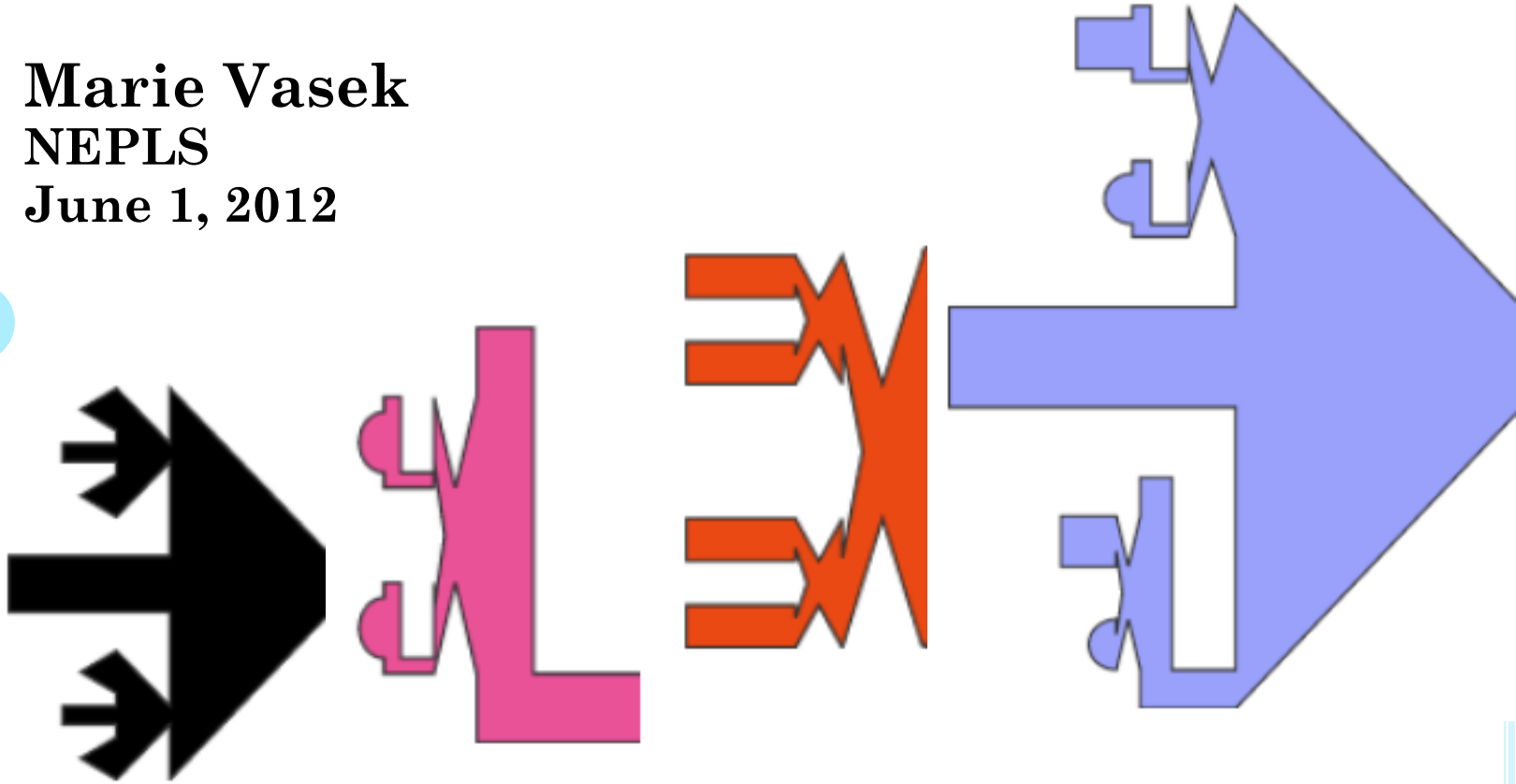
Representing Expressive Types In Blocks Programming Languages

Marie Vasek
NEPLS
June 1, 2012



Expanding textual type systems to block systems

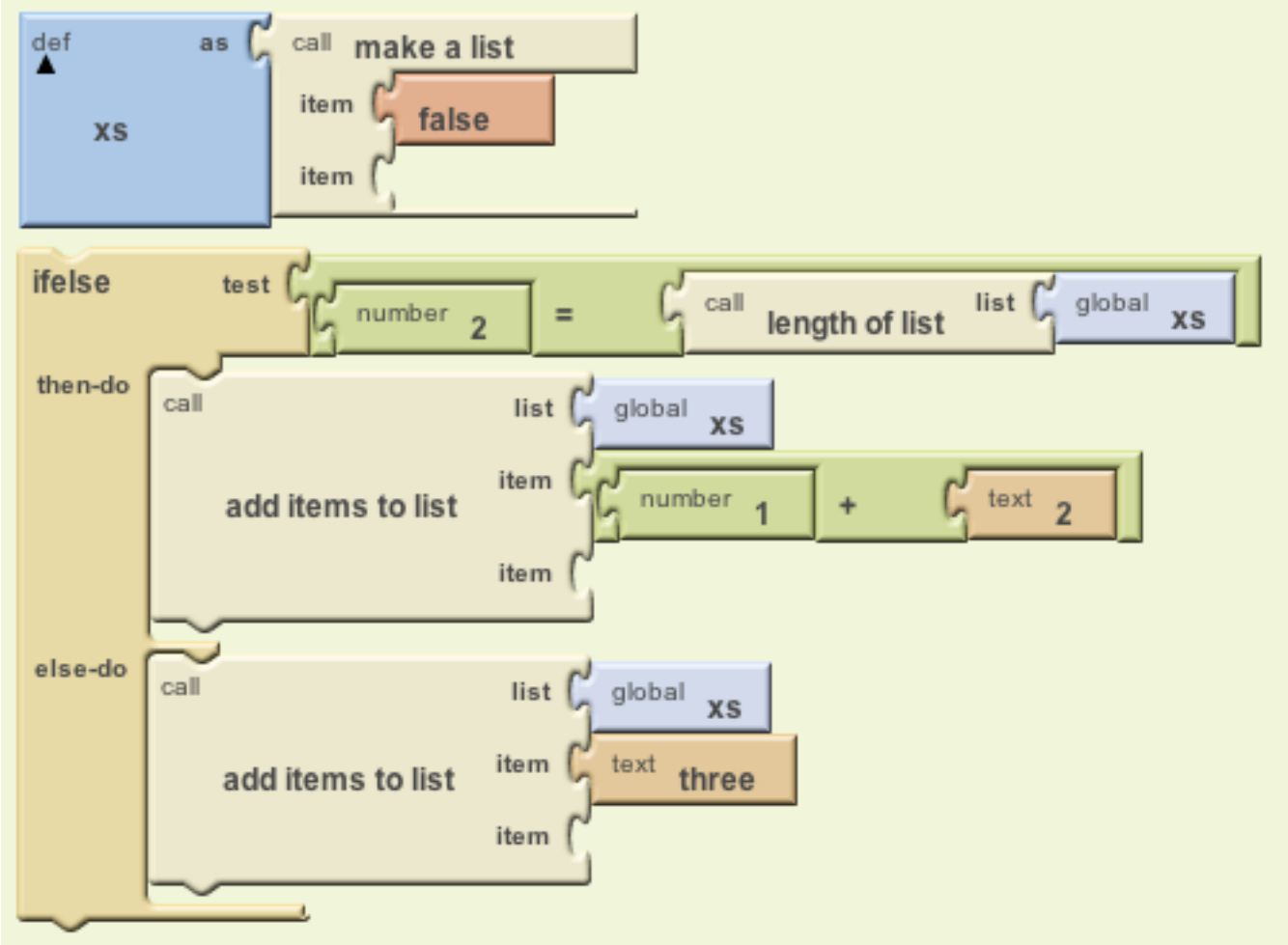
Problem: Current block languages aim to lower barriers to programming but only make weak attempts at implementing a type system.

