Mining Temporal Relationships between Data Invariants

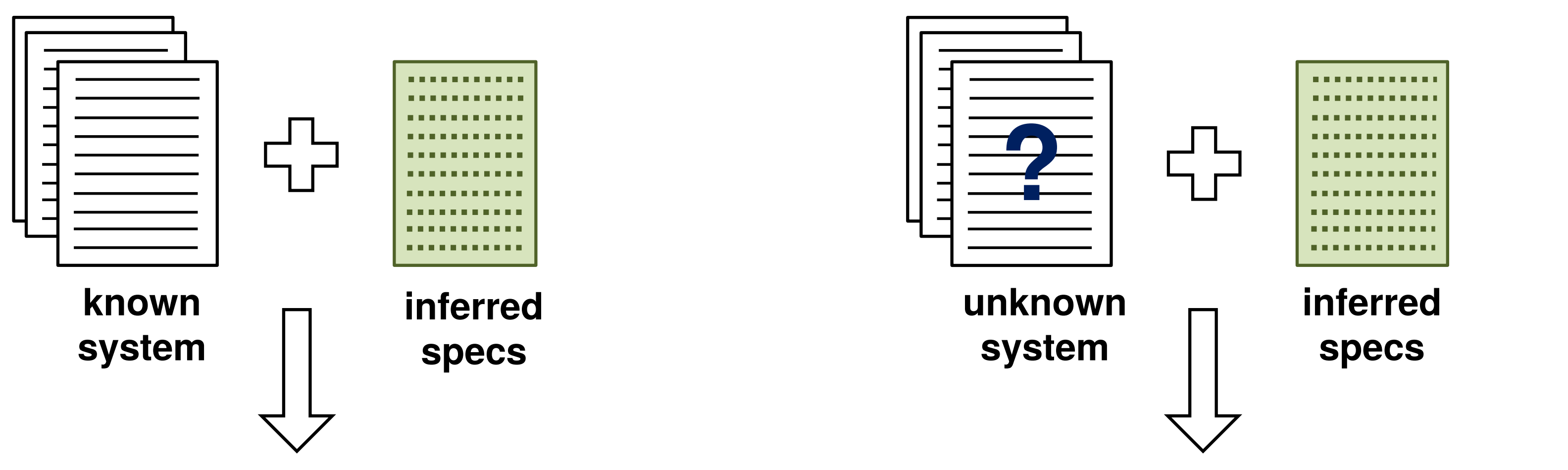
Caroline Lemieux
Department of Computer Science
University of British Columbia

Program specifications (specs) are useful

Developers rarely write down program specs

- specifications can be tedious to specify manually
- may fall out of date quickly

Spec inference: likely specs without manual effort



- program maintenance [1]
- confirm expected behavior[2]
- bug detection[2]
- test generation[3]

- system comprehension[4]
- system modeling[4]
- reverse engineering[1]

Temporal Specs

`enqueue()` is always followed by `dequeue()`

```
create()
enqueue(5)
enqueue(1)
observe()
enqueue(24)
enqueue(7)
dequeue()
enqueue(19)
observe()
dequeue()
```

relate events through time

Data Specs

at exit of `enqueue()`, `size >= 1`

```
enQ::enter
size == 0
enQ::exit
size == 1
enQ::enter
size == 1
enQ::exit
size == 2
deQ::enter
size == 2
```

describe data at specific time

But: data values may persist or interact through time

data-temporal specifications

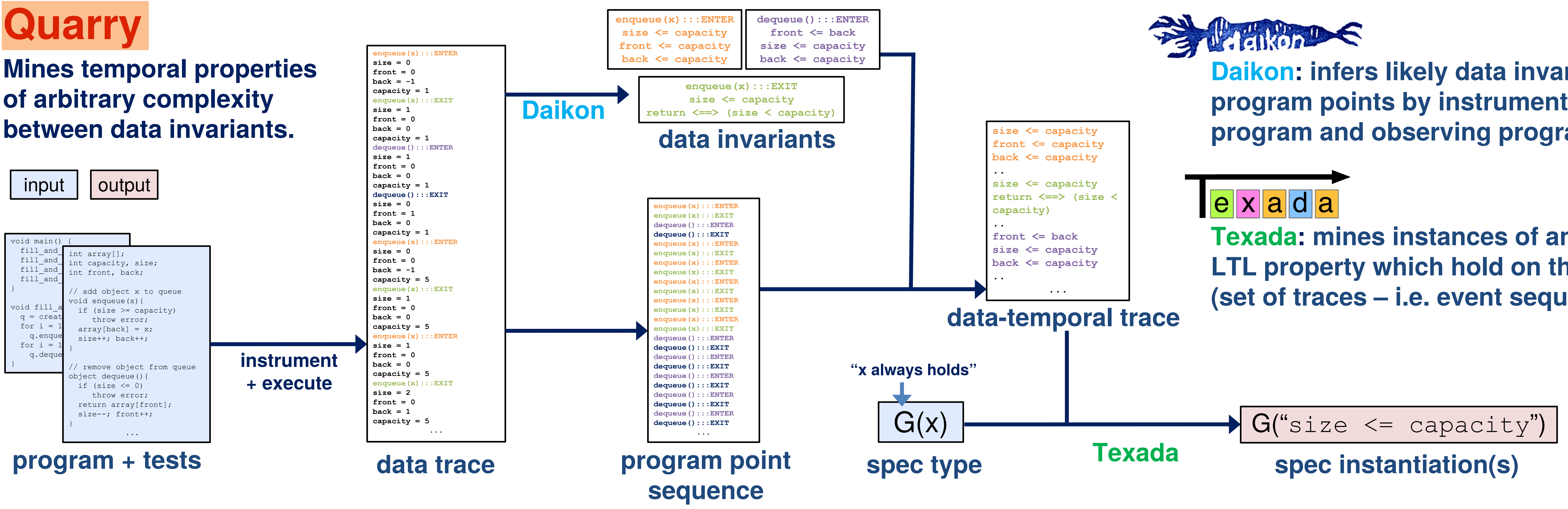
`size >= 1` is always followed by `size == 0`

