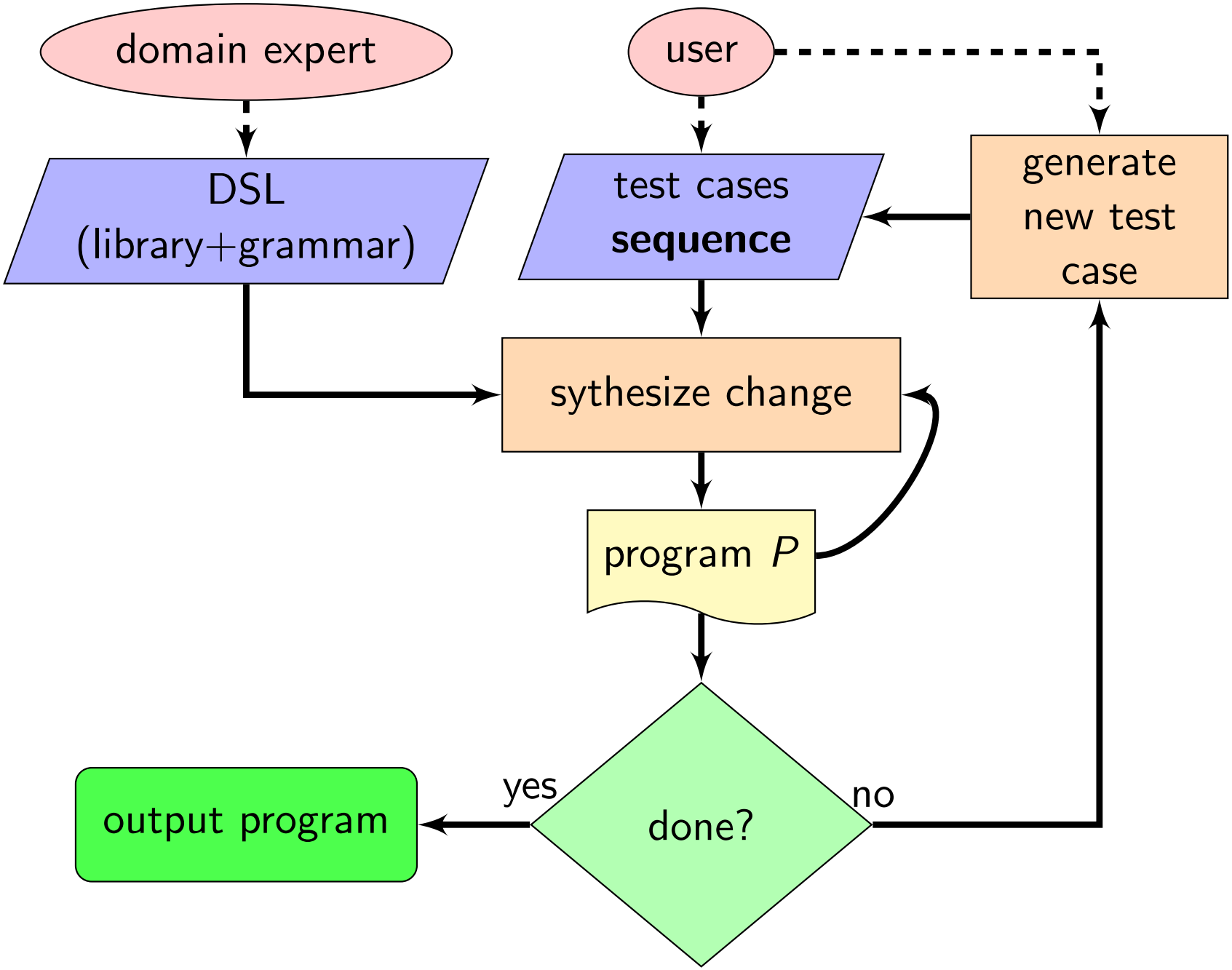
Workflow


domain expert
user
DSL
(library+grammar)
test cases
sequence
generate
new test
case
sythesize change
program P
output program
yes
done?
no