Solution: Create blocks language where the shape of the block lends to the use.

Overview:

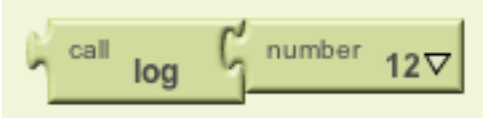
- Type systems in other blocks languages
- TypeBlocks:
  - Shape types
  - Polymorphism
  - … but no language in which they are embedded.

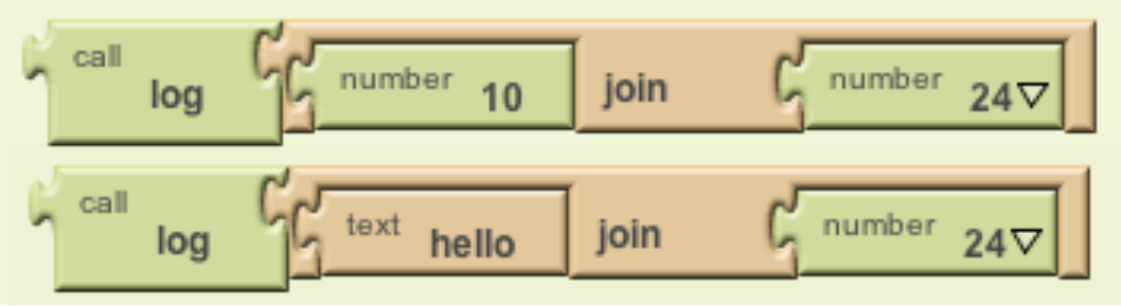
App Inventor – Dynamic-ish Typing



- All types have the same plug shape
- Basic type checking but not really

Timing of Errors

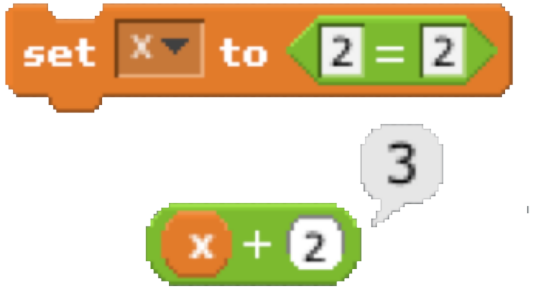


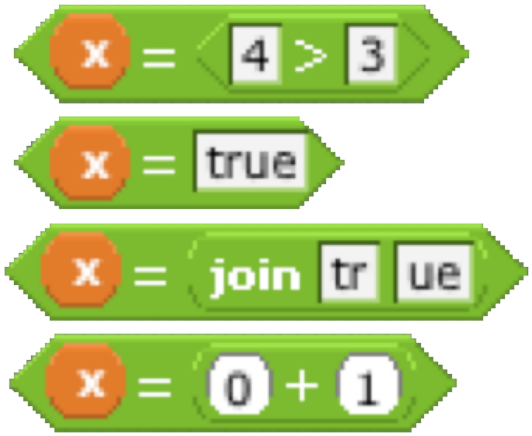


