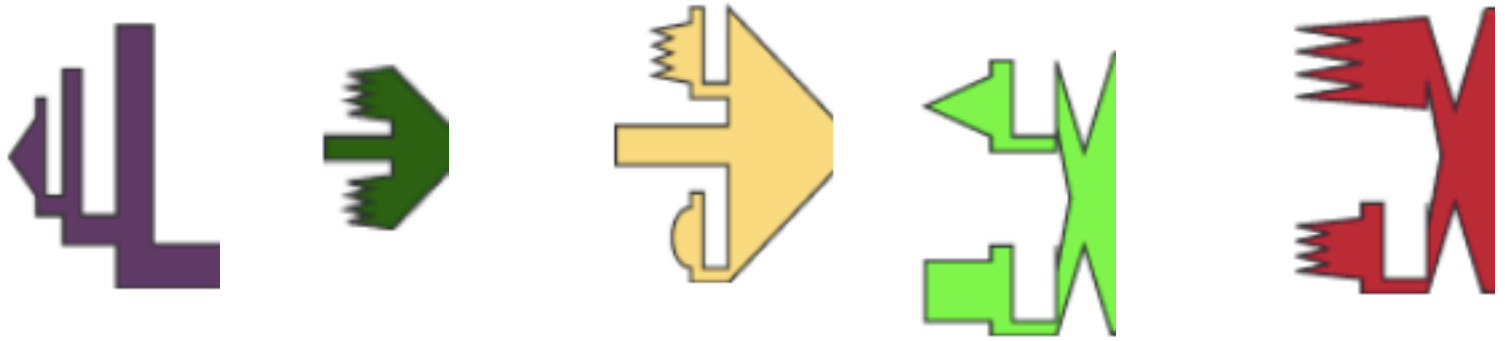
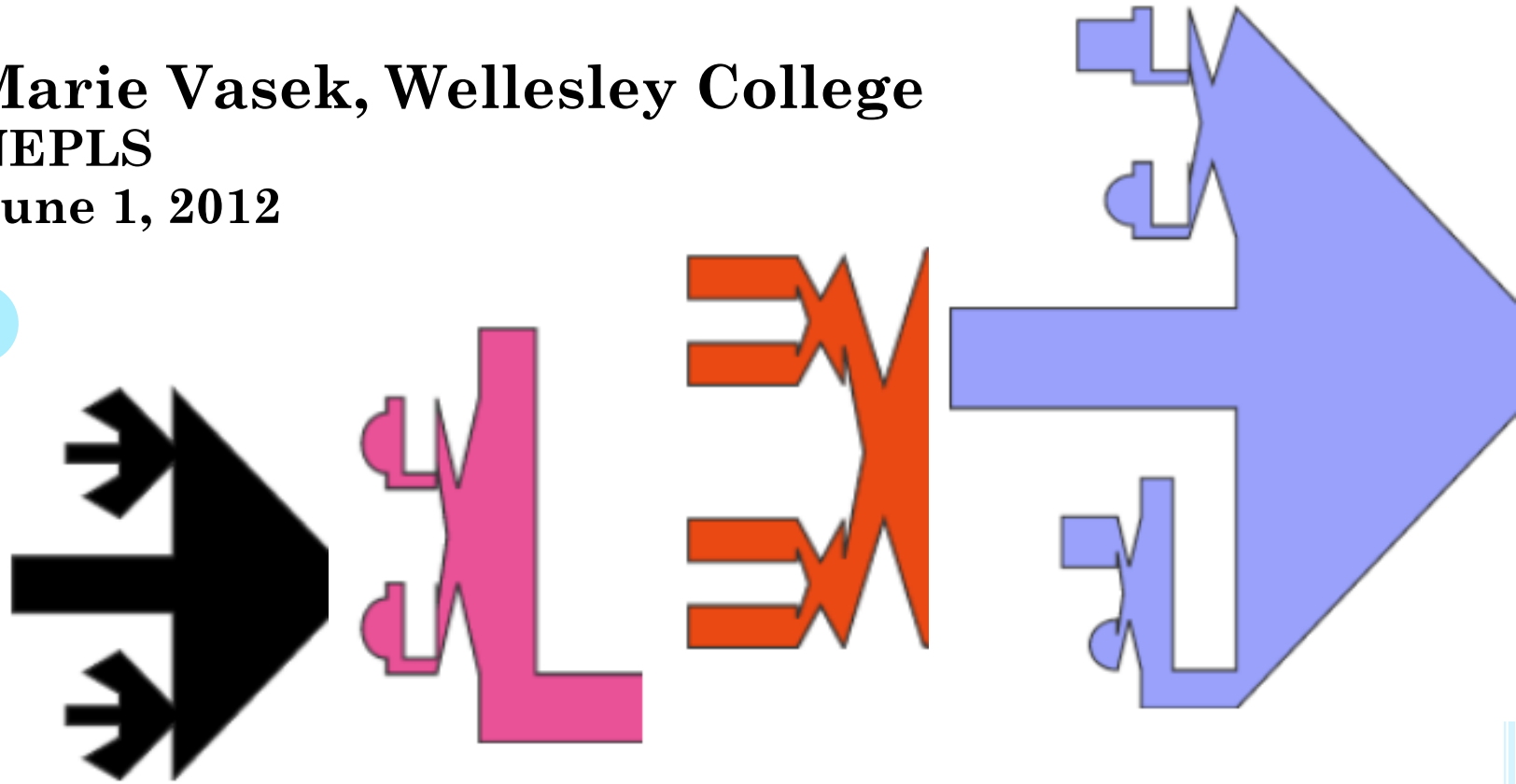