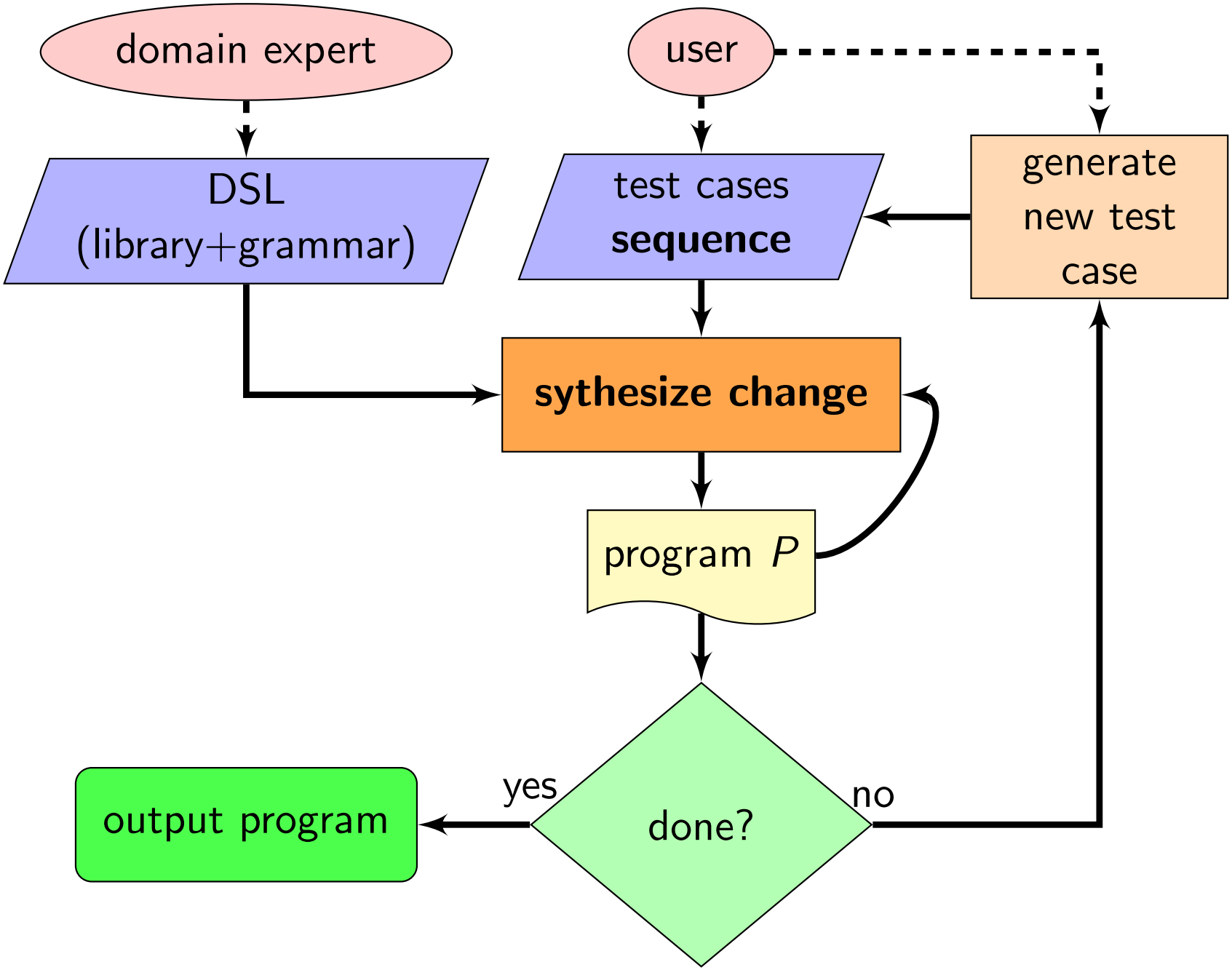
Contrast with genetic programming



Workflow


domain expert
user
DSL
(library+grammar)
test cases
sequence
generate
new test
case
sythesize change
program P
output program
yes
done?
no

Workflow


domain expert
user
DSL
(library+grammar)
test cases
sequence
generate
new test
case
sythesize change
program P
output program
yes
done?
no

Modifying program

- Replace single subexpression on new test case's path
- **Where** to modify program
- **What** to replace with

Where to modify

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`
5. `wrap("LongerWord", 2) == "Lo\nng\ner\nWo\nrd"`

Program:

```
wrap(string s, int len) {
// split string at len
    if(s.Len ≤ len)
        return s;
    else
        return s[0:len] +
"\n" + s[len:];
}
```

Where to modify

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`
5. `wrap("LongerWord", 2) == "Lo\nng\ner\nWo\nrd"`

Program:

```
wrap(string s, int len) {
// split string at len
    if(s.Len ≤ len)
        return s;
    else
        return s[0:len] +
"\n" + s[len:];
}
```

