Visual Denotative Programming

Steve Krouse

Why are spreadsheets great?

- Quick to learn
- Accessible to non-programmers
- Reached 100s of millions

The **combination** of

computational model **and** visual interface

Denotative programming

The Next 700 Programming Languages

P. J. Landin

Univac Division of Sperry Rand Corp., New York, New York

"... today ... 1,700 special programming languages used to 'communicate' in over 700 application areas."—*Computer Software* Issues, an American Mathematical Association Prospectus, July 1965.

A family of unimplemented computing languages is described that is intended to span differences of application area by a unified framework. This framework dictates the rules about the uses of user-coined names, and the conventions about characterizing functional relationships. Within this framework the design of a specific language splits into two inde-

differences in the set of things provided by the library or operating system. Perhaps had ALGOL 60 been launched as a family instead of proclaimed as a language, it would have fielded some of the less relevant criticisms of its deficiencies.

At first sight the facilities provided in ISWIM will appear

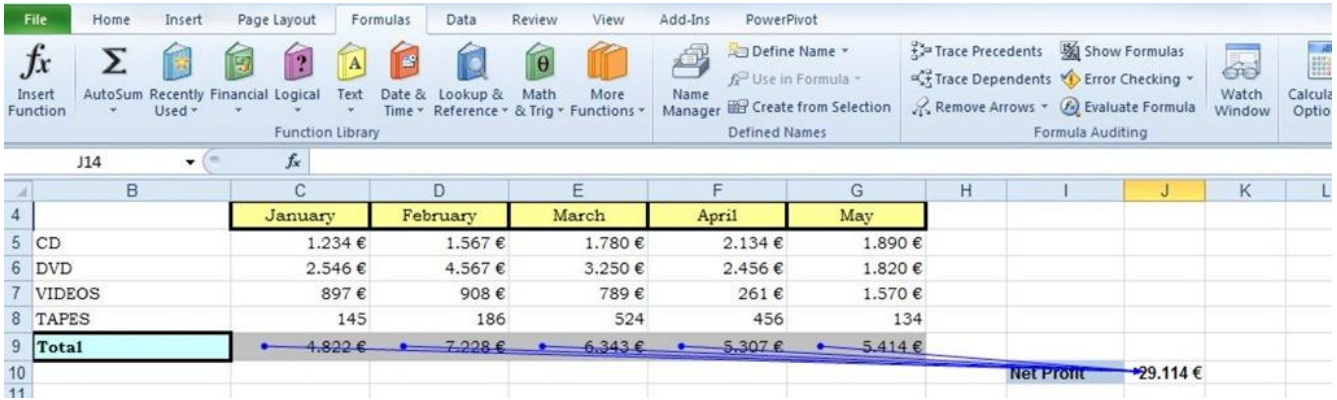
"The word "denotative" seems more appropriate than nonprocedural, declarative or functional."

Spreadsheets are denotative

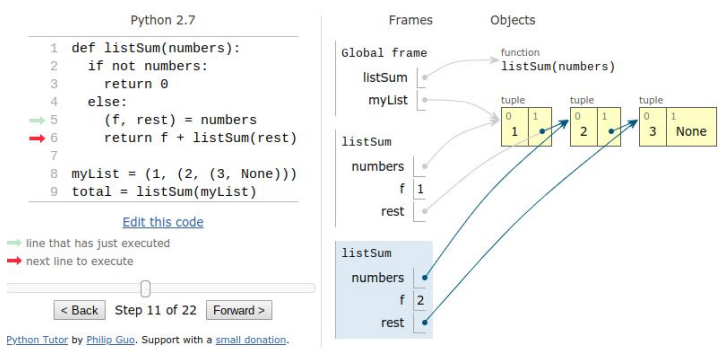
- a) No statements, only nested expressions
- b) Each expressions denotes a string, number or truth value
- c) Nothing affects a cell except its formula and its children, recursively...