Representing Expressive Types In Blocks Programming Languages

Marie Vasek, Wellesley College

NEPLS

June 1, 2012



Representing types from functional languages in blocks languages

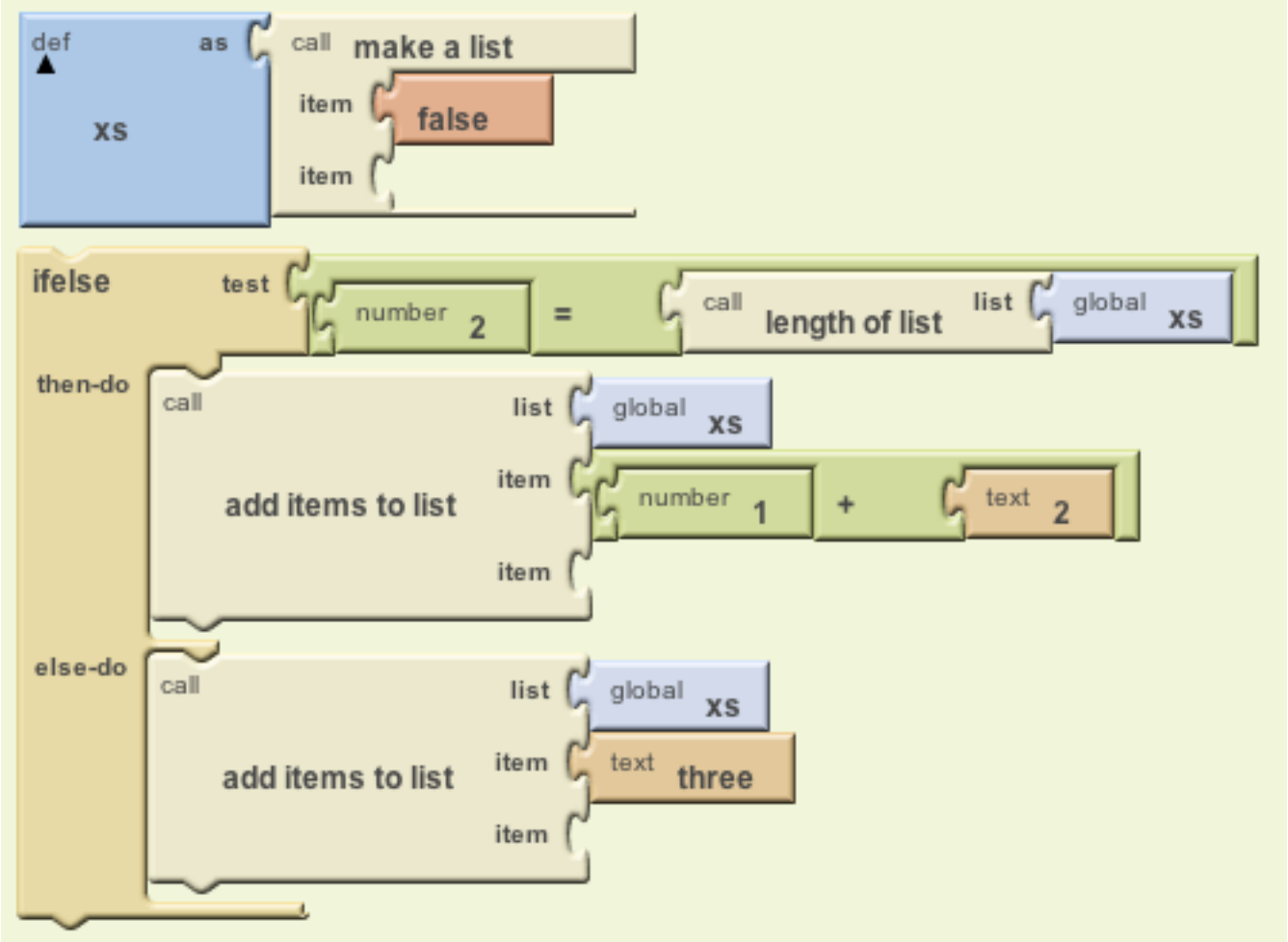
Problem: Current block languages aim to lower barriers to programming but only make weak attempts to implement static types and do not represent tree-structured types.

Solution: Create a blocks language where the shape of the block connector reflects the tree structure of a type.

Overview:

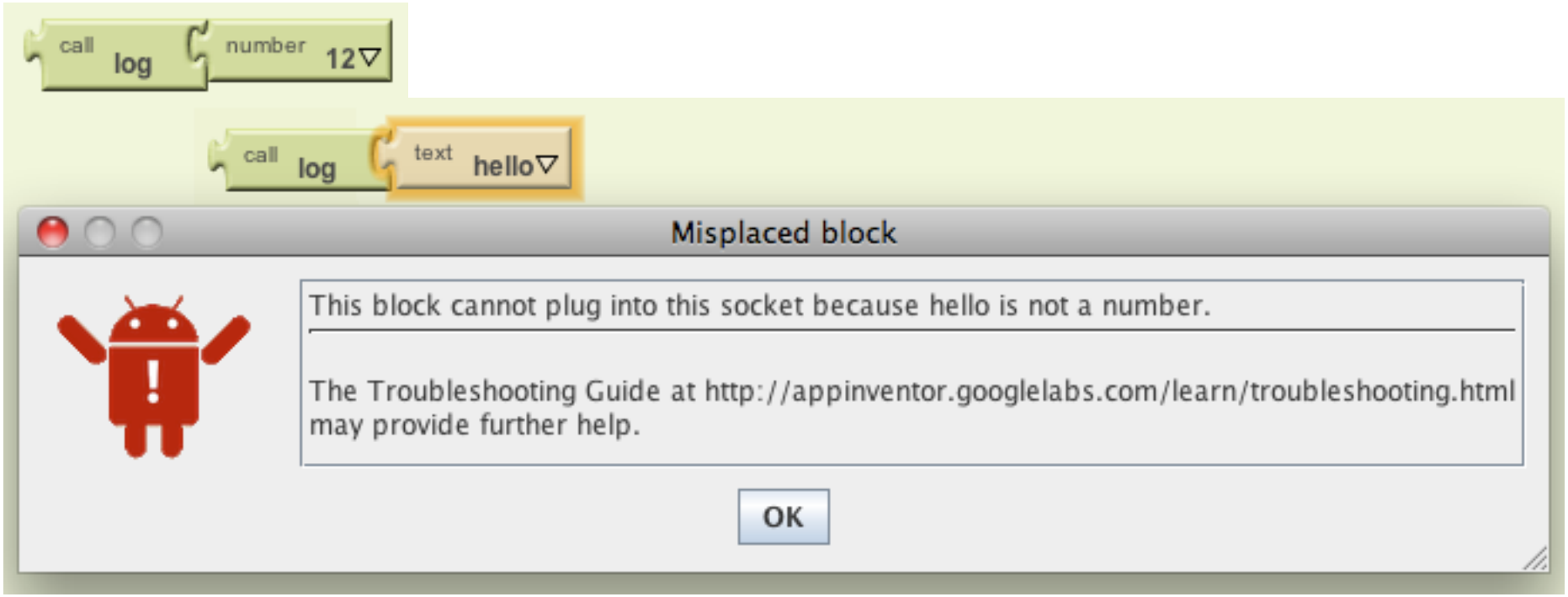
- Type systems in other blocks languages
- TypeBlocks:
  - Shape types
  - Polymorphism
  - Work in progress: functions, algebraic data types

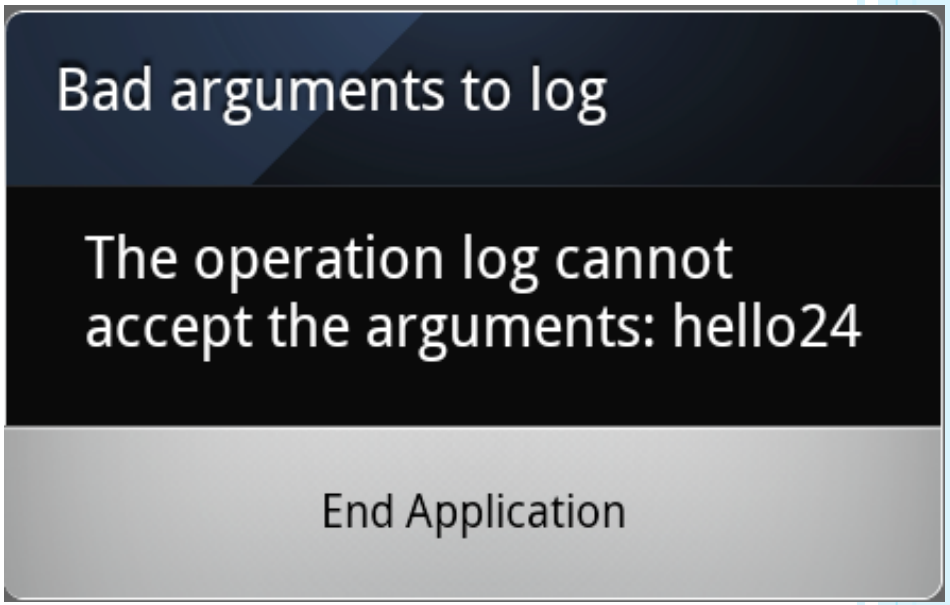
App Inventor – Dynamic-ish Typing



- All types have the same connector shape
- Most type checking at run time, but some at connection time

