Correct or usable?
The Limits of Traditional Verification

Daniel Jackson[1] & Mandana Vaziri[2]

[1]Massachusetts Institute of Technology
[2]IBM T.J. Watson Research Center

FSE'16

Linked List Deletion

Is this correct?

```
class List {
    List next;
    Val val;
    void delete (Val v) {
        List l = this;
        List prev = null;
        while (l != NULL)
            if (l.val == v) {
              prev.next = l.next ;
              return;
            } else {
              prev=l;
              l=l.next;
            }
}
```

Linked List Deletion

Writing tests

```
class List {
    List next;
    Val val;
    void delete (Val v) {
        List l = this;
        List prev = null;
        while (l != NULL)
            if (l.val == v) {
              prev.next = l.next ;
              return;
            } else {
              prev=l;
              l=l.next;
            }
}
```

Linked List Deletion

Writing a proof

```
class List {
    List next;
    Val val;
    void delete (Val v) {
        List l = this;
        List prev = null;
        while (l != NULL)
            if (l.val == v) {
              prev.next = l.next ;
              return;
            } else {
              prev=l;
              l=l.next;
            }
}
```

Linked List Deletion

Our Approach

Given correctness spec, exhaustively check finitized code

```
class List {
    List next;
    Val val;
    void delete (Val v) {
        List l = this;
        List prev = null;
        while (l != NULL)
            if (l.val == v) {
              prev.next = l.next ;
              return;
            } else {
              prev=l;
              l=l.next;
            }
}
```

Finitized code?

Consider small finite heap

Unroll loops N times

```
class List {
    List next;
    Val val;
    void delete (Val v) {
        List l = this;
        List prev = null;
        while (l != NULL)
            if (l.val == v) {
              prev.next = l.next ;
              return;
            } else {
              prev=l;
              l=l.next;
            }
}
```

Correctness specification?

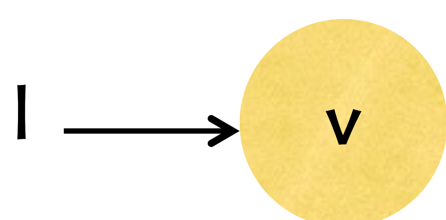
No cell with value v after

*no c: l.*next′ | c.val′ = v*

```
class List {
    List next;
    Val val;
    void delete (Val v) {
        List l = this;
        List prev = null;
        while (l != NULL)
            if (l.val == v) {
              prev.next = l.next ;
              return;
            } else {
              prev=l;
              l=l.next;
            }
}
```

Counterexample: Bug 1

First cell is never deleted




```
class List {
      List next;
      Val val;
      void delete (Val v) {
            List l = this;
            List prev = null;
            while (l != NULL)
                  if (l.val == v) {
                    prev.next = l.next ;
                    return;
                  } else {
                    prev=l;
                    l=l.next;
                  }
}
```

Add a precondition

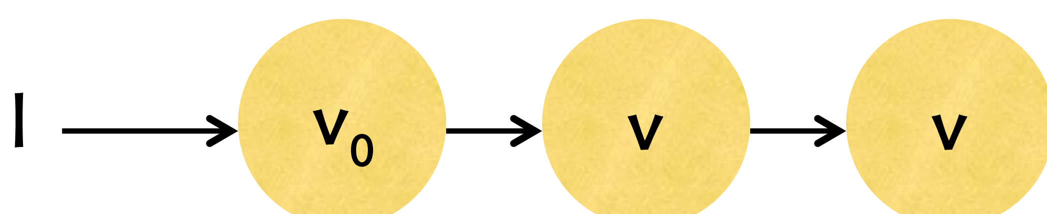
v does not occur in the first cell

l.val != v

```
class List {
    List next;
    Val val;
    void delete (Val v) {
        List l = this;
        List prev = null;
        while (l != NULL)
            if (l.val == v) {
              prev.next = l.next ;
              return;
            } else {
              prev=l;
              l=l.next;
            }
}
```

Counterexample: Bug 2

Not all values are deleted




```
class List {
    List next;
    Val val;
    void delete (Val v) {
        List l = this;
        List prev = null;
        while (l != NULL)
            if (l.val == v) {
              prev.next = l.next ;
              return;
            } else {
              prev=l;
              l=l.next;
            }
}
```

Rep invariant

No duplicates

*all x | lone cell: l.*next | cell.val = x*

```
class List {
    List next;
    Val val;
    void delete (Val v) {
        List l = this;
        List prev = null;
        while (l != NULL)
            if (l.val == v) {
              prev.next = l.next ;
              return;
            } else {
              prev=l;
              l=l.next;
            }
}
```

No counterexample found!

```
class List {
    List next;
    Val val;
    void delete (Val v) {
        List l = this;
        List prev = null;
        while (l != NULL)
            if (l.val == v) {
              prev.next = l.next ;
              return;
            } else {
              prev=l;
              l=l.next;
            }
}
```

How does it work?