Visual benefits

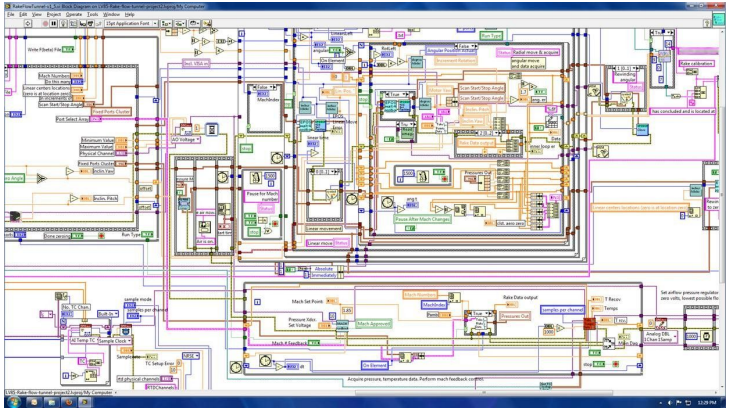
- Always running
- Data always visible
- Spatial metaphor
- Trace precedents



Non-denotative visualization: **control** flow

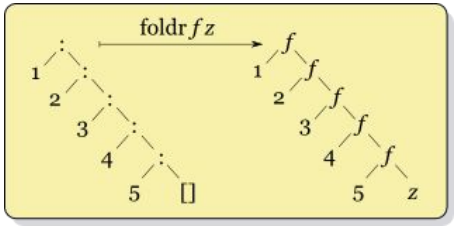


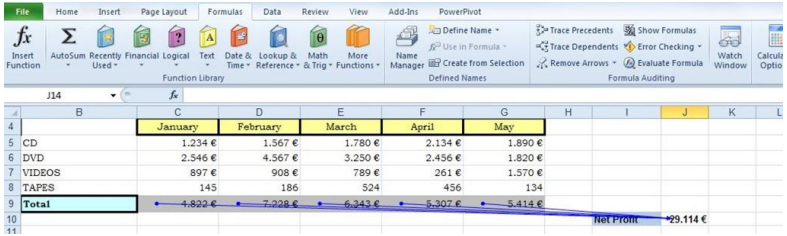
Source: pythontutor.com



Denotative visualizations: **data** flow

```
(foldr plus 0 [1,2,3]) (edit) (clear)
(plus 1 (foldr plus 0 [2,3]))
(plus 1 (plus 2 (foldr plus 0 [3])))
(plus 1 (plus 2 (plus 3 (foldr plus 0 []))))
(plus 1 (plus 2 (plus 3 0)))
(plus 1 (plus 2 (3 + 0)))
(plus 1 (2 + (3 + 0)))
(1 + (2 + (3 + 0)))
(1 + (2 + 3))
(1 + 5)
6
```



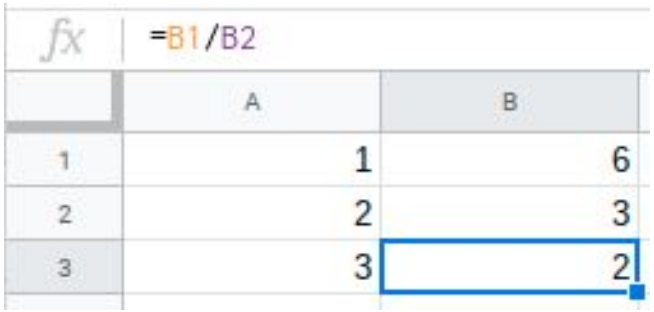


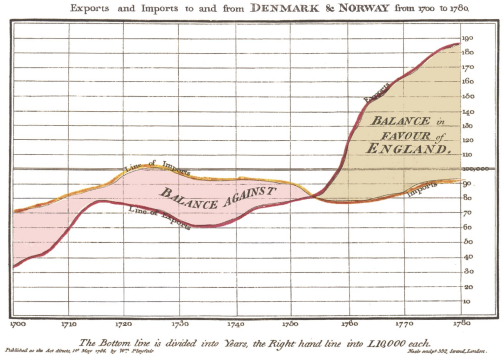
Denotative Programming Needs Visuals

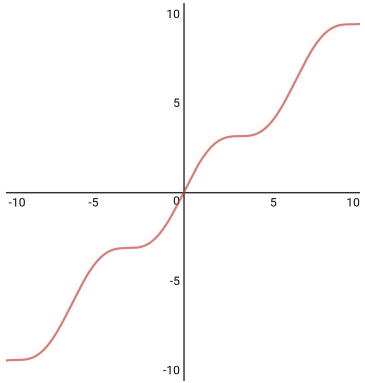
```
A1 = 1
A2 = 2
A3 = 3
B1 = SUM(A1:A3)
B2 = COUNT(A1:A3)
B3 = B1/B2
```