Timing of Errors

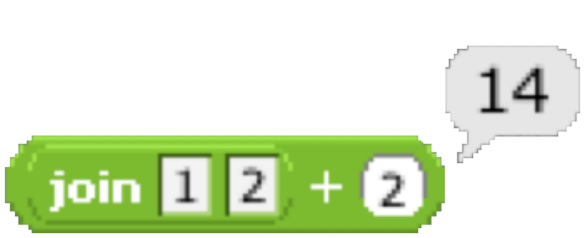


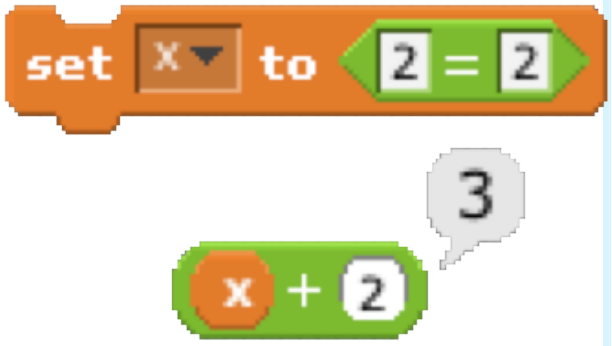


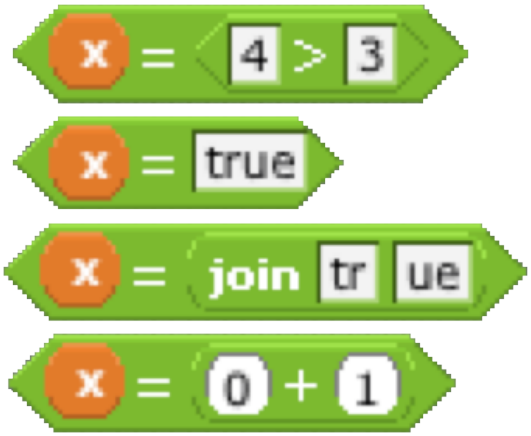
Scratch – Wonky Typing

- Three primitive types (boolean, string, number)
- Three shapes (angle = boolean, rounded = string or number, box = any)