Scratch – Wonky Typing

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

Type Conversion

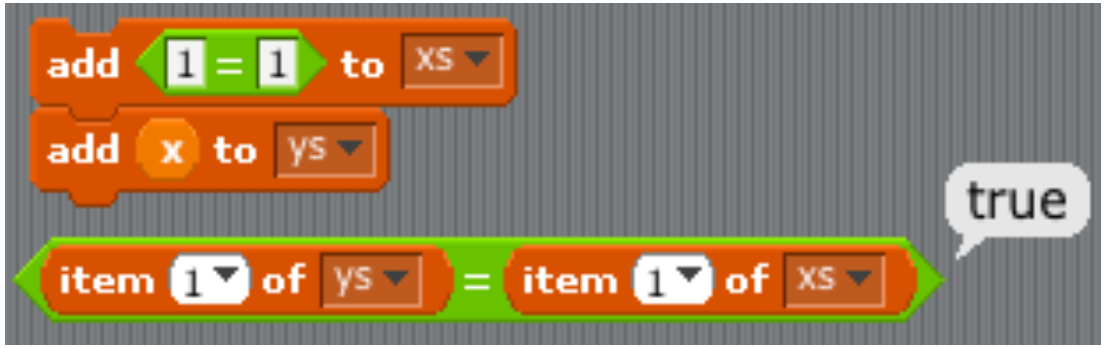




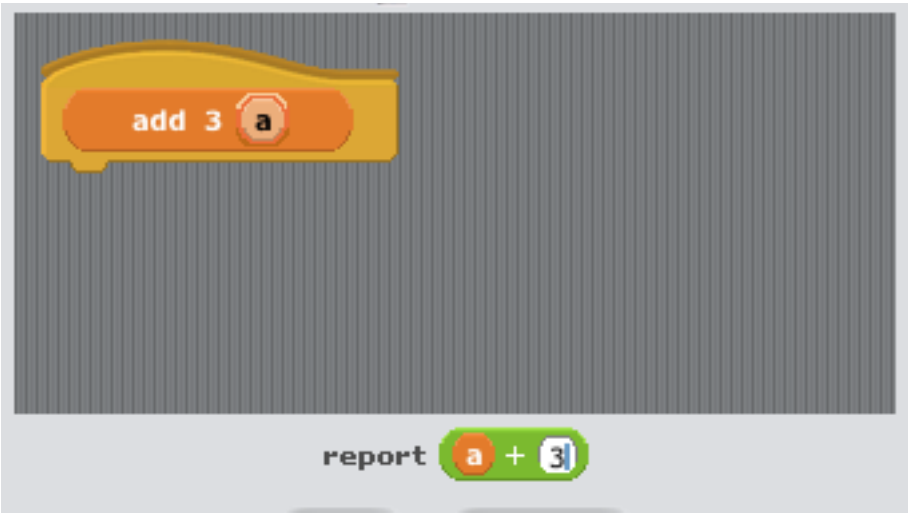
Evaluate to true

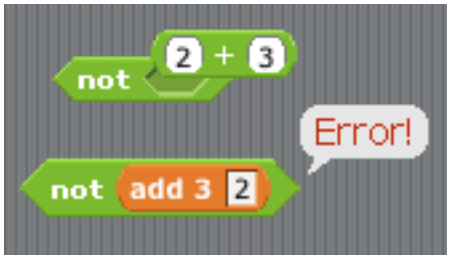
Evaluates to false

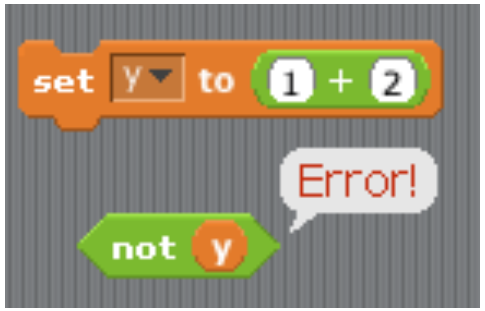
Type Conversion - Lists



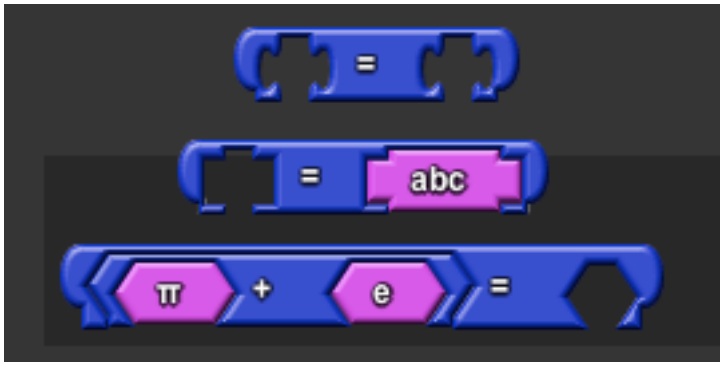
BYOB – More Wonkiness

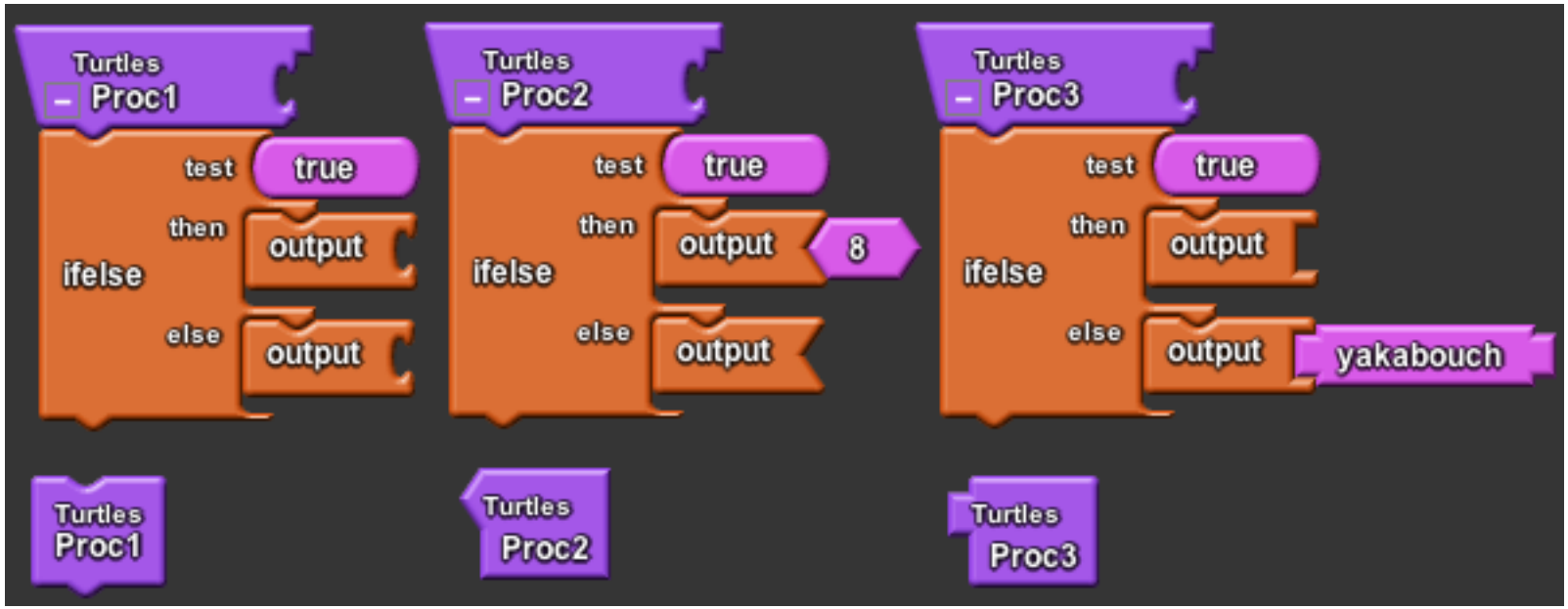




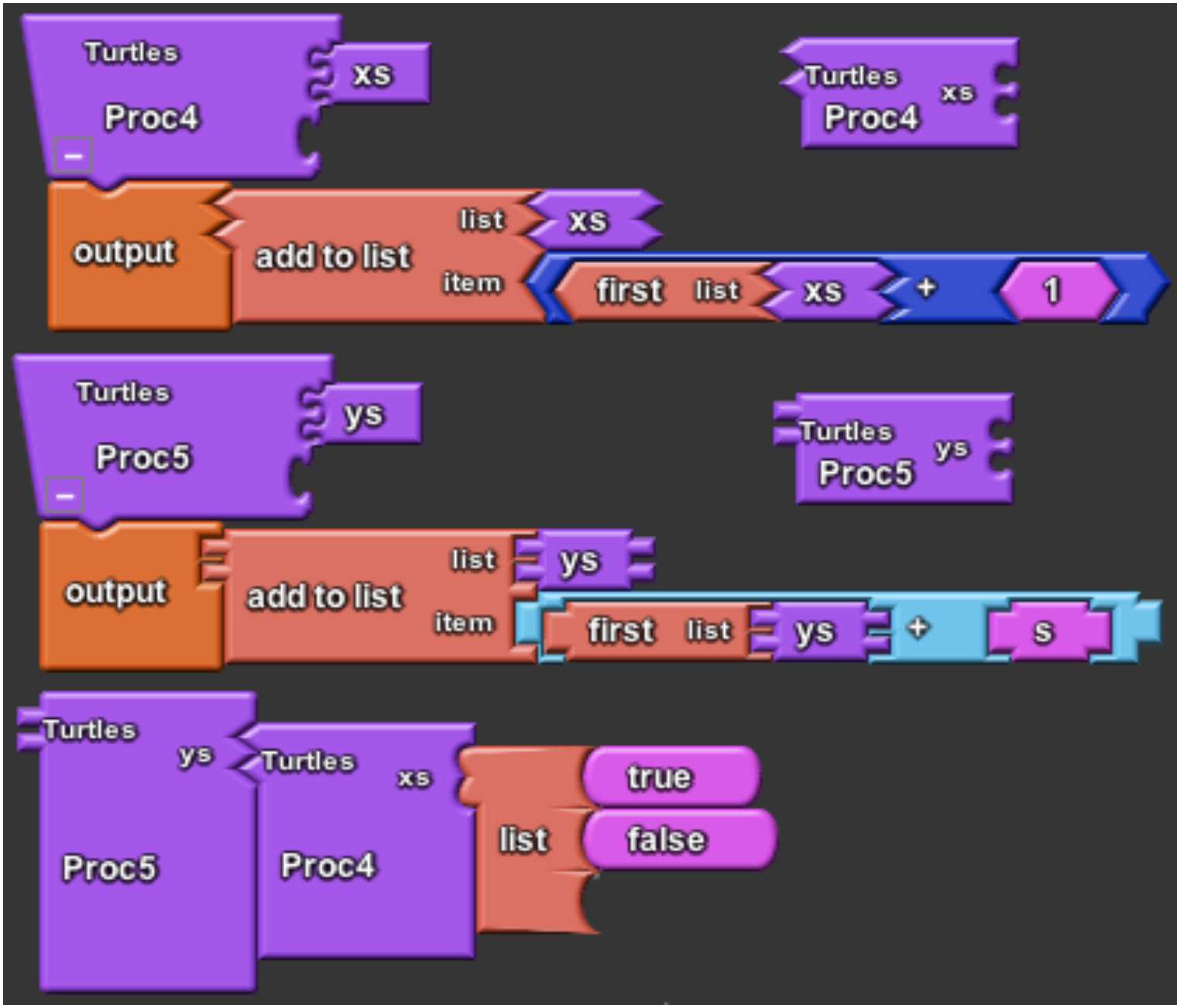


StarLogo: TNG - Polymorphism