Where to modify

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`
5. `wrap("LongerWord", 2) == "Lo\nng\ner\nWo\nrd"`

Program:

```
wrap(string s, int len) {
// split string at len
    if(s.Len ≤ len)
        return s;
    else
        return s[0:len] +
"\n" + s[len:];
}
```

Where to modify

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`
5. `wrap("LongerWord", 2) == "Lo\nng\ner\nWo\nrd"`

Program:

```
wrap(string s, int len) {
// split string at len
    if(s.Len ≤ len)
        return s;
    else
        return s[0:len] +
"\n" + s[len:];
}
```

Where to modify

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`
5. `wrap("LongerWord", 2) == "Lo\nng\ner\nWo\nrd"`

Program:

```
wrap(string s, int len) {
// split string at len
    if(s.Len ≤ len)
        return s;
    else
        return s[0:len] +
"\n" + s[len:];
}
```

Where to modify

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`
5. `wrap("LongerWord", 2) == "Lo\nng\ner\nWo\nrd"`

Program:

```
wrap(string s, int len) {
// split string at len
    if(s.Len ≤ len)
        return s;
    else
        return s[0:len] +
"\n" + s[len:];
}
```

Where to modify

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`
5. `wrap("LongerWord", 2) == "Lo\nng\ner\nWo\nrd"`

Program:

```
wrap(string s, int len) {
// split string at len
    if(s.Len ≤ len)
        return s;
    else
        return s[0:len] +
"\n" + s[len:];
}
```

Where to modify

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`
5. `wrap("LongerWord", 2) == "Lo\nng\ner\nWo\nrd"`

Program:

```
wrap(string s, int len) {
// split string at len
    if(s.Len ≤ len)
        return s;
    else
        return s[0:len] +
"\n" + s[len:];
}
```

Where to modify

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`
5. `wrap("LongerWord", 2) == "Lo\nng\ner\nWo\nrd"`

Program:

```
wrap(string s, int len) {
// split string at len
    if(s.Len ≤ len)
        return s;
    else
        return s[0:len] +
"\n" + s[len:];
}
```

Where to modify

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`
5. `wrap("LongerWord", 2) == "Lo\nng\ner\nWo\nrd"`

Program:

```
wrap(string s, int len) {
// split string at len
    if(s.Len ≤ len)
        return s;
    else
        return s[0:len] +
"\n" + s[len:];
}
```

Where to modify

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`
5. `wrap("LongerWord", 2) == "Lo\nng\ner\nWo\nrd"`

Program:

```
wrap(string s, int len) {
// split string at len
    if(s.Len ≤ len)
        return s;
    else
        return s[0:len] +
"\n" + s[len:];
}
```

Where to modify

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`
5. `wrap("LongerWord", 2) == "Lo\nng\ner\nWo\nrd"`

Program:

```
wrap(string s, int len) {
// wrap word to length len
    if(s.Len ≤ len)
        return s;
    else
        return s[0:len] +
"\n" + wrap(s[len:], len);
}
```

What to put there

- DSL defines space of expressions
- Prefer expressions found in previous program
  - e.g., `wrap(s[len:], len)` contains `s[len:]`

Conditionals

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`

program path	test case 1	2	3	4
`""`				
`"word"`	✓			
`"foobar"`		✓		
`"Long\nWord"`			✓	
`"Longer\nWord"`				✓
`s`	✓	✓		
`s[0:len] + "\n" + s[:len]`			✓	✓

Conditionals

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`

program path	test case 1	2	3	4
`""`				
`"word"`	✓			
`"foobar"`		✓		
`"Long\nWord"`			✓	
`"Longer\nWord"`				✓
`s`	✓	✓		
`s[0:len] + "\n" + s[:len]`			✓	✓

Conditionals

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`

	test case			
program path	1	2	3	4
`""`				
`"word"`	✓			
`"foobar"`		✓		
`"Long\nWord"`			✓	
`"Longer\nWord"`				✓
`s`	✓	✓		
`s[0:len] + "\n" + s[:len]`			✓	✓
guard	1	2	3	4
`s == "word"`	T			
`len == 6`		T		T
`s.Len == len`		T		
`s.Len ≤ len`	T	T		

Conditionals

Test cases:

1. `wrap("word", 5) == "word"`
2. `wrap("foobar", 6) == "foobar"`
3. `wrap("LongWord", 4) == "Long\nWord"`
4. `wrap("LongerWord", 6) == "Longer\nWord"`

	test case			
program path	1	2	3	4
`""`				
`"word"`	✓			
`"foobar"`		✓		
`"Long\nWord"`			✓	
`"Longer\nWord"`				✓
`s`	✓	✓		
`s[0:len] + "\n" + s[:len]`			✓	✓
guard	1	2	3	4
`s == "word"`	T			
`len == 6`		T		T
`s.Len == len`		T		
`s.Len ≤ len`	T	T		

Conditionals

Program:

```
wrap(string s, int len) {
// split string at len
    if(s.Len ≤ len)
        return s;
    else
        return s[0:len]
    + "\n"
    + s[len:];
}
```

	test case			
program path	1	2	3	4
`""`				
`"word"`	✓			
`"foobar"`		✓		
`"Long\nWord"`			✓	
`"Longer\nWord"`				✓
`s`	✓	✓		
`s[0:len] + "\n" + s[:len]`			✓	✓
guard	1	2	3	4
`s == "word"`	T			
`len == 6`		T		T
`s.Len == len`		T		
`s.Len ≤ len`	T	T		

Loops

Program:

```
wrap(string s, int len) {
// wrap word to length len
    if(s.Len ≤ len)
        return s;
    else
        return s[0:len]
    + "\n"
    + wrap(s[len:], len);
}
```

	test case			
program path	1	2	3	4
`""`				
`"word"`	✓			
`"foobar"`		✓		
`"Long\nWord"`			✓	
`"Longer\nWord"`				✓
`s`	✓	✓		
`s[0:len] + "\n" + s[:len]`			✓	✓
guard	1	2	3	4
`s == "word"`	T			
`len == 6`		T		T
`s.Len == len`		T		
`s.Len ≤ len`	T	T		

Loops

- Recursion (if in DSL)
- Higher-order functions (if in DSL)
- Direct support for certain loops: see paper for details

Evaluation

- Three DSLs focused on end-user data transformation tasks.
  - String transformations (FlashFill+extensions)
  - Spreadsheet table transformations
  - XML transformations
- Able to synthesize many examples from help forums in under a minute.[3]

[3] https://homes.cs.washington.edu/~perelman/publications/pldi14-tds.zip

Evaluation

- Three DSLs focused on end-user data transformation tasks.
  - String transformations (FlashFill+extensions)
  - Spreadsheet table transformations
  - XML transformations
- Able to synthesize many examples from help forums in under a minute.[3]
- Good news: test case sequences are very short (easy for user)