i.e., all items are eventually removed from the queue

```
size == 0
size == 1
size == 2
size == 2
size == 1
size == 2
size == 3
size == 2
size == 1
size == 0
```

provide more information than data specs or temporal specs alone

Quarry



Mines temporal properties of arbitrary complexity between data invariants.

```
void main() {
  fill_and_
  fill_and_
  fill_and_
  fill_and_
}

void fill_a
  q = creat
  for i = 1
    q.enque
  for i = 1
    q.deque
}
```

```
int array[];
int capacity, size;
int front, back;

// add object x to queue
void enqueue(x){
  if (size >= capacity)
     throw error;
  array[back] = x;
  size++; back++;
}

// remove object from queue
object dequeue(){
  if (size <= 0)
     throw error;
  return array[front];
  size--; front++;
}
          ...
```

program + tests

```
enqueue(x):::ENTER
size = 0
front = 0
back = -1
capacity = 1
enqueue(x):::EXIT
size = 1
front = 0
back = 0
capacity = 1
dequeue():::ENTER
size = 1
front = 0
back = 0
capacity = 1
dequeue():::EXIT
size = 0
front = 1
back = 0
capacity = 1
enqueue(x):::ENTER
size = 0
front = 0
back = -1
capacity = 5
enqueue(x):::EXIT
size = 1
front = 0
back = 0
capacity = 5
enqueue(x):::ENTER
size = 1
front = 0
back = 0
capacity = 5
enqueue(x):::EXIT
size = 2
front = 0
back = 1
capacity = 5
        ...
```

data trace

```
enqueue(x):::ENTER
size <= capacity
front <= capacity
back <= capacity
```

```
dequeue():::ENTER
front <= back
size <= capacity
back <= capacity
```

```
enqueue(x):::EXIT
size <= capacity
return <==> (size < capacity)
```

data invariants

```
enqueue(x):::ENTER
enqueue(x):::EXIT
dequeue():::ENTER
dequeue():::EXIT
enqueue(x):::ENTER
enqueue(x):::EXIT
enqueue(x):::ENTER
enqueue(x):::EXIT
enqueue(x):::ENTER
enqueue(x):::EXIT
enqueue(x):::ENTER
enqueue(x):::EXIT
enqueue(x):::ENTER
enqueue(x):::EXIT
dequeue():::ENTER
dequeue():::EXIT
dequeue():::ENTER
dequeue():::EXIT
dequeue():::ENTER
dequeue():::EXIT
dequeue():::ENTER
dequeue():::EXIT
dequeue():::ENTER
dequeue():::EXIT
      ...
```

program point sequence

```
size <= capacity
front <= capacity
back <= capacity
..
size <= capacity
return <==> (size <
capacity)
..
front <= back
size <= capacity
back <= capacity
..
        ...
```

data-temporal trace

G(x)

spec type

G("size <= capacity")