Procedures


Turtles
Proc4
xs
output
add to list
list
xs
item
first
list
xs
+
1
Turtles
Proc4
xs
Turtles
Proc5
ys
output
add to list
list
ys
item
first
list
ys
+
s
Turtles
Proc5
ys
Turtles
Proc4
xs
list
true
false

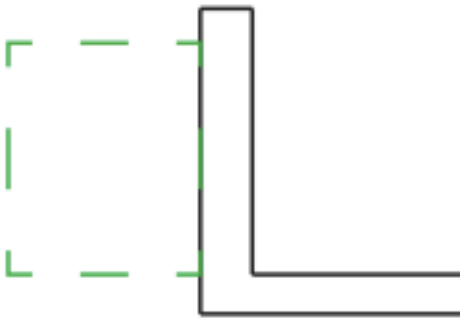
What I Did

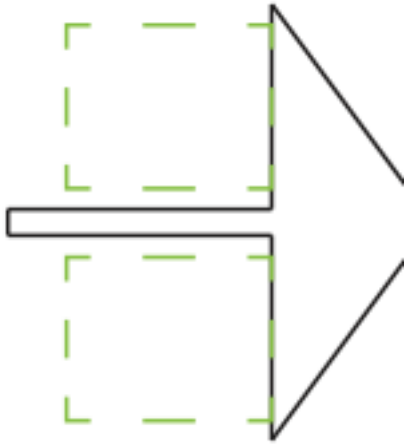
- Blocks types inspired by SML
- Base types + type constructors => ability to represent countably many types
- Each arbitrarily complex type = unique connector shape
- ML- style universal polymorphism