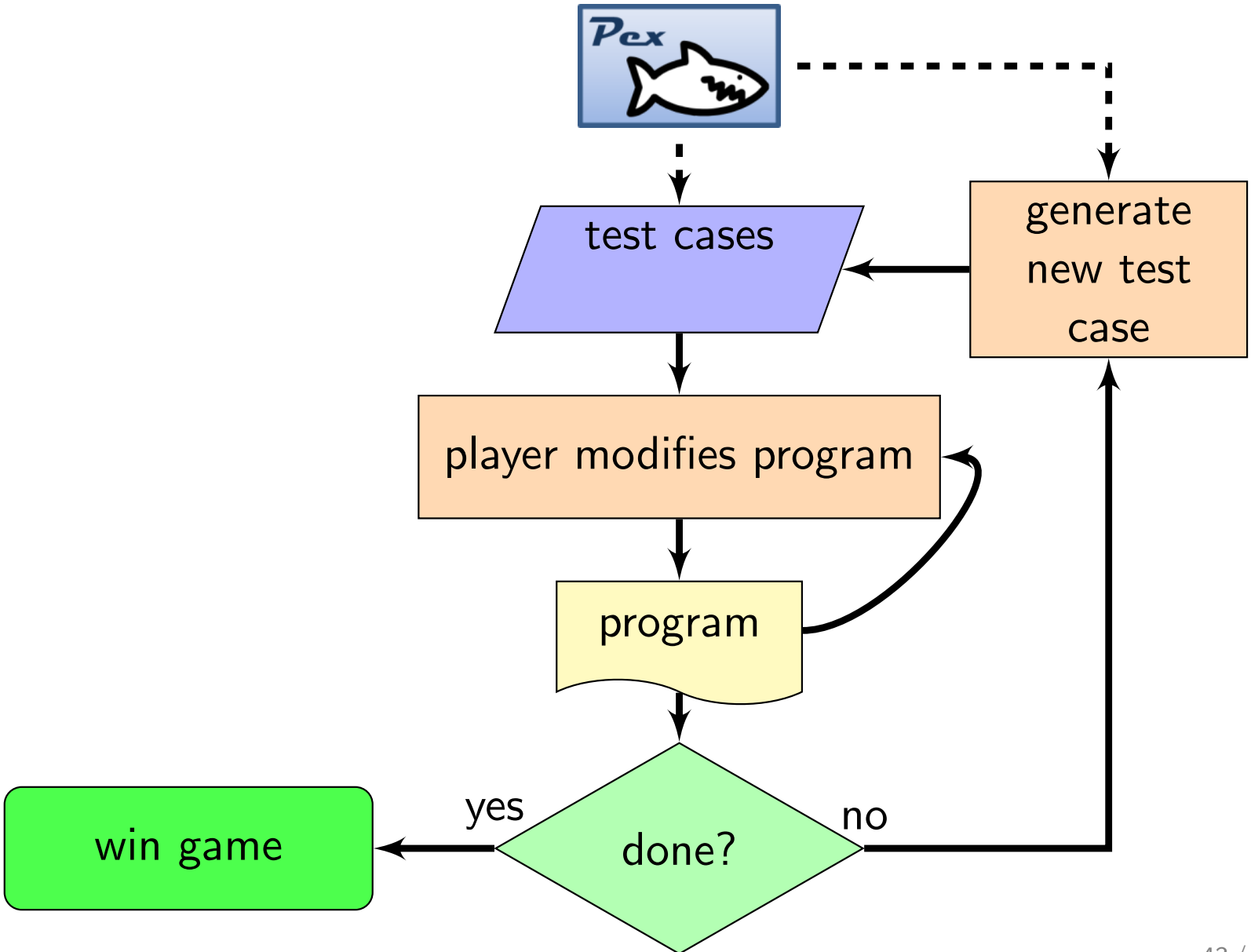
[3] https://homes.cs.washington.edu/~perelman/publications/pldi14-tds.zip

Evaluation

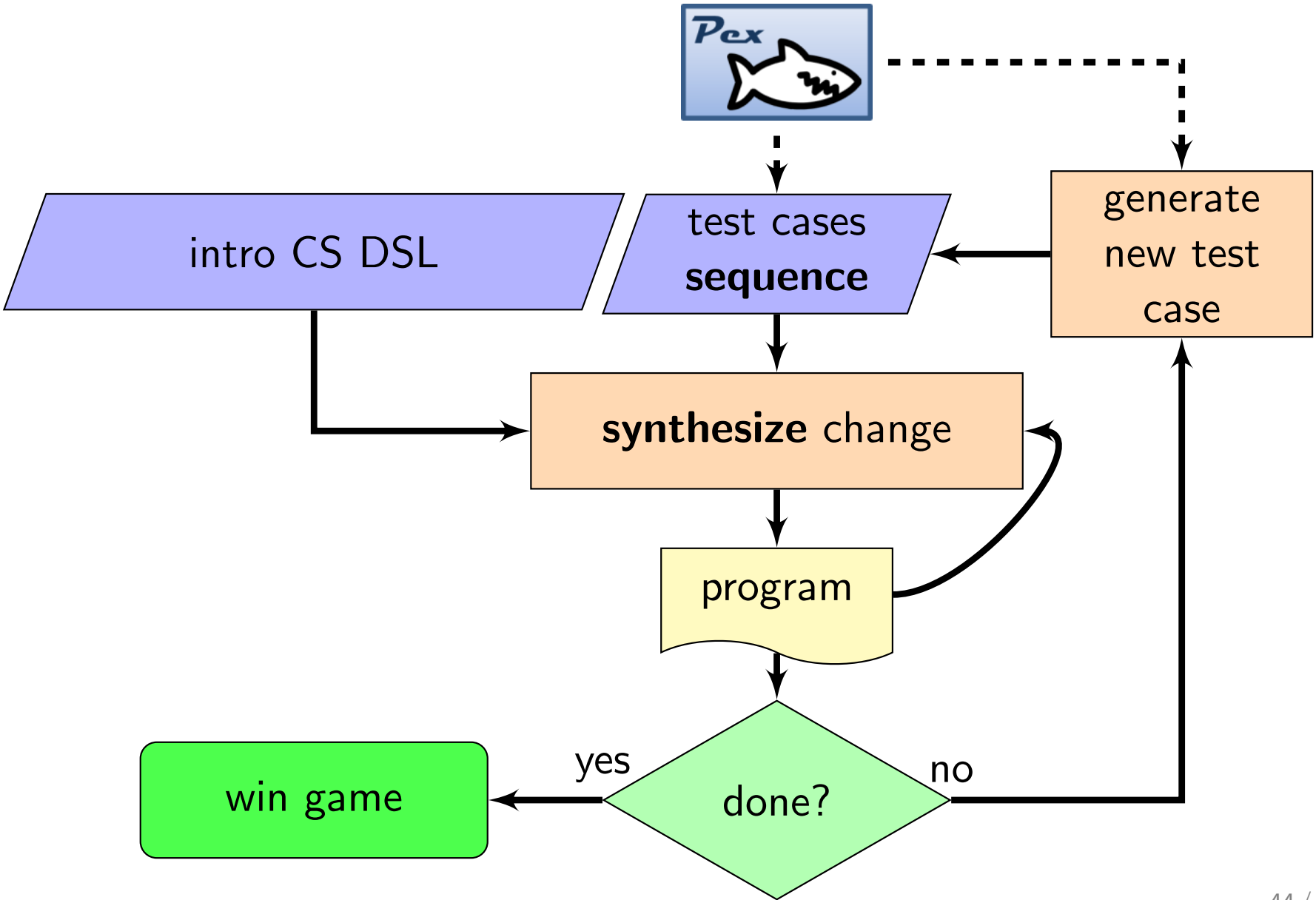
- Three DSLs focused on end-user data transformation tasks.
  - String transformations (FlashFill+extensions)
  - Spreadsheet table transformations
  - XML transformations
- Able to synthesize many examples from help forums in under a minute.[3]
- Good news: test case sequences are very short (easy for user)
- Bad news: test case sequences are very short (difficult to evaluate usefulness of sequences)