Obtain a logical formula representing all possible paths in the finite code

formula ∧ !spec

Satisfying assignment is an input and path that violate spec

Solve using Alloy, based on SAT solving

Pros & Cons

✔ No false positives

✔ Declarative spec

✔ Refine spec to gain program understanding

✗ There is more to correctness than logical specs

Performance

Productivity

Usability

✗ Hard to manage two artifacts

Code *and* spec

Unrealistic in agile development

Wishful Thinking

Have a single artifact

Specify behavior and get correctness by construction

Single artifact should allow exploration!

Straight-line code is hard to understand

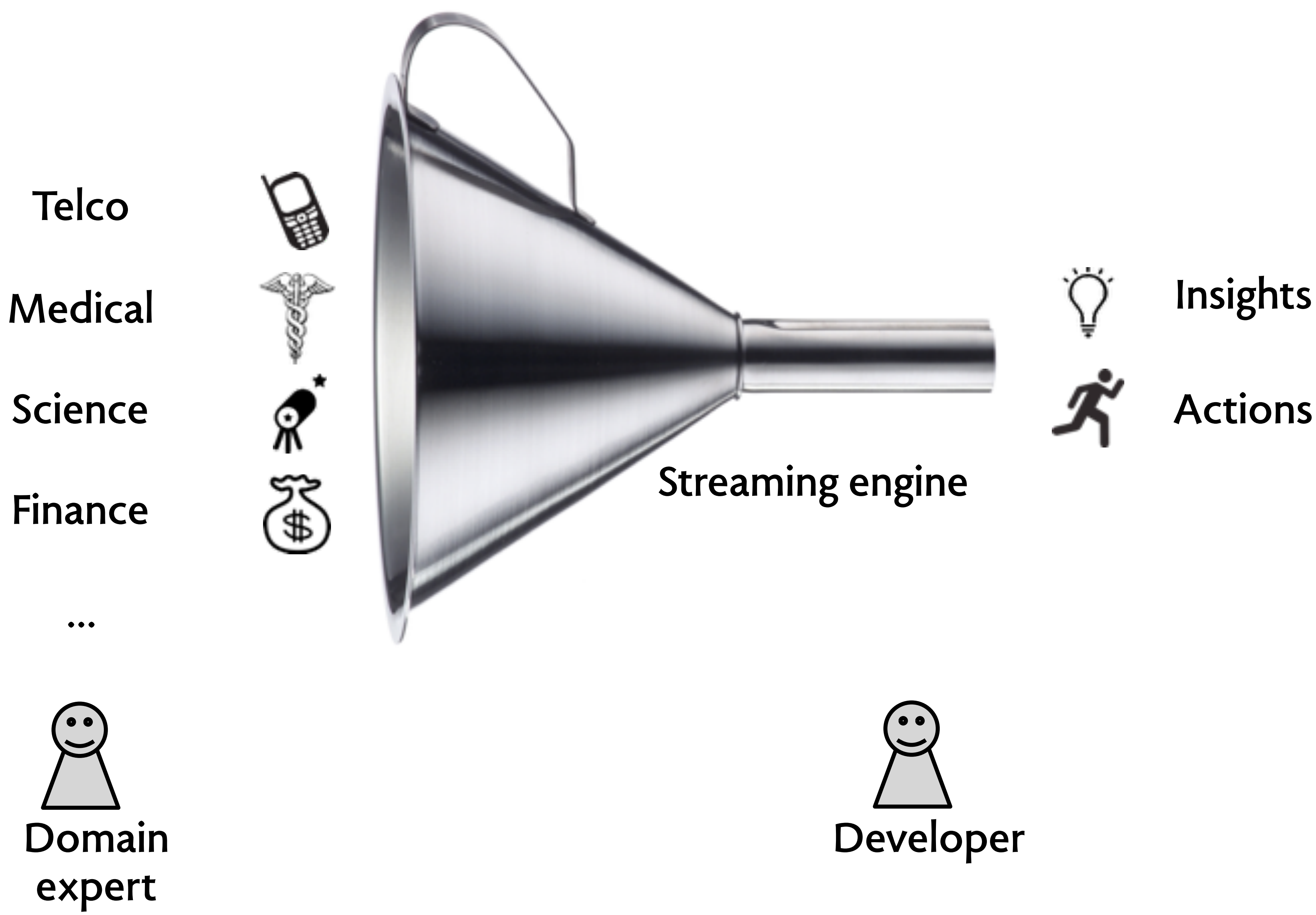
Requires other tools

Better to have a real Swiss army knife than a picture!

ActiveSheets

Stream processing with a spreadsheet [ECOOP'14]

Enable domain experts to write executable specs

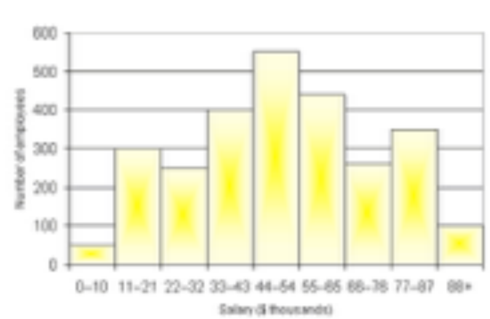


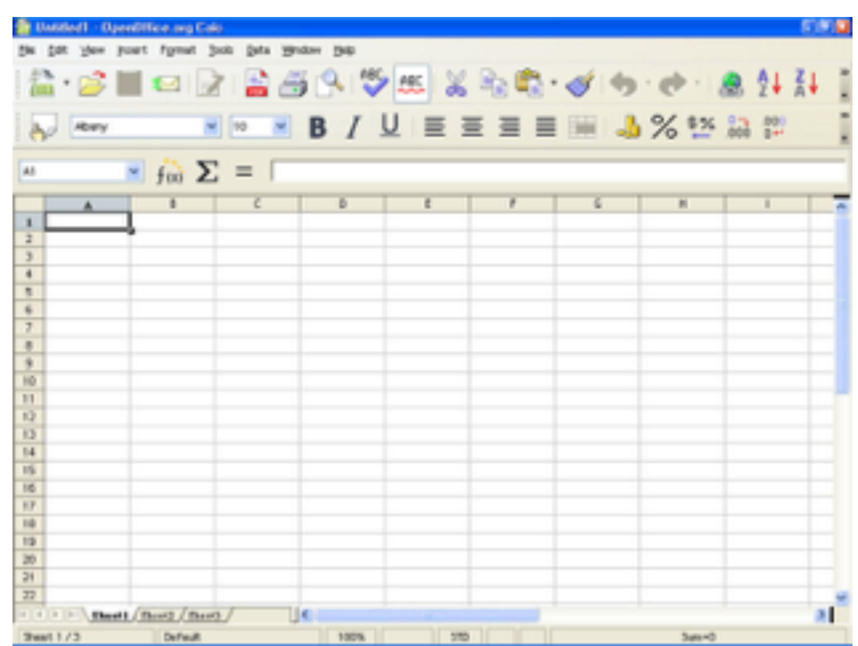
Why spreadsheets?

Easy-to-use, pervasive interface

500 million MS Excel users vs 10 million Java users (sources: mrexcel.com, wikipedia)