3 base types: `number, boolean, string`

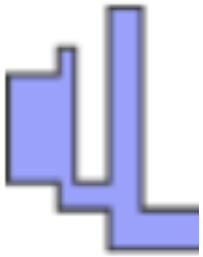
BUILD-A-TYPE

3 constructors:





`listof` `pair` `function`



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

More Example Plugs

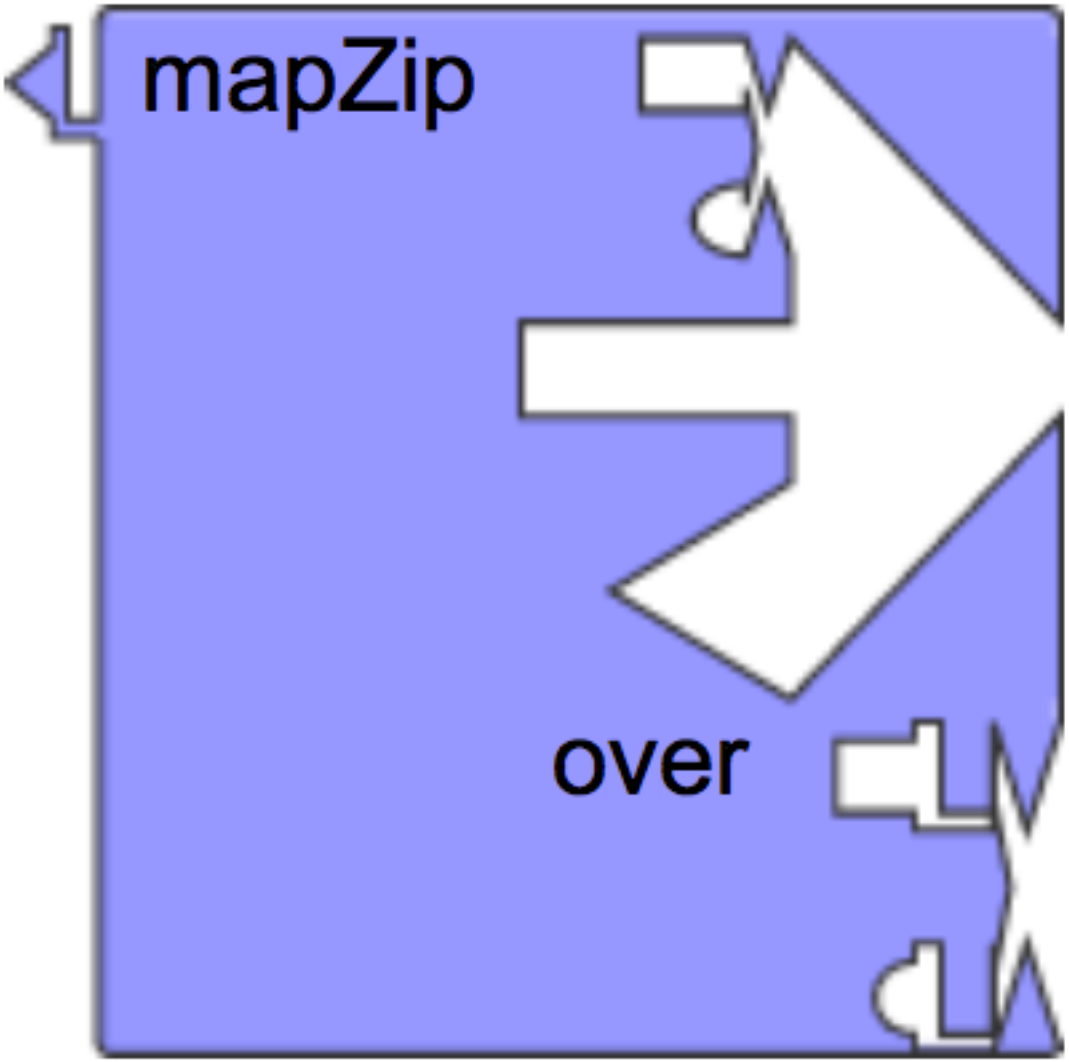
listof (string * boolean)

(listof string) * boolean

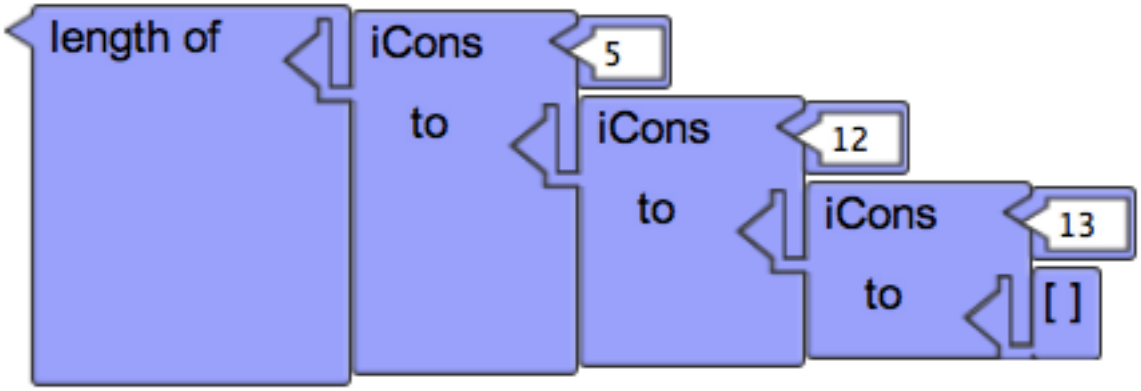
boolean * (string -> listof number)