Type	Value	Year
Export	35	1700
Import	70	1700
Export	60	1710
Import	80	1710
Export	78	1720
Import	97	1720
Export	65	1730
Import	97	1730
Export	65	1740
Import	91	1740
Export	79	1750
Import	90	1750
Export	82	1760
Import	82	1760
Export	120	1770
Import	79	1770
Export	162	1780
Import	82	1780
Export	185	1790
Import	92	1790

```
f(x) = sin(x) + x
```







What about dynamism and reactivity?

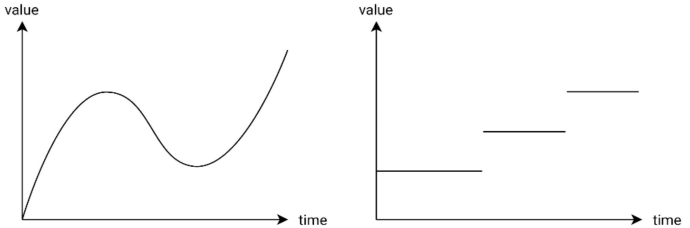
- User interfaces
- Robotics

Functional Reactive Programming

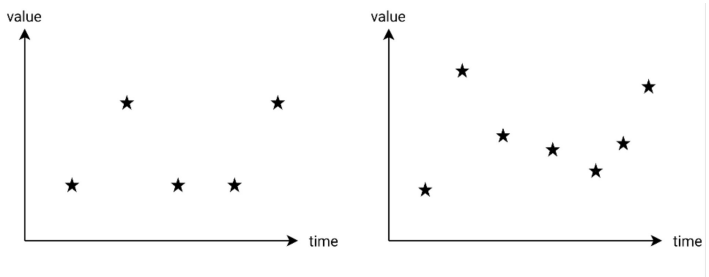
or

Denotative Continuous Time Programming

Behavior



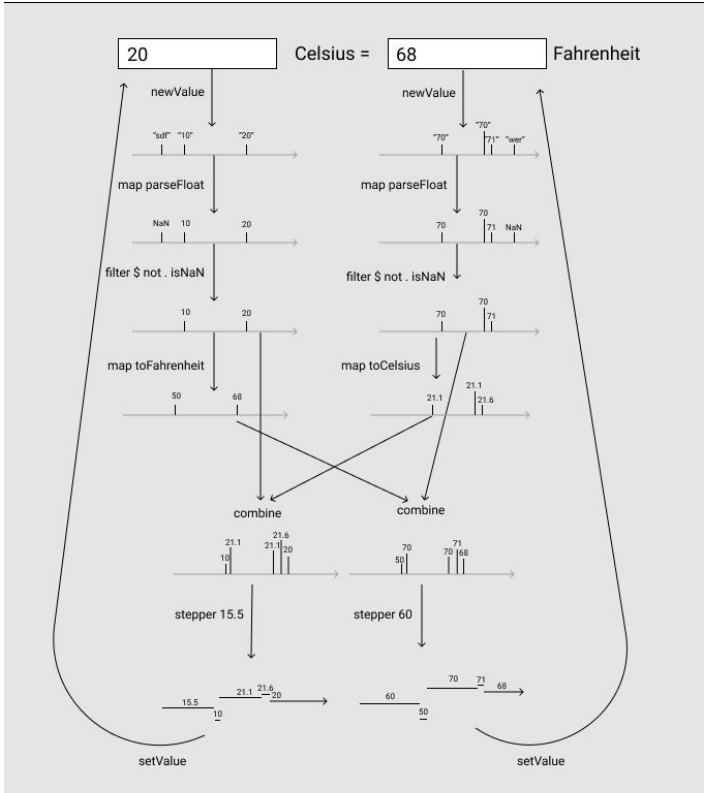
Event



Denotative UI Programming

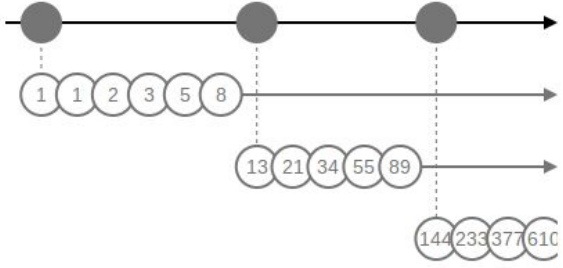
0 Celcius = 32 Fahrenheit

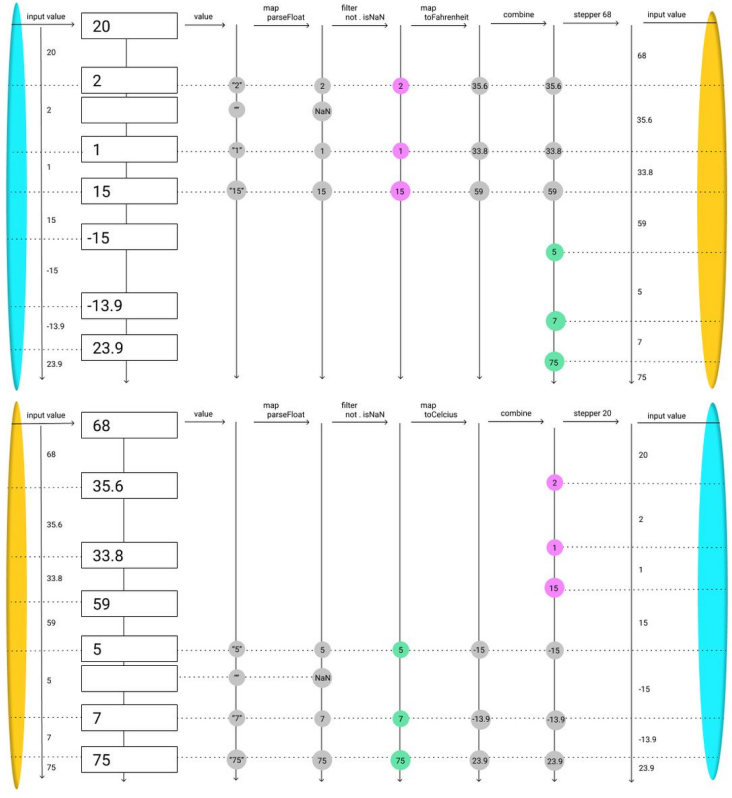
```
const temp = loop(
  ({ c, f }) =>
    function*() {
      const { value: cVal } = yield input({ value: c });