Type Conversion





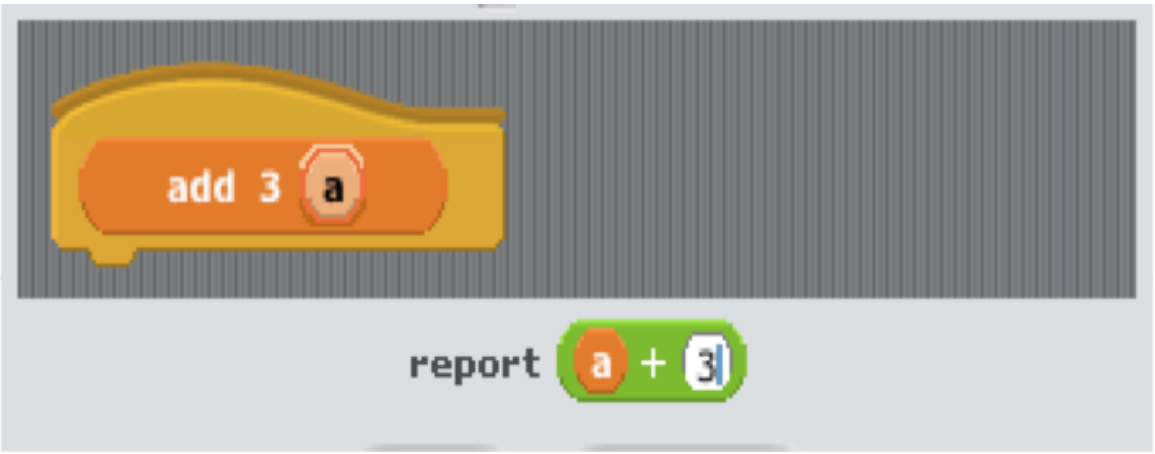


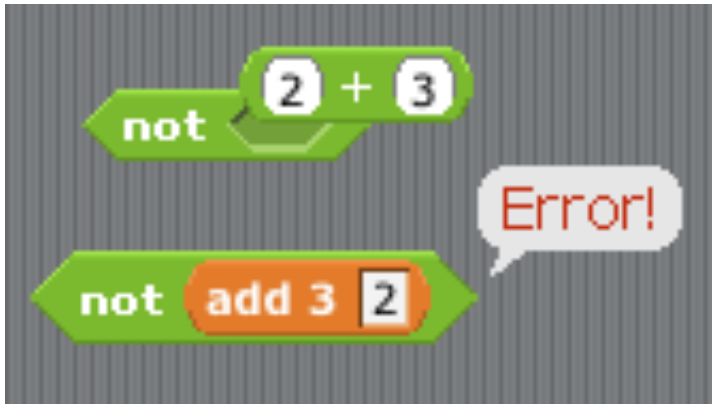
Evaluate to true

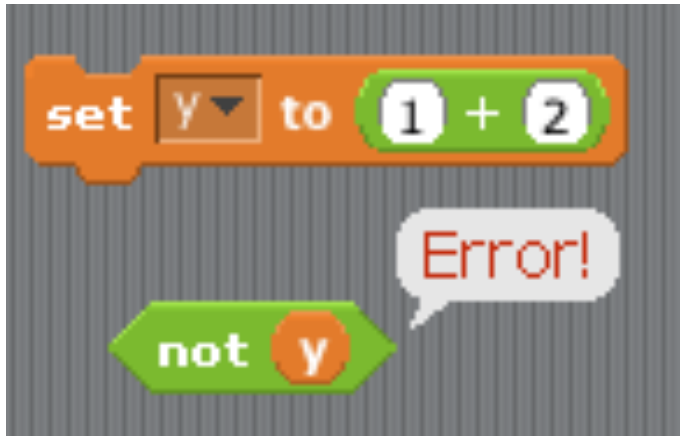
Evaluates to false

Type Conversion - Lists

BYOB – More Wonkiness

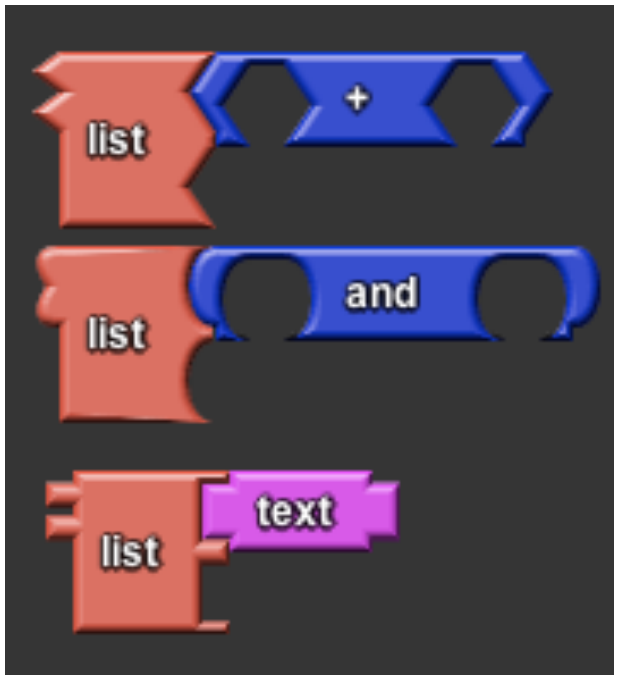


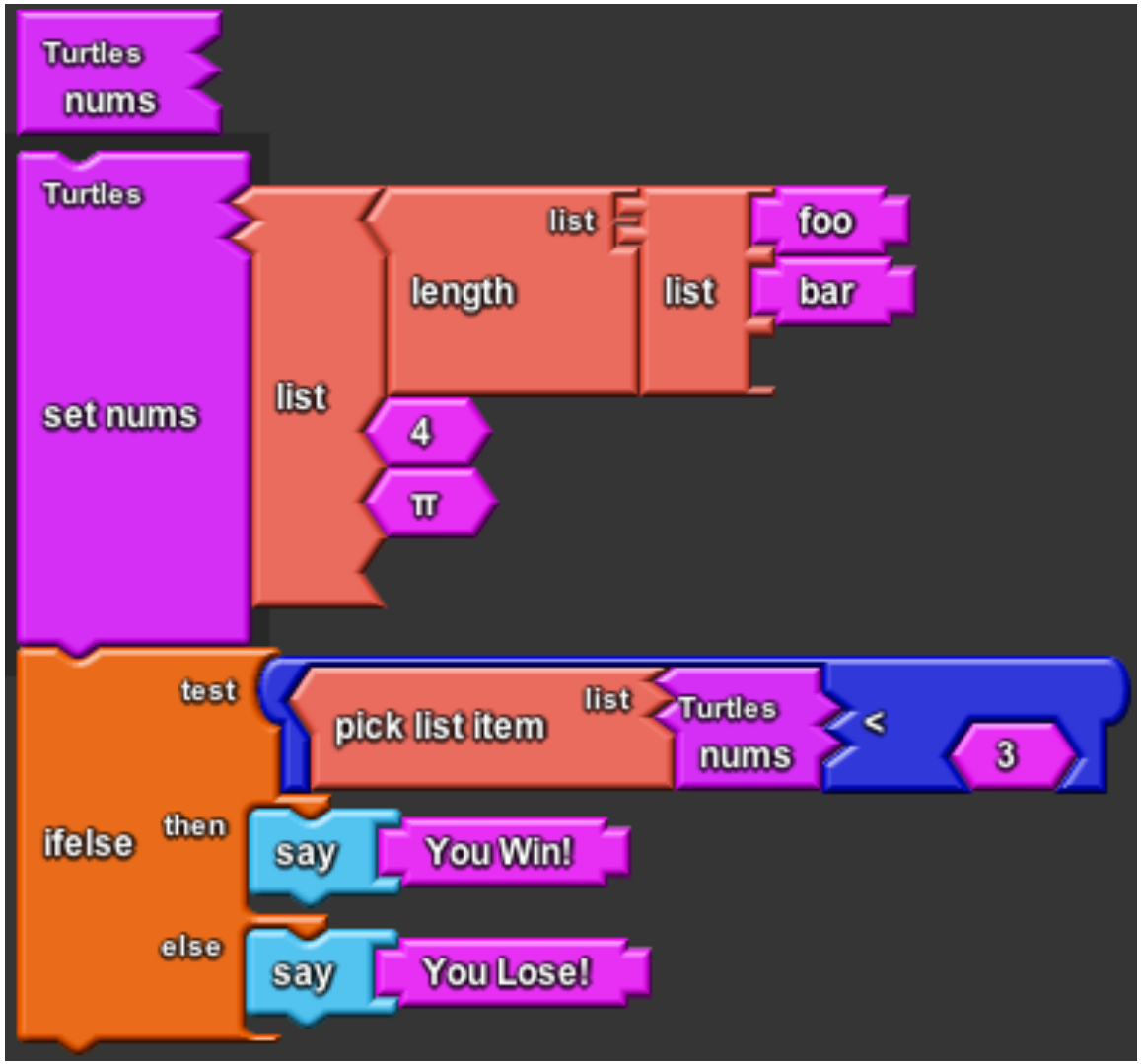




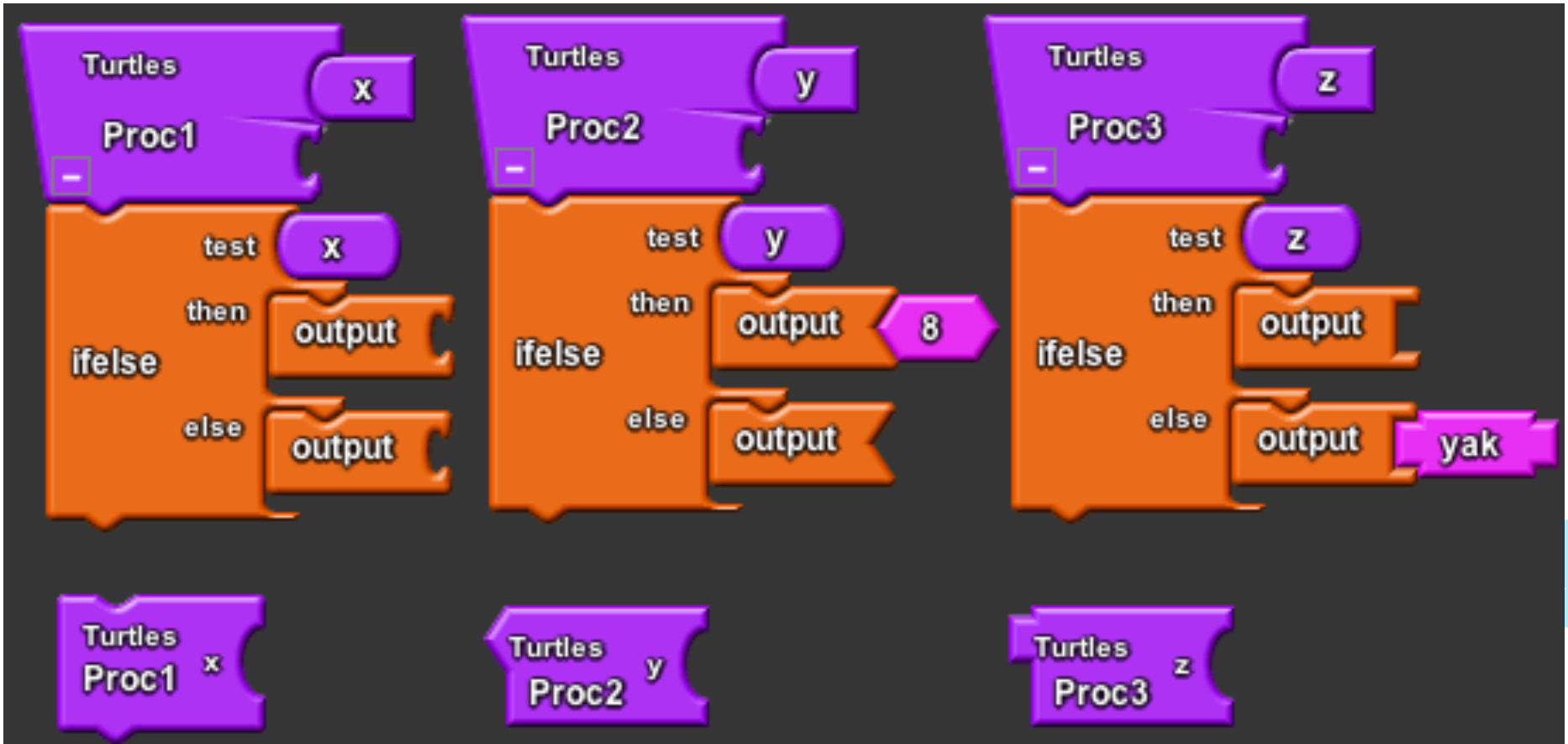
StarLogo: TNG

6 types:

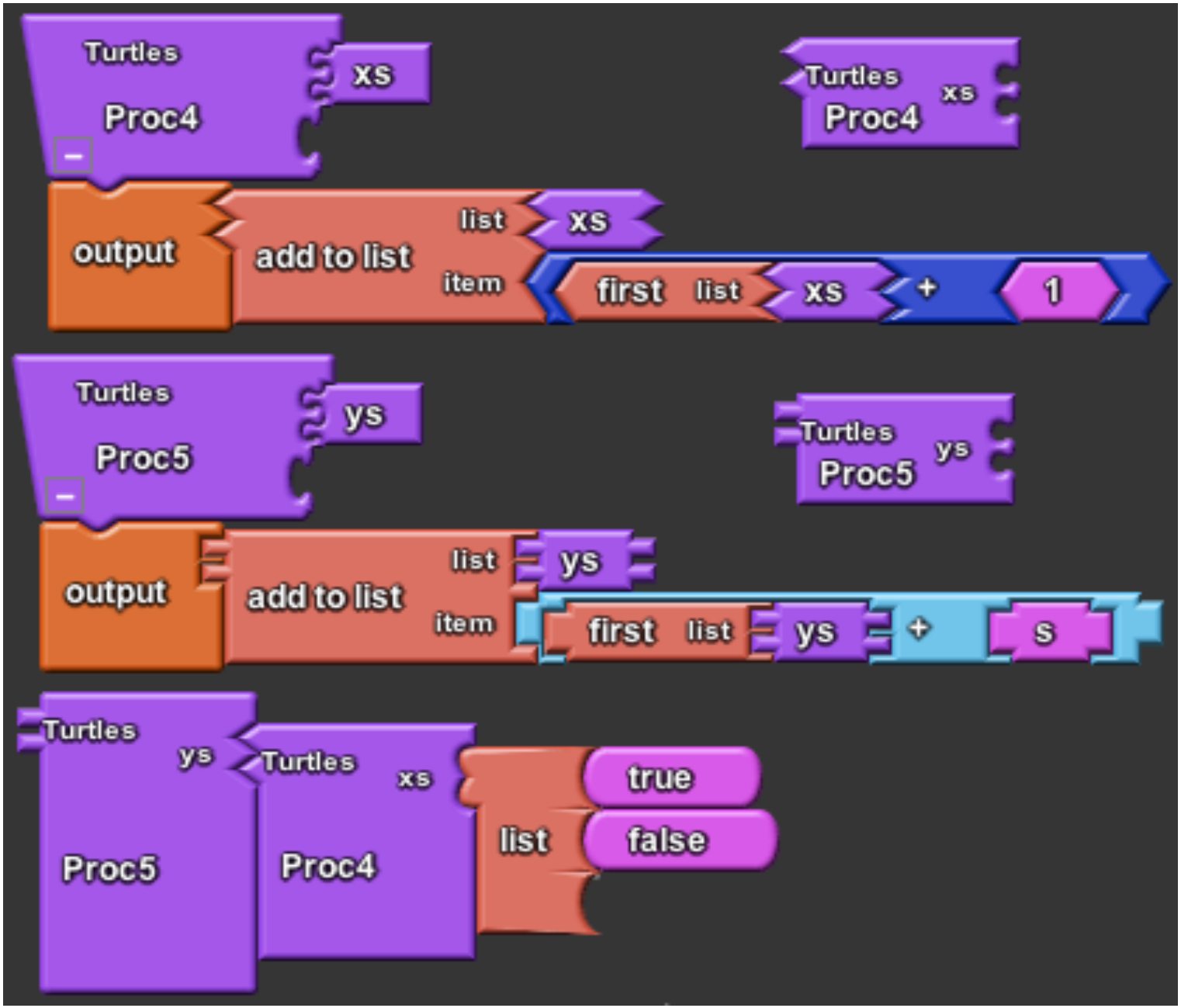




POLYMORPHISM



Buggy Procedure Typing



What I Did

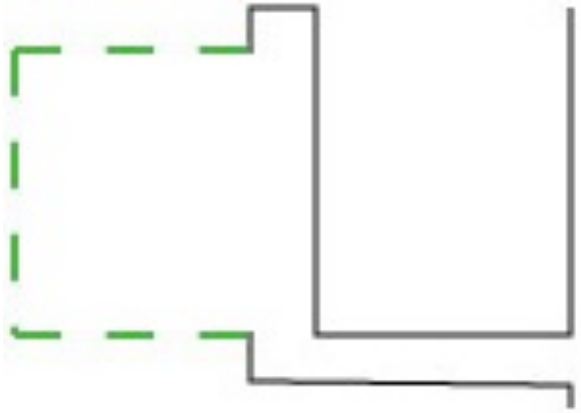
- Blocks types inspired by SML
- Base types + type constructors
  => ability to represent countably many types
- Each arbitrarily complex type = unique connector shape
- ML- style universal polymorphism
- … but no blocks language constructed from this yet
  - no functions or algebraic datatypes

Base Types

3 base types: `number, boolean, string`

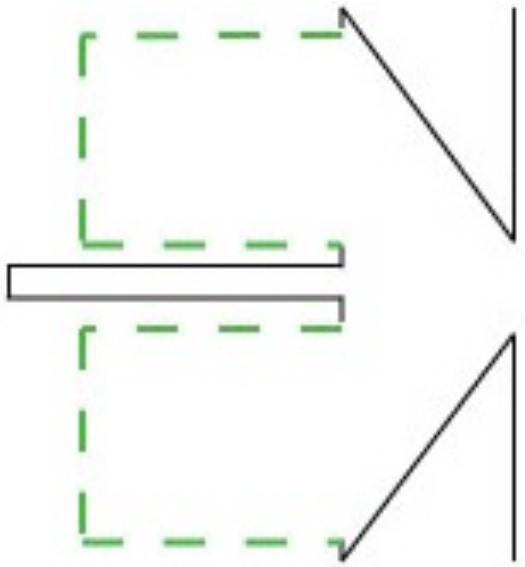
BUILD-A-TYPE

3 constructors:



`listof`

`pair`



`function`

`listof int`	`listof (listof string)`	`int * string`	`bool -> string`

More Example Plugs

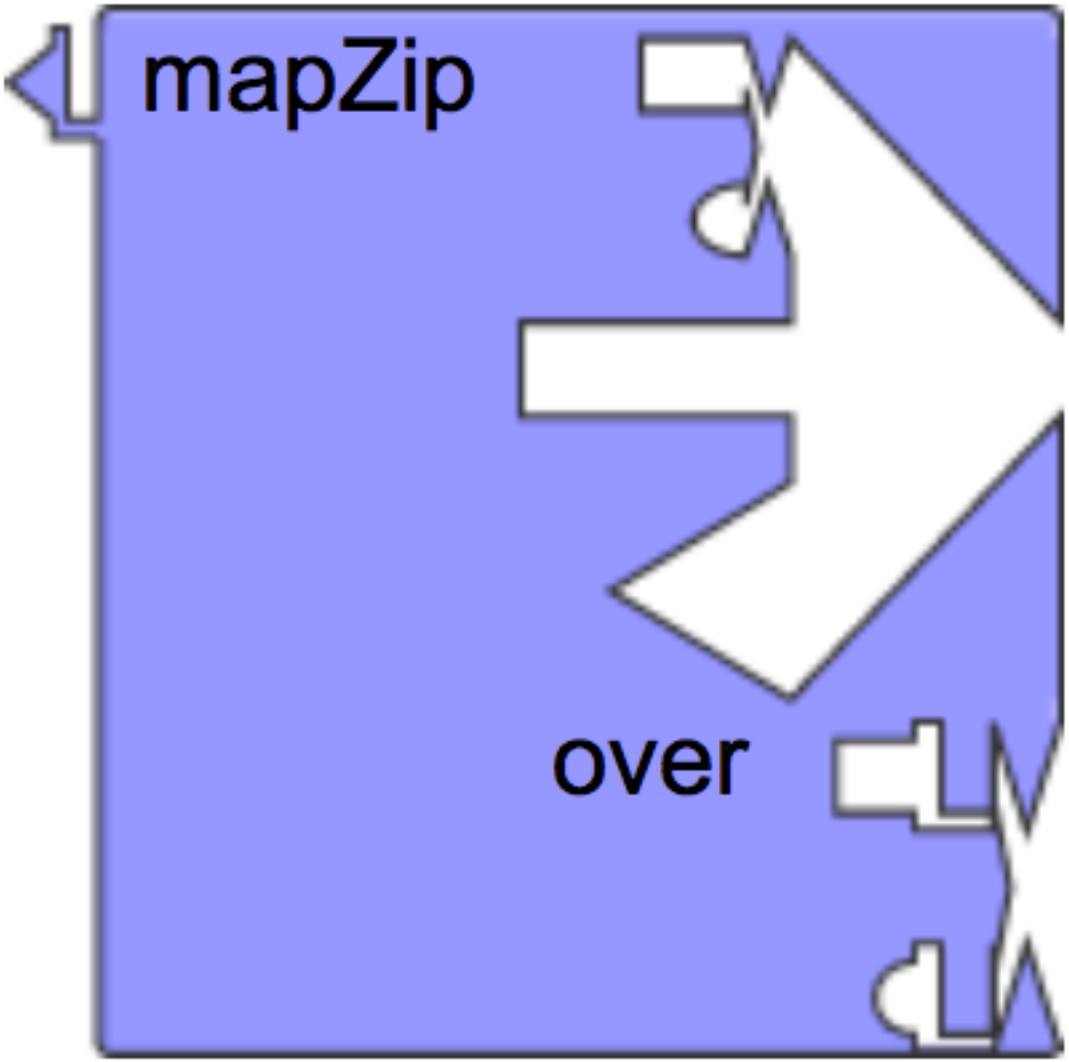
`listof (string * boolean)`

`(listof string) * boolean`

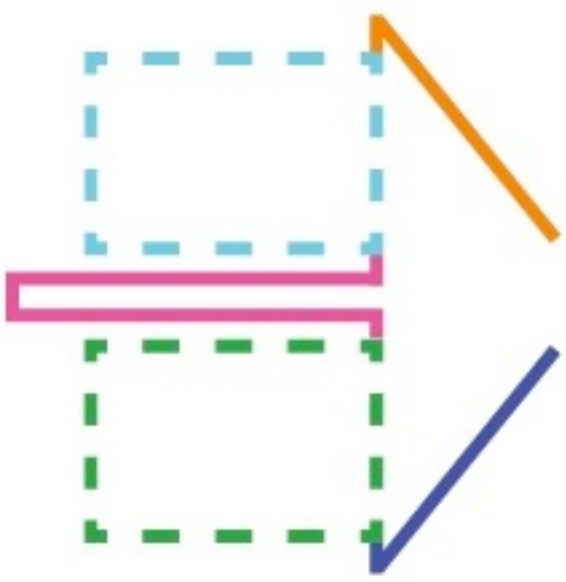
`boolean  * (string -> listof number)`

Pop Quiz

Zip and Map



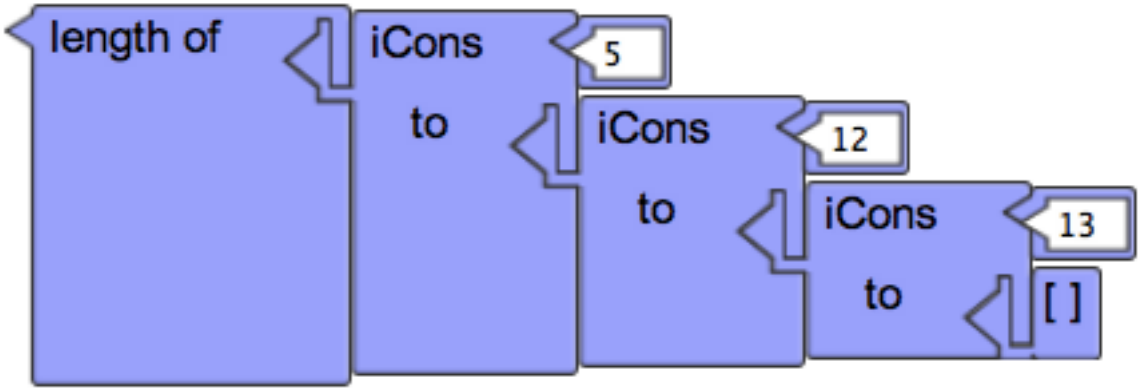
Types to Shapes



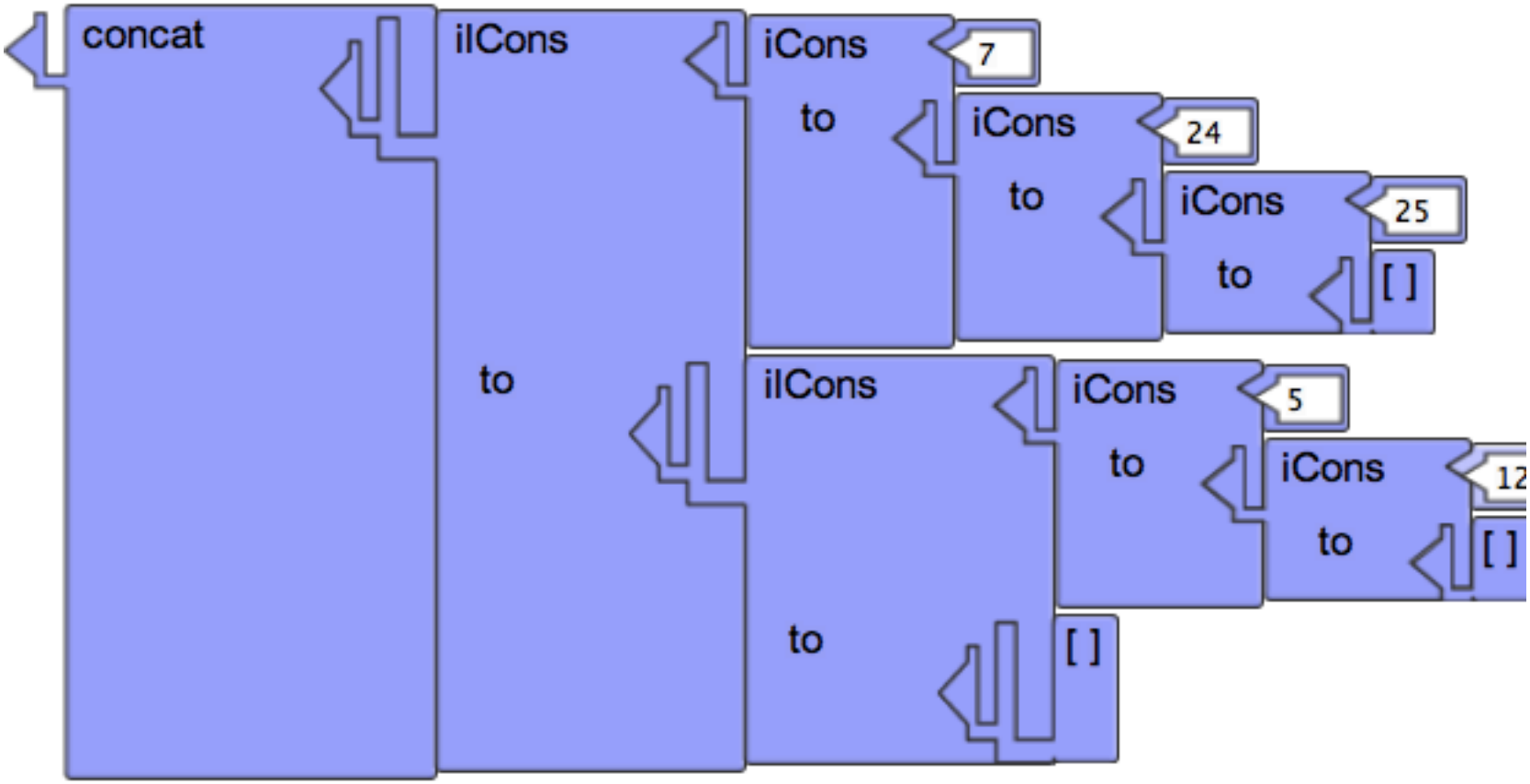
- Recursive drawing method
- Draw:
  - Bottom of arrow
  - Range argument
  - Middle of the arrow
  - Domain argument
  - Top of the arrow