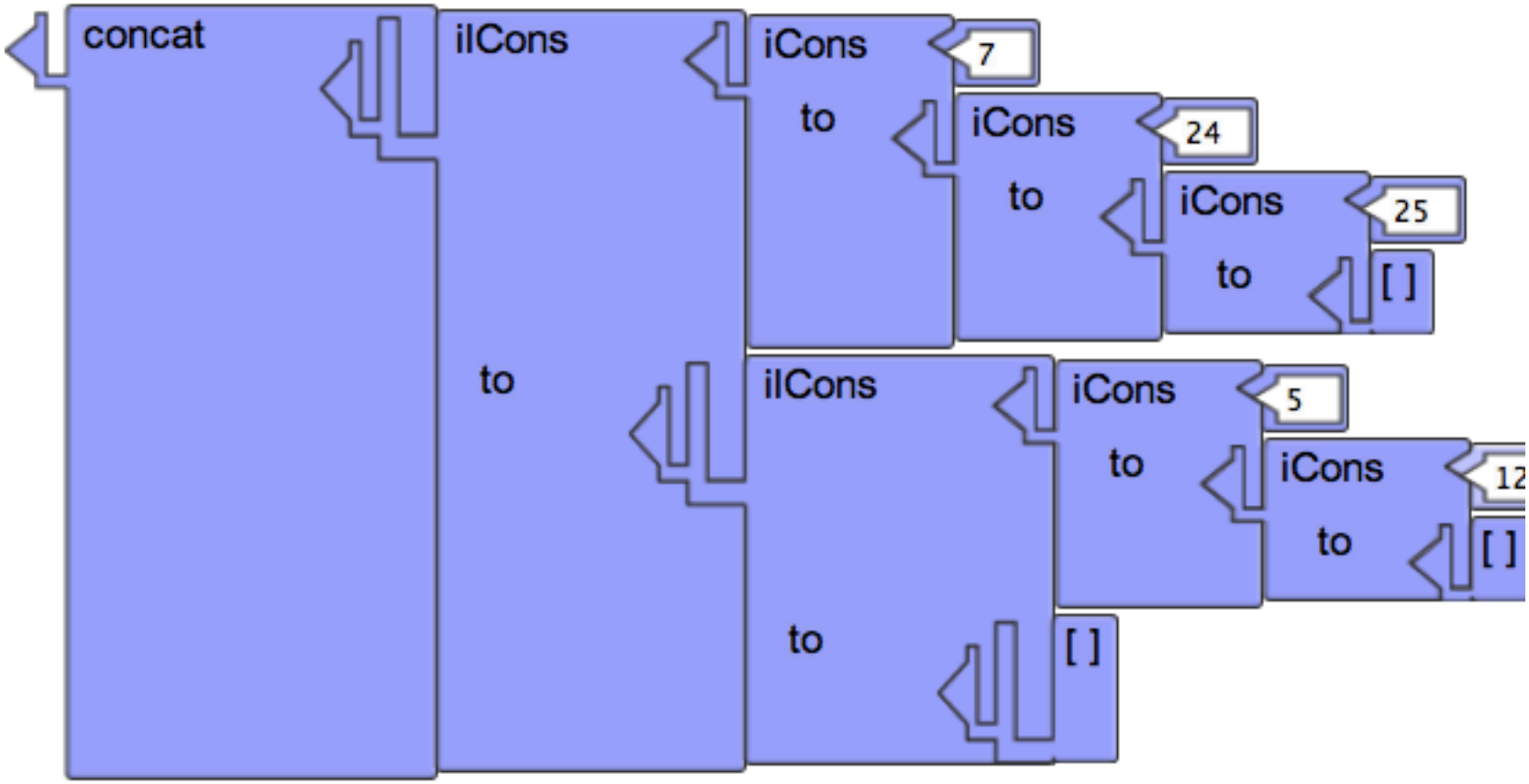
(boolean * string) -> (listof number)

Zip and Map

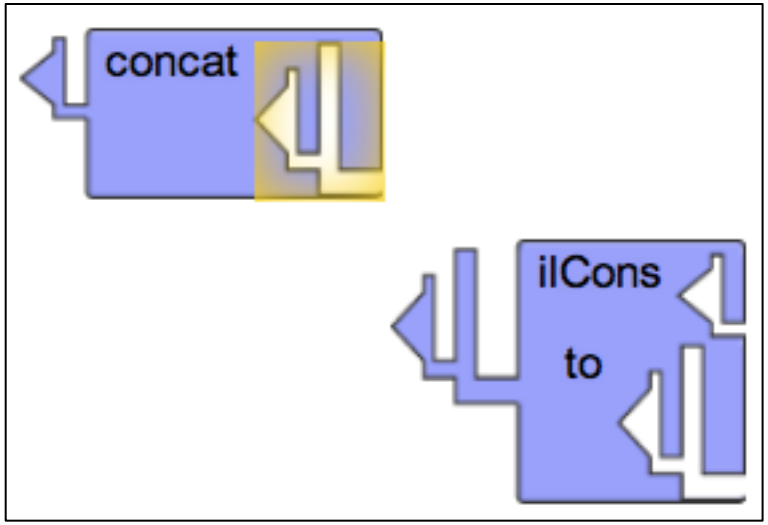


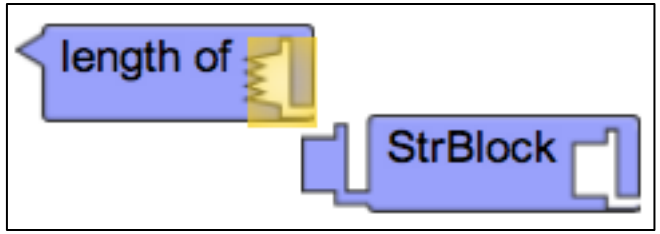
Type Construction in Practice

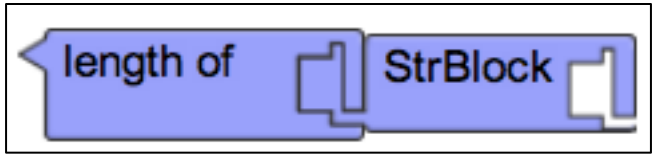


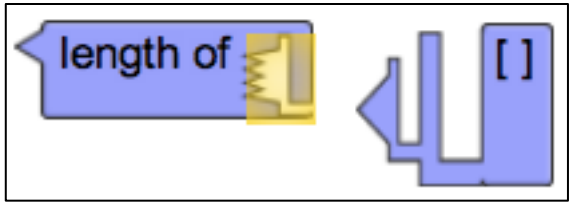


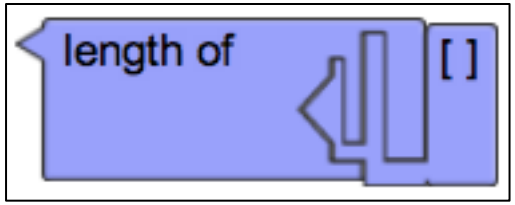
SEE IT GO!