[3] https://homes.cs.washington.edu/~perelman/publications/pldi14-tds.zip

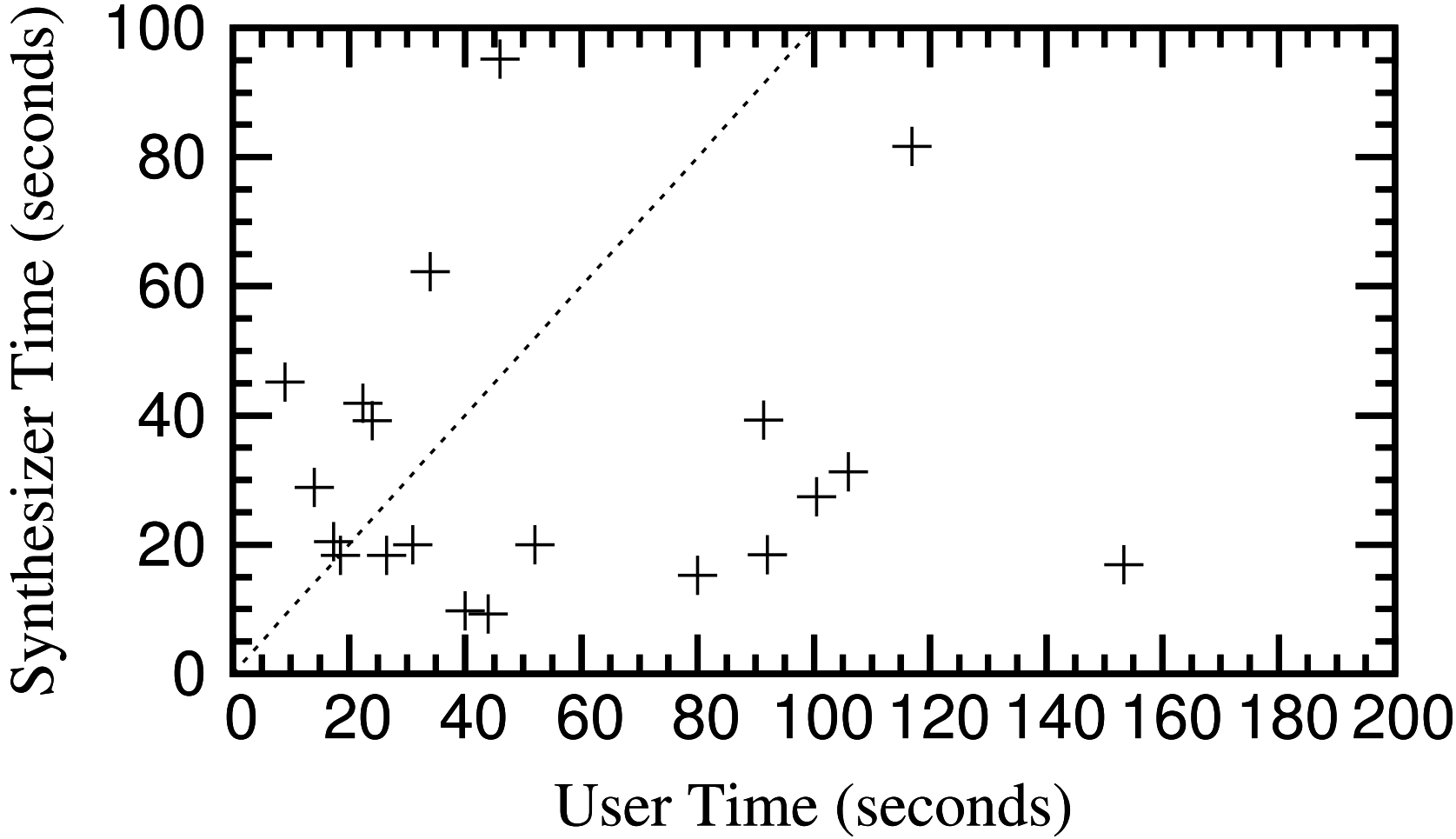
Pex4Fun (now https://www.CodeHunt.com/)