- Smallest type has size unit
- 2 arguments: take the max

Type Construction in Practice



DEMONSTRATION



POLYMORPHISM

- On events connection and disconnection

On Connection:

- Unifies types of socket and plug
- If type of plug / socket changes:
  - Propogate change to all uses of that polymorphic type

On Disconnection:

- “Reset” type
- Propogate type changes to the parent / children

Implementation Details

- ScriptBlocks
- in JavaScript using Google Closure Library
- Represent recursive types by strings and objects

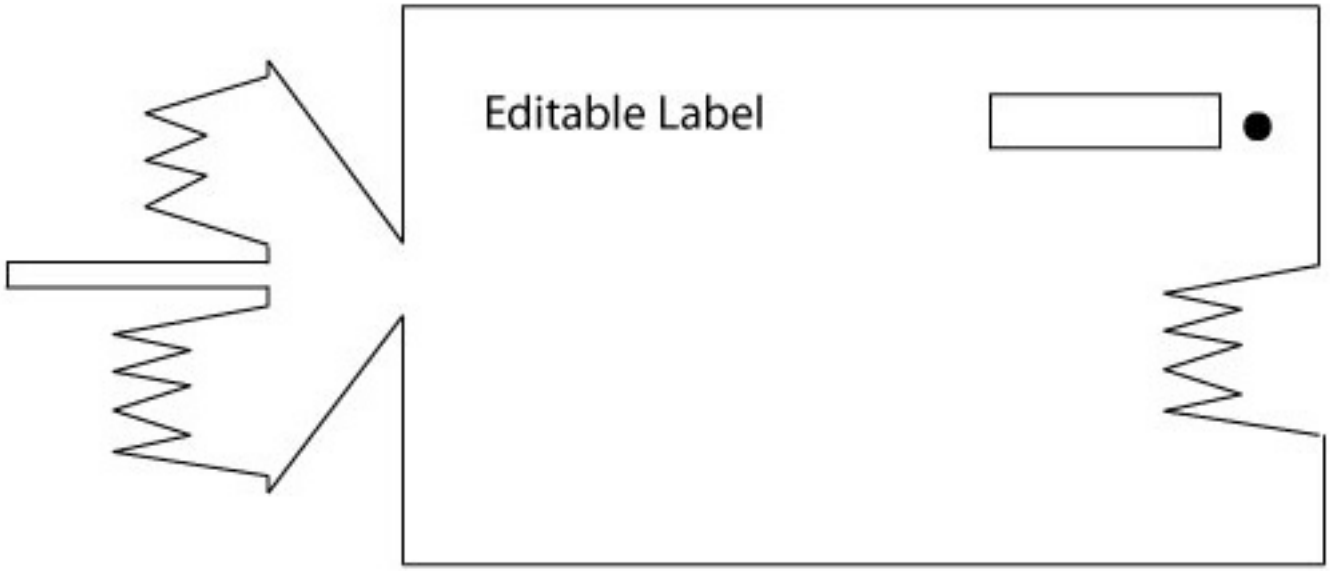
```
{“funD”:  {“tupX”: “boolean”, “tupY”: “string”},
 “funR”: {“listOf”: “number”}}
```

- Represent poly types by objects
  - Ie {“poly” : “a”} or {“poly”: “b”} where “a” and “b” are like sml’s ‘a and ‘b.

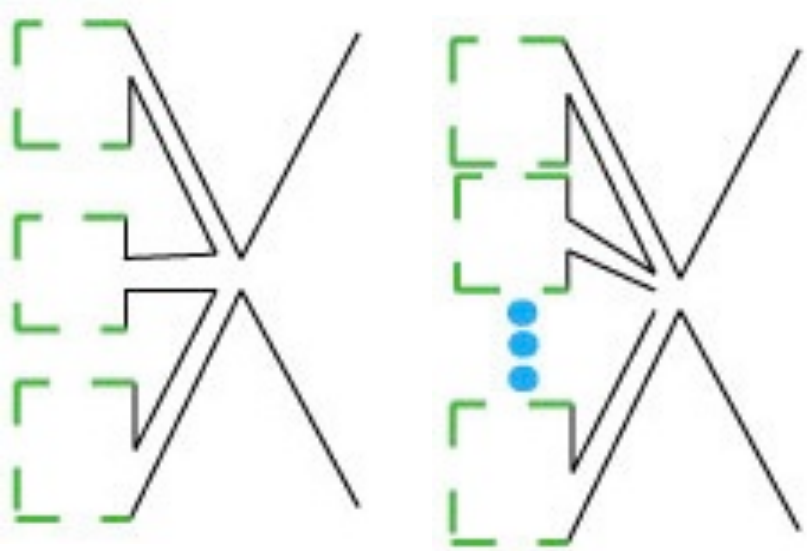
What should functions look like?

StarLogo: TNG

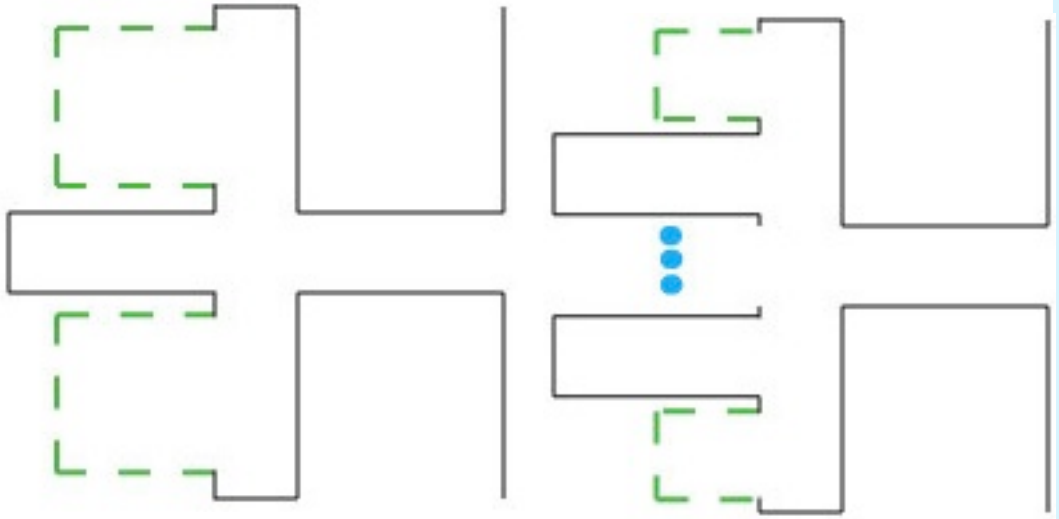
BYOB



What should algebraic datatypes look like?