concat
ilCons
to

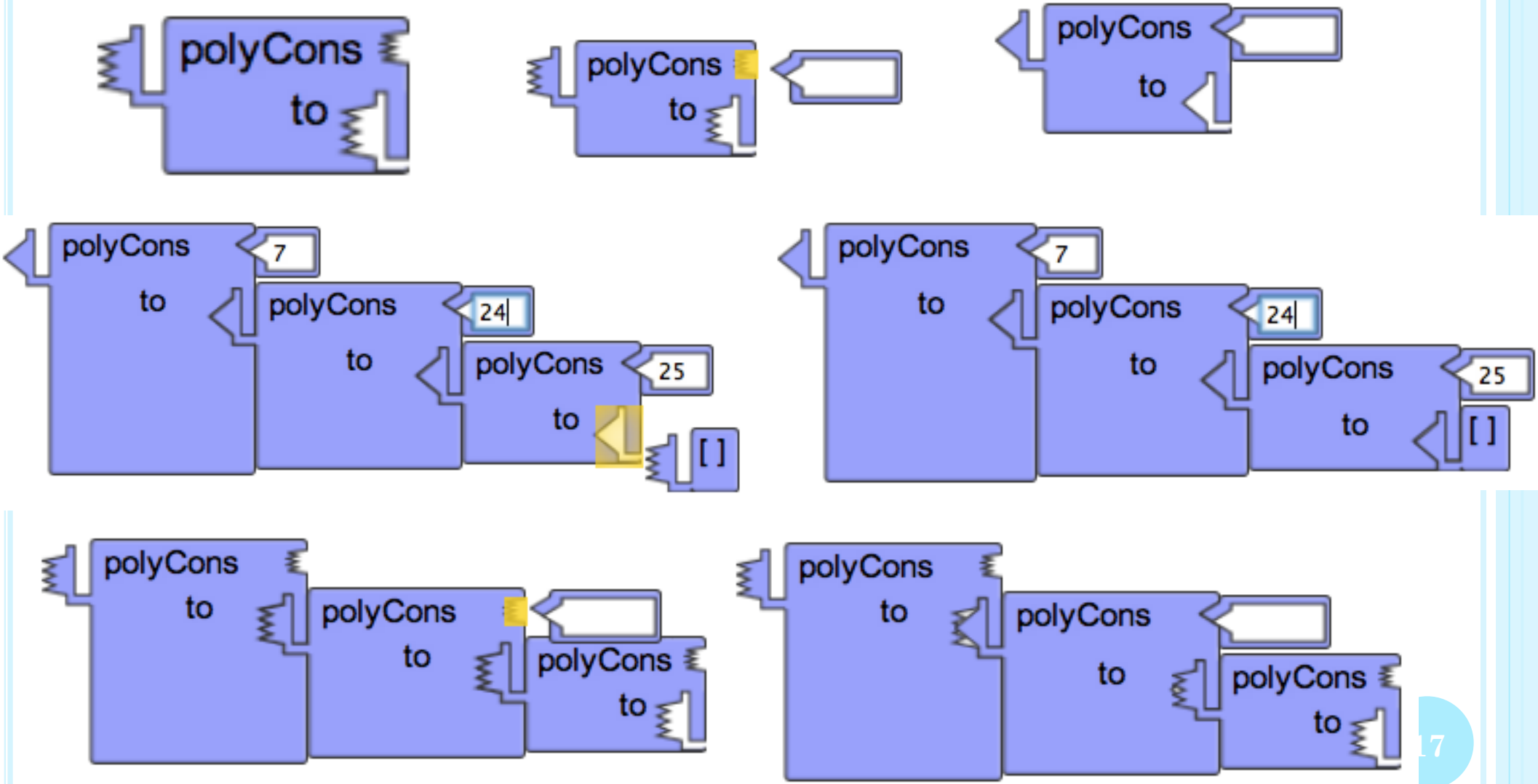

length of
StrBlock


length of
StrBlock


length of
[ ]


length of
[ ]

ML-Style Universal Polymorphism



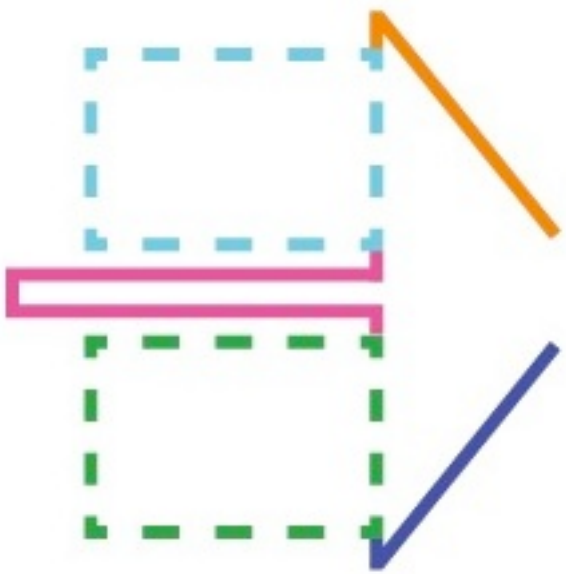
Implementation Details

- 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`.

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

Polymorphism

- On events plug and unplug

On Plug:

- Unifies types of blocks
- If type of plug / socket changes:
  - Change the other plug/sockets on current block to reflect change
  - Do the same to the parent / children of the block

On Unplug:

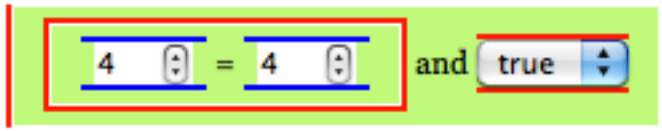
- "Reset" type
- Propogate type changes to the parent / children

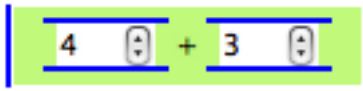
For a later date

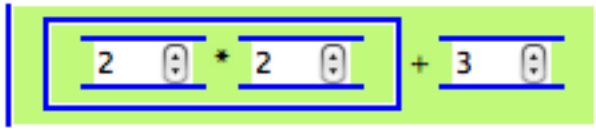
- A sml-like statically typed functional blocks language using these types
  - differentiating visually between ‘a and ‘b.
    - better visualization of polymorphic types
  - algebraic data types
  - pattern matching
- Block Java
  - objects
  - “ad hoc” polymorphism
- Usability
  - highlighting of all compatible connections
  - user testing
- Other representations of type
  - Waterbear – types as color
  - any others???