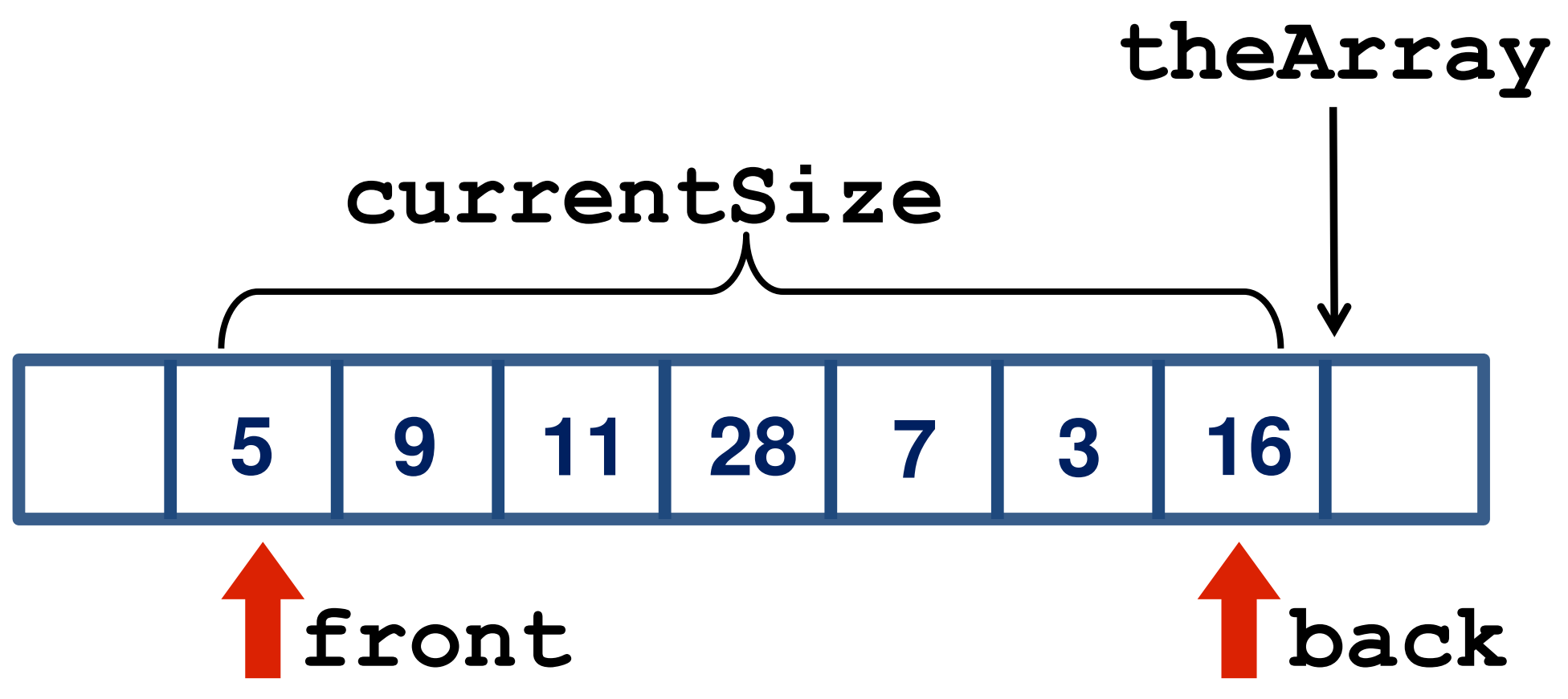
spec instantiation(s)

Daikon: infers likely data invariants at program points by instrumenting program and observing program runs

Texada: mines instances of any given LTL property which hold on the given log (set of traces – i.e. event sequences)

Preliminary evaluation

Ran Quarry on QueueAr, a queue implemented with a wrap-around array.



`this.currentSize == 0` always precedes `this.currentSize >=1`

- Holds because queue is constructed empty
- Confirms expected behaviour of test suite

`this.currentSize == 0` never occurs until `this.theArray[] elements == null`

- Queue constructed with null elements
- Elaborates how queue is initially created empty

`this.currentSize == this.front` is always followed by `this.currentSize == 0`

`this.currentSize >= 1` is never followed by `this.currentSize == this.back`

- Both are initialization invariants
- Temporal connectives provide essential context

Ongoing work 1

Does "`size >= 3`" always hold on this trace?

current string semantics: no

`size >= 3` and `size == 4` are different strings

```
size >= 3
..
size >= 3
..
size == 4
..
size >= 3
..
```

data invariant semantics: yes

`size == 4` is stronger than `size >= 3`

Future work: incorporate SMT/theorem proving tools

Ongoing work 2

Quarry mined 100s of spec instances on QueueAr

Future work: design interestingness filter

Quarry confidence measure ⟺ ? Daikon confidence measure + Texada confidence measure

[1] M. P. Robillard, et al. Automated API property inference techniques. *TSE*, 613-637, 2013.
[2] M. D. Ernst, et al. Dynamically discovering likely program invariants to support program evolution. *TSE*, 27(2):99–123, 2001.
[3] V Dallmeier, et al. Generating Test Cases for Specification Mining. *ISSTA*, 85-96, 2010.
[4] I. Beschastnikh, et al. Leveraging existing instrumentation to automatically infer invariant-constrained models. *FSE*, 267–277, 2011.