Fluidity between code and data





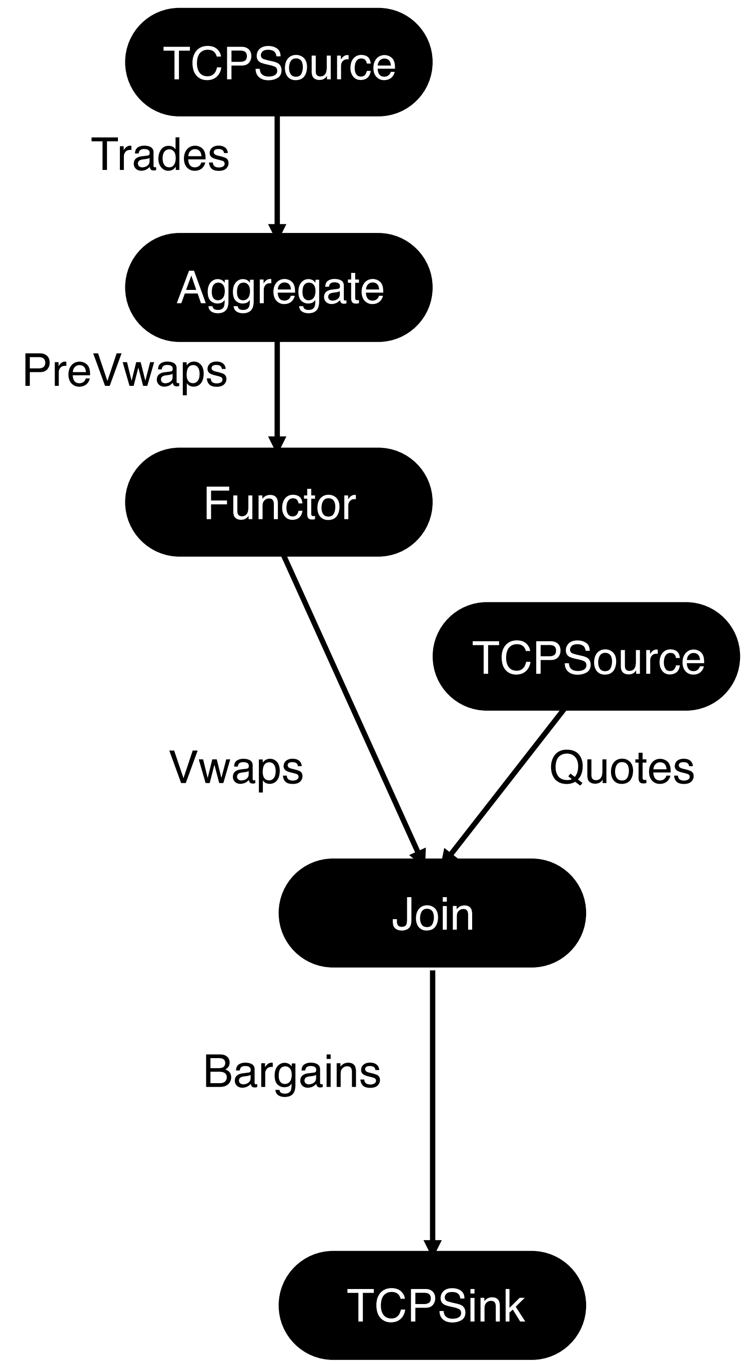
Example from finance

Determine bargains from stock quotes

Quotes compared to volume-weighted average price (VWAP), and output if lower

$$\text{vwap} = \frac{\Sigma\, \text{price*vol}}{\Sigma\, \text{vol}}$$

```
composite BargainFinder {
 type
  TradeOrQuote
  PreVwap       = tuple<rstring sym, timestamp ts, decimal64 price, decimal64 vol>;
  Vwap          = tuple<rstring sym, timestamp ts, decimal64 priceVol, decimal64 vol>;
  Bargain       = tuple<rstring sym, timestamp ts, decimal64 vwap>;
 graph          = tuple<rstring sym, timestamp ts, decimal64 index>;
  stream<TradeOrQuote> Trades = TCPSource() {
    param  role              : server;
  }        port              : 40000u;

 stream<PreVwap> PreVwaps = Aggregate(Trades) {
    window Trades            : sliding, delta(ts, 60.0), count(1), partitioned;
    param  partitionBy       : sym;
    output PreVwaps          : priceVol = Sum(price*vol), vol = Sum(vol);
  }

 stream<Vwap> Vwaps = Functor(PreVwaps) {
    output Vwaps             : vwap = priceVol / vol;
  }

 stream<TradeOrQuote> Quotes = TCPSource() {
    param  role              : server;
  }        port              : 40001u;

  stream<Bargain> Bargains = Join(Vwaps; Quotes) {
    window Vwaps             : sliding, count(1), partitioned;
           Quotes            : sliding, count(0);

    param
           equalityLHS       : Vwaps.sym;
           equalityRHS       : Quotes.sym;
    output partitionByLHS    : Vwaps.sym;
  }
           Bargains          : index = vwap > price
 () as Sink = TCPSink(Bargains) {   ? price * exp(vwap - price) : 0d;
    param
           role              : client;
           address           : "10.0.0.2";
  }}       port              : 40002u;
```