      yield text("Celcius = ");
      const { value: fVal } = yield input({ value: f });
      yield text("Fahrenheit");

      const [cValN, fValN] = [cVal, fVal].map(s =>
        changes(s)
          .map(parseFloat)
          .filter(n => !isNaN(n))
      );
      const fToC = f => (f - 32) / 1.8;
      const cToF = c => (c * 9) / 5 + 32;
      const initialC = 0;
      const initialF = cToF(initialC);

      const c_ = yield stepperFromComponentCreation(
        initialC,
        combine(cValN, fValN.map(fToC))
      );
      const f_ = yield stepperFromComponentCreation(
        initialF,
        combine(fValN, cValN.map(cToF))
      );

      return { c: c_, f: f_ };
```

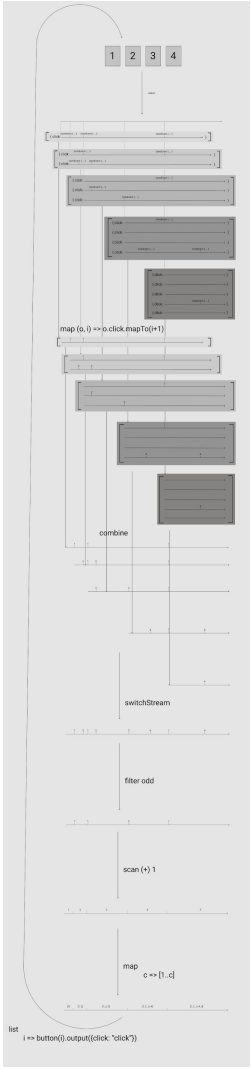


Challenges

- Higher order streams
- Cycles
- Stream dependency layout
- Informational density issues







power + simplicity = denotative + visual