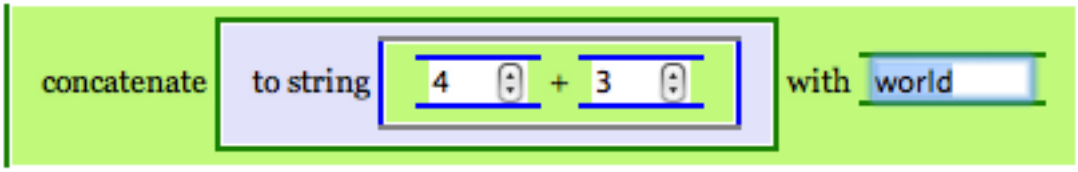
WATERBEAR

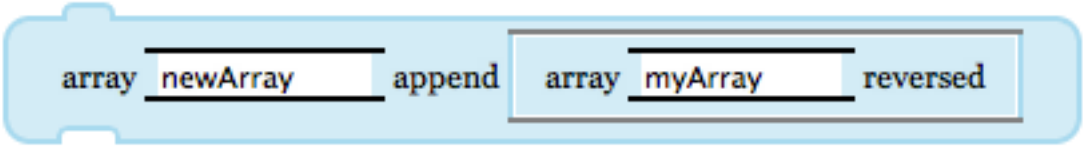
- Inspired by Scratch
- Represents type through color
- 4 basic types: boolean, number, string, array + “all” type
- Explicit casting to convert types



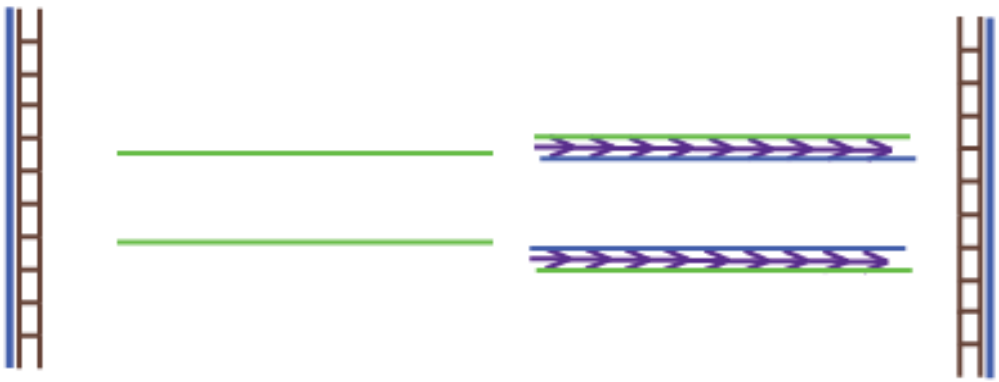


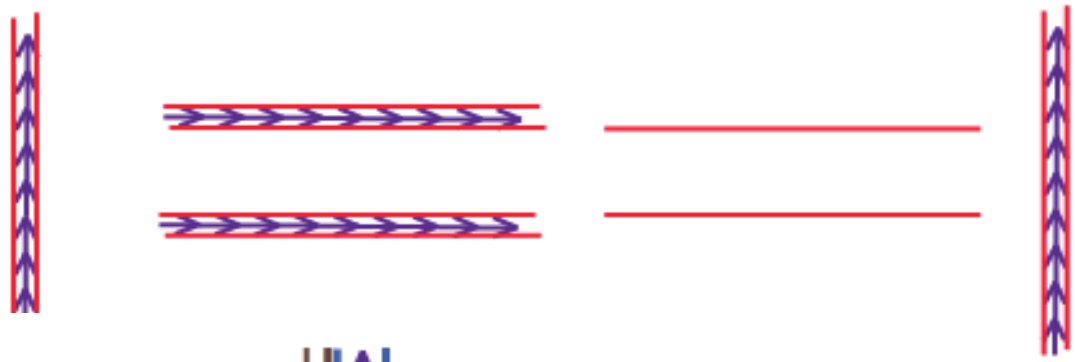


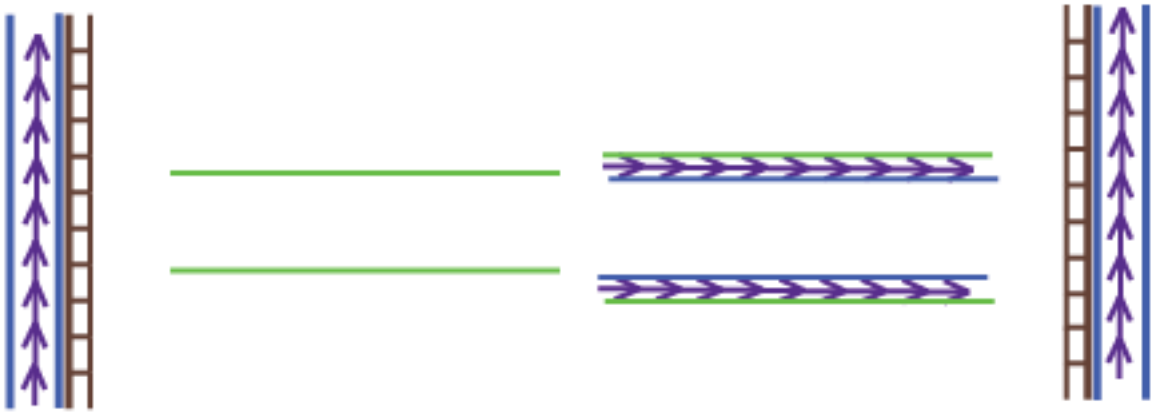




IDEAS FOR COMPOSABLE TYPES - COLOR








any questions?


polyCons
to
any questions?
[ ]


any questions?


any questions?