VWAP in ActiveSheets

	A	B	C	D
1	input Trades			
2	sym	ts	price	vol
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16	"IBM"	"Mon Sep	194.77	2,740
17	"IBM"	"Mon Sep	195.13	2,141
18	"IBM"	"Mon Sep	195.56	2,539
19	"IBM"	"Mon Sep	197.96	2,498
20	"IBM"	"Mon Sep	194.96	2,639
21	"IBM"	"Mon Sep	197.04	2,758
22	"IBM"	"Mon Sep	198.64	3,296
23	"IBM"	"Mon Sep	200.99	3,111
24				
25				
26				

Live

Client/Server architecture
- Server publishes streams
- Client (spreadsheet) can subscribe to them

Visualization of live data

Ability to pause and continue a live data stream

VWAP in ActiveSheets

	A	B	C	D	E	F	G	H	I	J	K
1	input Trades								output		
2	sym	price	vol				price*vol		VWAP		
3	"IBM"	194.77	2,740				533,670		197.77		
4	"IBM"	195.13	2,141				417,773				
5	"IBM"	195.56	2,539				496,527				
6	"IBM"	197.96	2,498				494,504				
7	"IBM"	194.96	2,639				514,499				
8	"IBM"	197.04	2,758				543,436				
9	"IBM"	198.64	3,296				654,717				
10	"IBM"	200.99	3,111				625,280				
11	"IBM"	200.69	2,335				468,611				
12	"IBM"	200.99	1,042				209,432				
13	"IBM"	198.77	744				147,885				
14	"IBM"	199.20	726				144,619				
15	"IBM"	199.07	842				167,617				
16	"IBM"	198.99	718				142,875				
17	"IBM"	197.70	773				152,822				
18	"IBM"	198.84	496				98,625				
19	"IBM"	198.16	424				84,020				
20	"IBM"	199.15	737				146,774				
21	"IBM"	198.71	664				131,943				
22	"IBM"	196.73	736				144,793				
23			sum				sum				
24			31,959				6,320,423				
25											
26											

=G24/C24

=SUM(C3:C22)

=SUM(G3:G22)

Use familiar gestures to compute new data

$$\text{vwap} = \frac{\Sigma\ \text{price*vol}}{\Sigma\ \text{vol}}$$

Live

VWAP in ActiveSheets

	A	B	C	D	E	F	G	H	I	J	K
1	input Trades				input Quotes				output		
2	sym	price	vol		price		price*vol		VWAP		
3	"IBM"	194.77	2,740		196.96		533,670		197.77		
4	"IBM"	195.13	2,141		=PROJECT(Quotes, pr=price)		417,773		=G24/C24		
5	"IBM"	195.56	2,539				496,527				
6	"IBM"	197.96	2,498				494,504		bargain?		
7	"IBM"	194.96	2,639				514,499		YES		
8	"IBM"	197.04	2,758				543,436		=IF(E3<I3,"YES","NO")		
9	"IBM"	198.64	3,296				654,717				
10	"IBM"	200.99	3,111				625,280				
11	"IBM"	200.69	2,335				468,611				
12	"IBM"	200.99	1,042				209,432				
13	"IBM"	198.77	744				147,885				
14	"IBM"	199.20	726				144,619				
15	"IBM"	199.07	842				167,617				
16	"IBM"	198.99	718				142,875				
17	"IBM"	197.70	773				152,822				
18	"IBM"	198.84	496				98,625				
19	"IBM"	198.16	424				84,020				
20	"IBM"	199.15	737				146,774				
21	"IBM"	198.71	664				131,943				
22	"IBM"	196.73	736				144,793				
23			sum				sum				
24			31,959				6,320,423				
25			=SUM(C3:C22)				=SUM(G3:G22)				
26											

Query language to obtain desired structures

Data export

Computation export

Programming model

Stateful computation

Reactive programming model

Live input streams are clocks into the spreadsheet

Cells are registers that get updated at each tick

Simple control structure

```
while(true) {
    await(tick);
    calculate_spreadsheet();
}
```

Benefits

Ease-of-use: No need to think about control
(no sequencing, no loops)