Pex4Fun (now https://www.CodeHunt.com/)



Comparison to human players

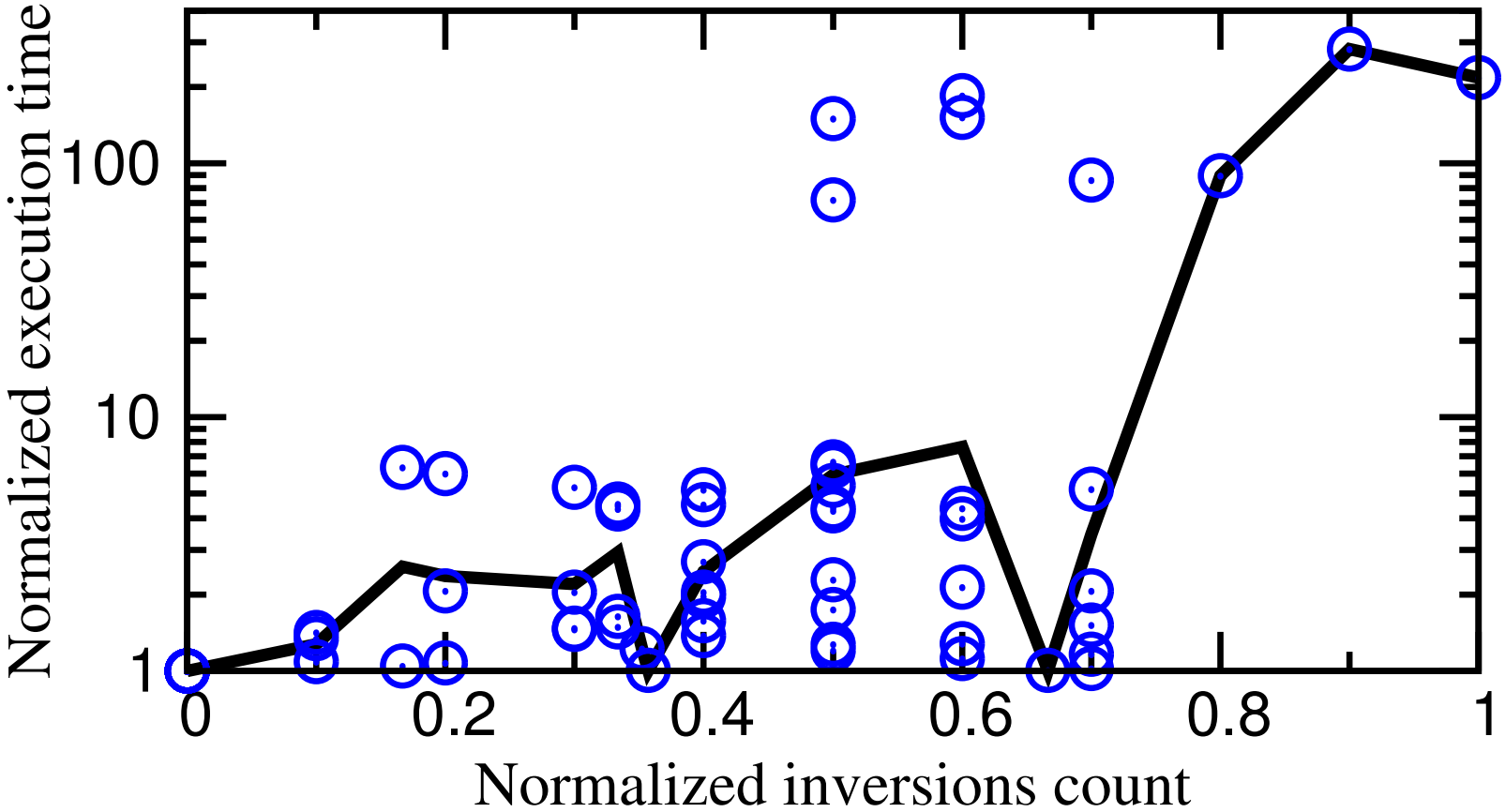


Completion times comparable to human players

Hypothesis:

- Test sequence order matters...
- ...but not too much.

Slowdown due to reordering test case sequence



Robust to small reorderings, fails on larger reorderings

Contributions

- Synthesis workflow inspired by TDD
- Search strategy enabled by this workflow, parameterized by a DSL

Questions

Backup slides

Component-based synthesis example

Starting components:

- a+b
- 0
- x
- y

Generated components (1 step):

- 0+b
- x+b
- y+b

Generated components (2 steps):

- 0+0
- x+0
- y+0
- 0+x
- x+x
- y+x
- 0+y
- x+y
- y+y

Component-based synthesis example

Starting components:

- a+b
- 0
- x
- y

Generated components (1 step):

- 0+b
- x+b
- y+b

Generated components (2 steps):

- 0+0=0
- x+0=x
- y+0=y
- 0+x=x
- x+x
- y+x
- 0+y=y
- x+y=y+x
- y+y

Component-based synthesis example

Starting components:

- a+b
- 0
- x
- y

Generated components (1 step):

- 0+b
- x+b
- y+b

Generated components (2 steps):

- 0+0=0
- x+0=x
- y+0=y
- 0+x=x
- x+x
- y+x
- 0+y=y
- x+y=y+x
- y+y

Component-based synthesis example

Starting components:

- a+b
- 0=[0,0]
- x=[10,-31]
- y=[-10,31]

Generated components (1 step):

- 0+b
- x+b
- y+b

Generated components (2 steps):

- 0+0=[0,0]
- x+0=[10,-31]
- y+0=[-10,31]
- 0+x=[10,-31]
- x+x=[20,-62]
- y+x=[0,0]
- 0+y=[-10,31]
- x+y=[0,0]
- y+y=[-20,62]

Component-based synthesis example

Starting components:

- a+b
- 0=[0,0]
- x=[10,-31]
- y=[-10,31]

Generated components (1 step):

- 0+b
- x+b
- y+b

Generated components (2 steps):

- 0+0=[0,0]
- x+0=[10,-31]
- y+0=[-10,31]
- 0+x=[10,-31]
- x+x=[20,-62]
- y+x=[0,0]
- 0+y=[-10,31]
- x+y=[0,0]
- y+y=[-20,62]

Loops

- Rule-based generation of test cases for loop body from test cases for loop

Loops

- Rule-based generation of test cases for loop body from test cases for loop

- For loop example:

```
Factorial(1)=1
Factorial(2)=2
Factorial(3)=6
Factorial(4)=24
```

$\Rightarrow$

```
Factorial_body(2,1)=2
Factorial_body(3,2)=6
Factorial_body(4,6)=24
```

Loops

- Rule-based generation of test cases for loop body from test cases for loop

- For loop example:

```
Factorial(1)=1
Factorial(2)=2
Factorial(3)=6
Factorial(4)=24
```

$\Rightarrow$

```
Factorial_body(2,1)=2
Factorial_body(3,2)=6
Factorial_body(4,6)=24
```

- Array by-element example:

```
CSum([6,2,8],[4,3,7])
    =[10,15,30]
```

$\Rightarrow$

```
CSum_body(6,4,0,[])=10
CSum_body(2,3,1,[10])=15
CSum_body(8,7,2,[10,15])=30
```