And



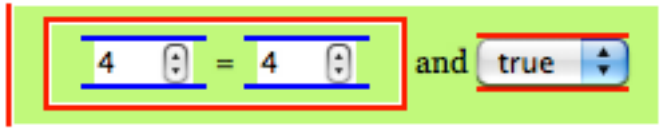
Or

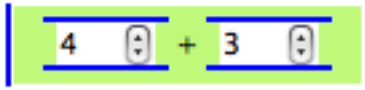
For a later date

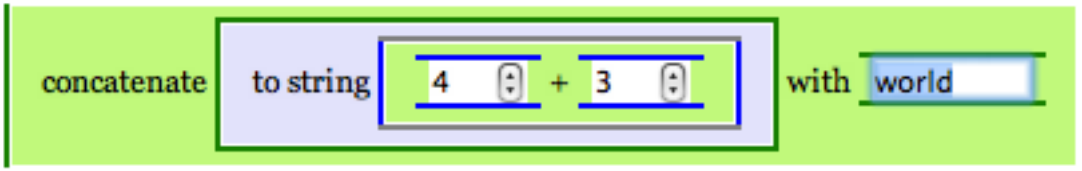
- A sml-like statically typed functional blocks language using these types
  - differentiating visually between ‘a and ‘b.
    - better visualization of polymorphic types
  - algebraic data types
  - pattern matching
- Usability
  - highlighting of all compatible connections
  - user testing
- Other languages and static semantics features
  - Object typing
  - Exception propagation
  - effects
- Other representations of type
  - Waterbear – types as color
  - any others???

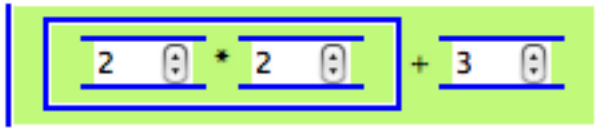
WATERBEAR

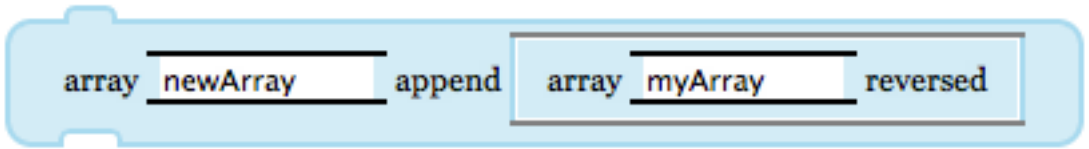
- Inspired by Scratch
- Represents type through color
- 4 basic types: boolean, number, string, array + “all” type
- Explicit casting to convert types



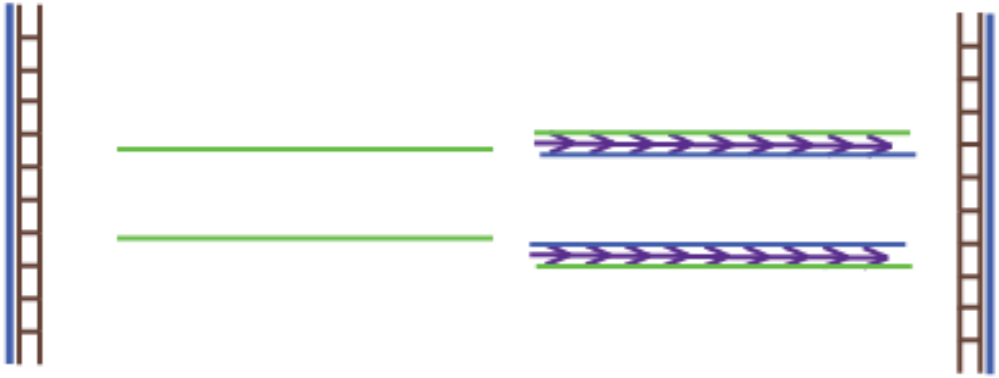


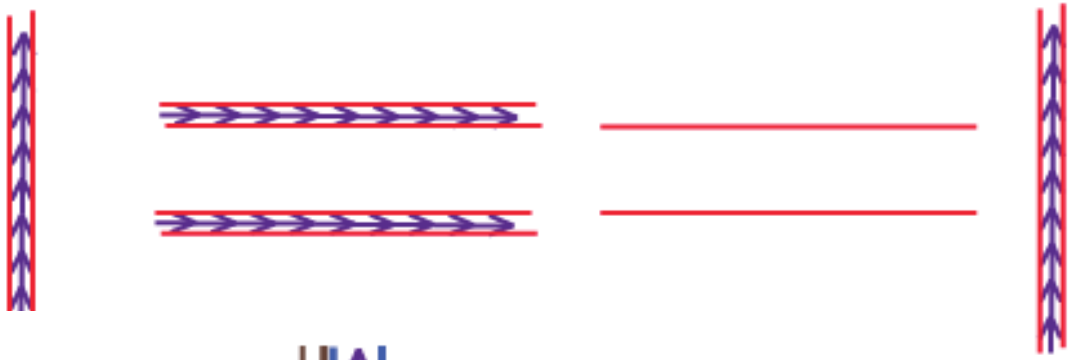


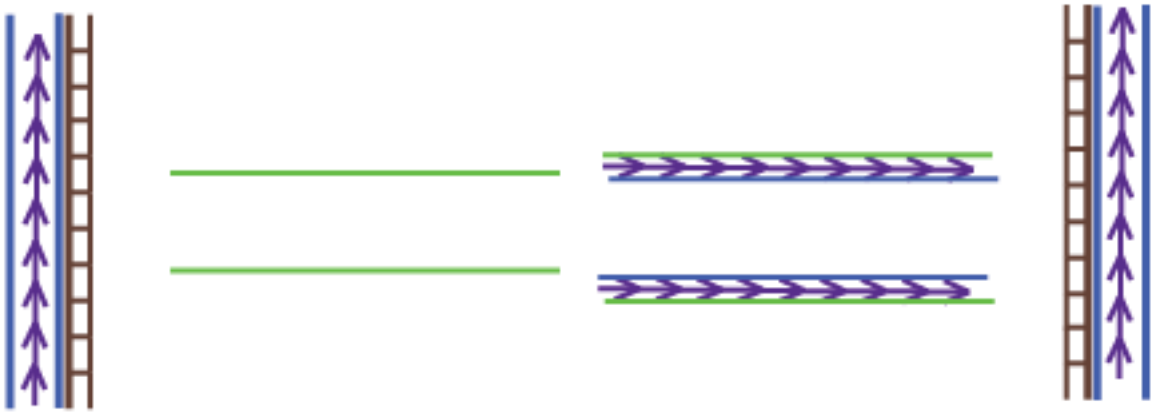




IDEAS FOR COMPOSABLE TYPES - COLOR








```
polyCons any questions?
to [ ]
```