Domain expert manipulates data directly

Guarantees
Determinism
Bounded computation
and memory usage at each tick

Live Programming

Expressive for a range of stream applications

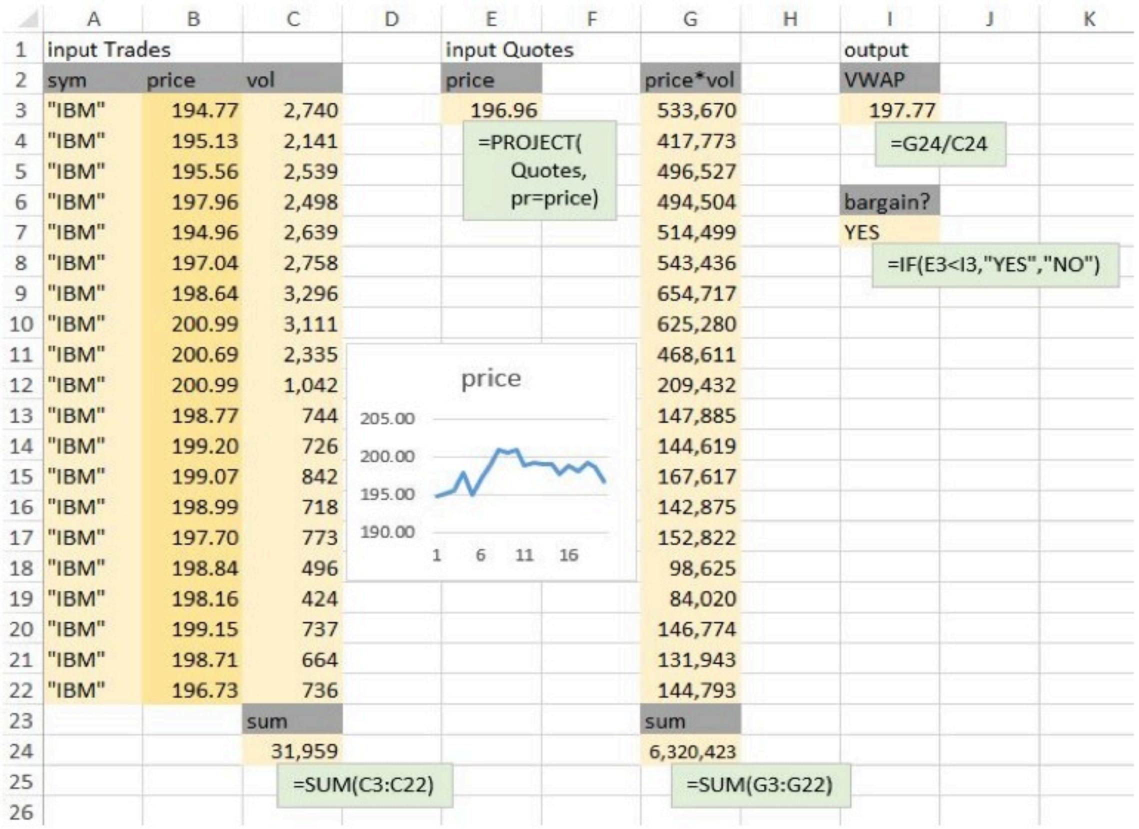
Correctness & Usability

Correctness by construction

Executable spec

Single artifact that is explorable

Meaningful to domain expert



	A	B	C	D	E	F	G	H	I	J	K
1	input Trades				input Quotes				output		
2	sym	price	vol		price		price*vol		VWAP		
3	"IBM"	194.77	2,740		196.96		533,670		197.77		
4	"IBM"	195.13	2,141		=PROJECT(Quotes, pr=price)		417,773		=G24/C24		
5	"IBM"	195.56	2,539				496,527				
6	"IBM"	197.96	2,498				494,504		bargain?		
7	"IBM"	194.96	2,639				514,499		YES		
8	"IBM"	197.04	2,758				543,436		=IF(E3<I3,"YES","NO")		
9	"IBM"	198.64	3,296				654,717				
10	"IBM"	200.99	3,111				625,280				
11	"IBM"	200.69	2,335				468,611				
12	"IBM"	200.99	1,042				209,432				
13	"IBM"	198.77	744				147,885				
14	"IBM"	199.20	726				144,619				
15	"IBM"	199.07	842				167,617				
16	"IBM"	198.99	718				142,875				
17	"IBM"	197.70	773				152,822				
18	"IBM"	198.84	496				98,625				
19	"IBM"	198.16	424				84,020				
20	"IBM"	199.15	737				146,774				
21	"IBM"	198.71	664				131,943				
22	"IBM"	196.73	736				144,793				
23			sum				sum				
24			31,959				6,320,423				
25			=SUM(C3:C22)				=SUM(G3:G22)				
26