Keyboard? How quaint.
Visual Dataflow Implemented in Lisp.

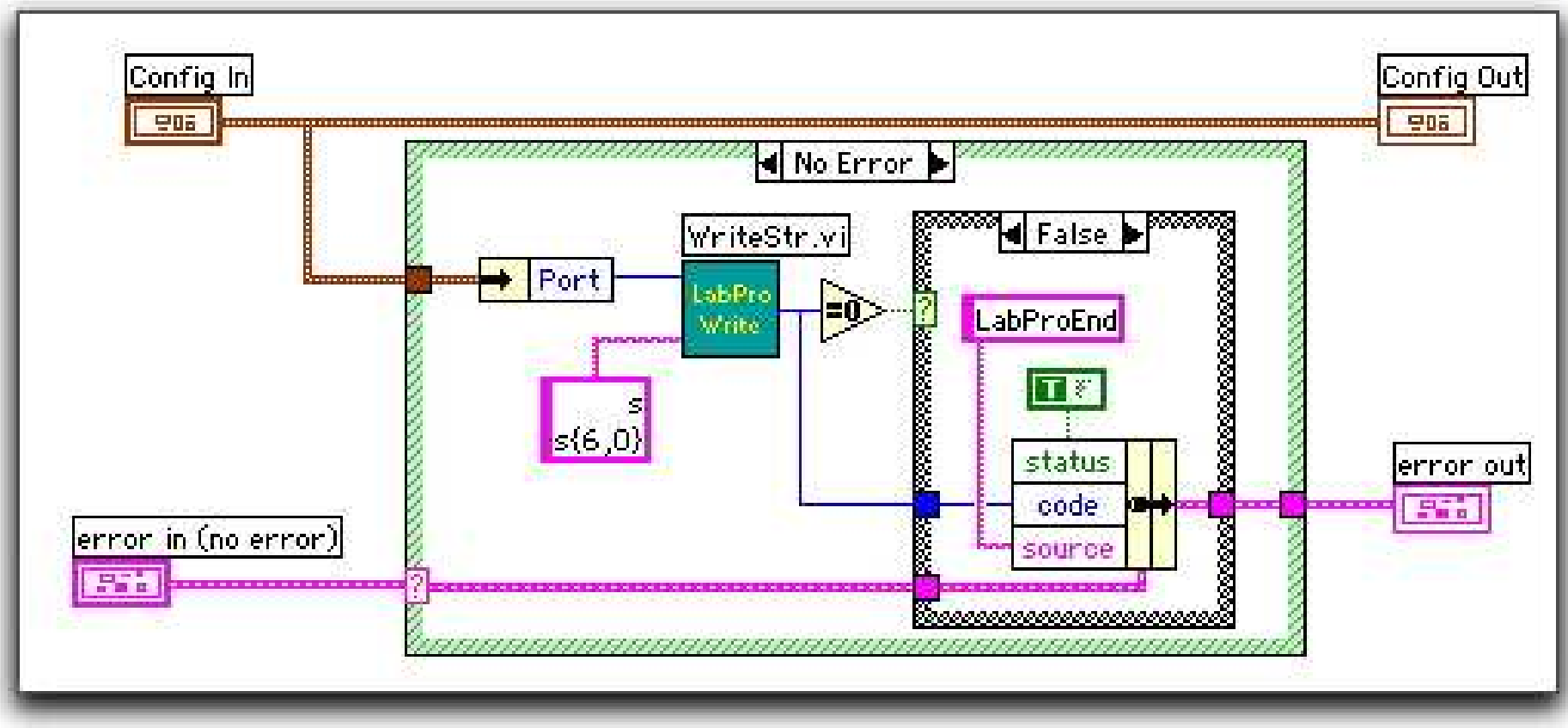
Donald Fisk

hibou@onetel.com

July 21, 2015

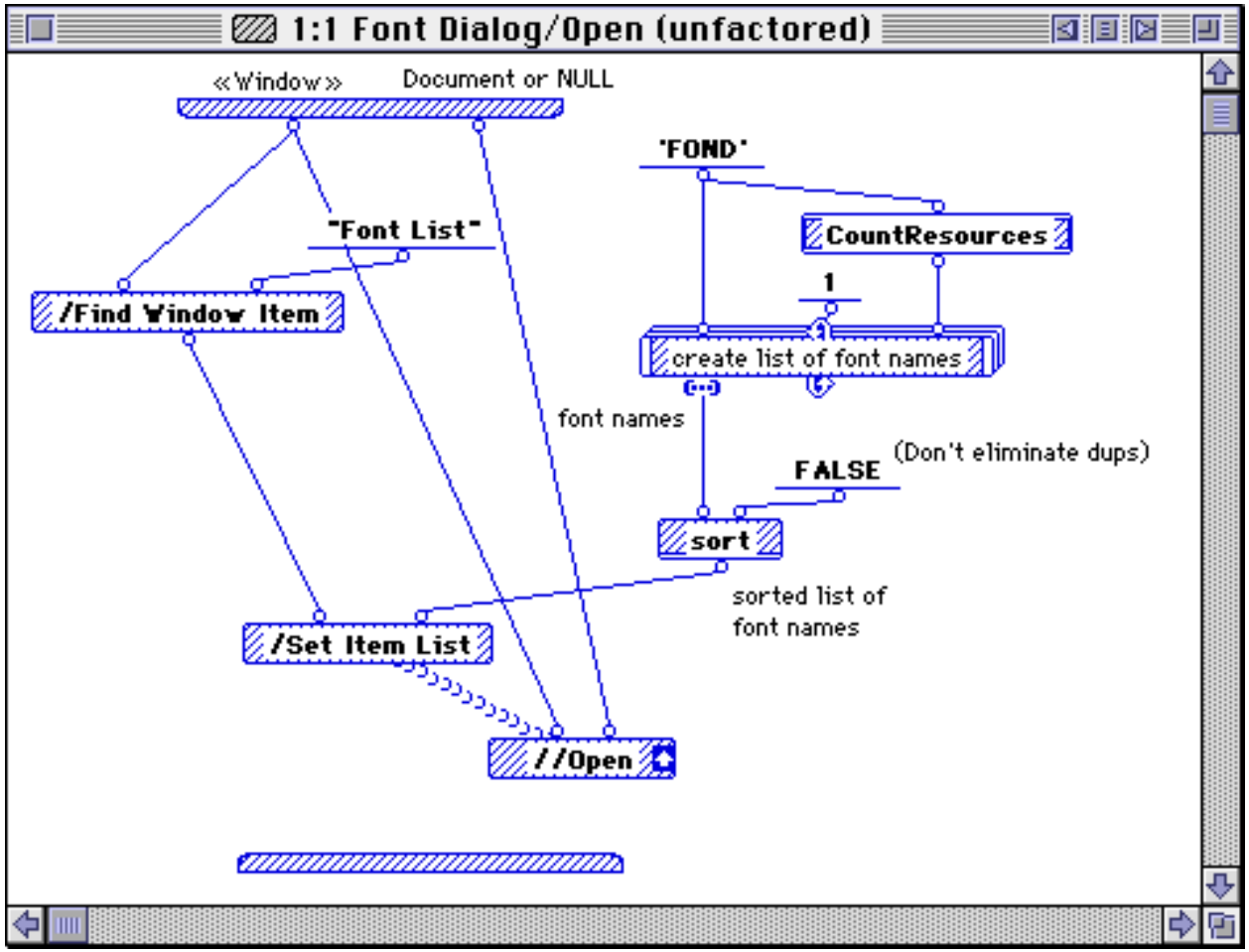
Table of Contents

LabVEIW



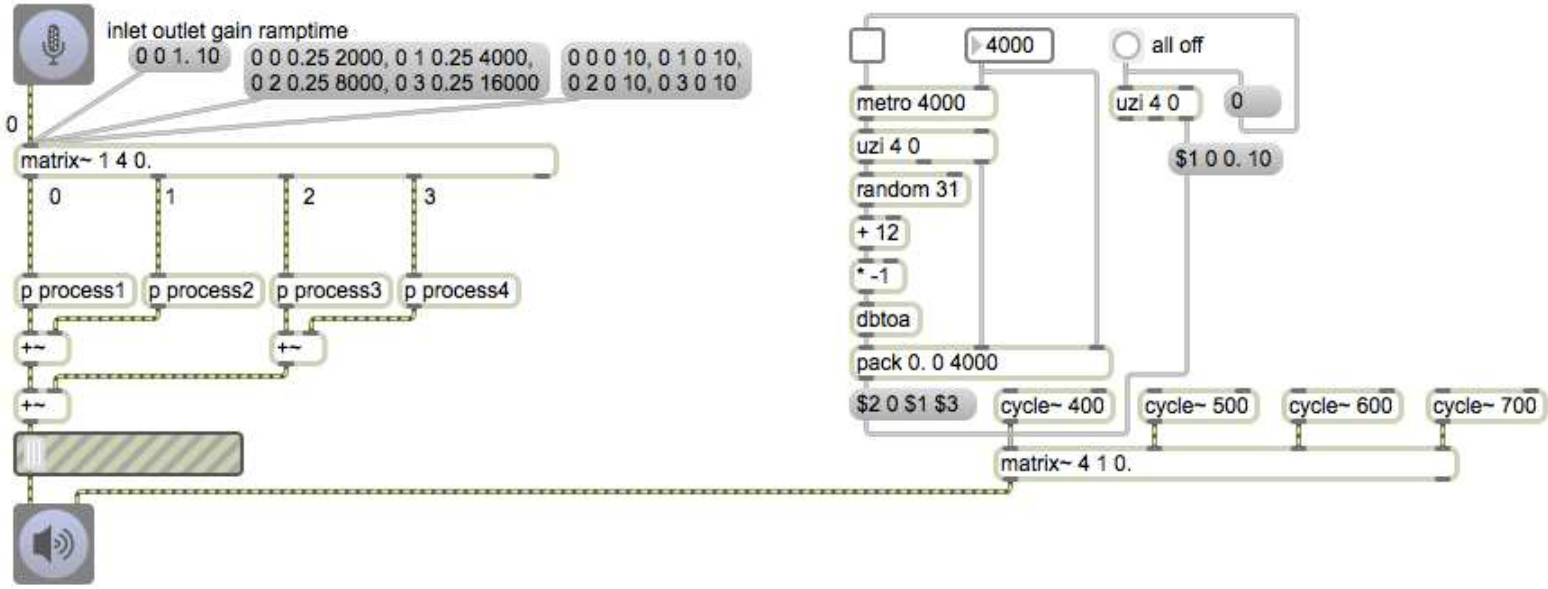
(from
http://www.physics.uoregon.edu/~torrence/classes/02S_390/week4.html)

Prograph



(from
`http://www.mactech.com/articles/mactech/Vol.10/10.11/PrographCPXTutorial/index.html`)

max

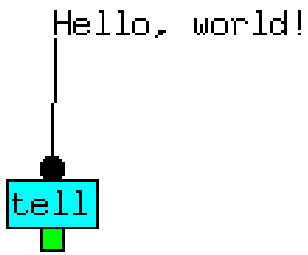


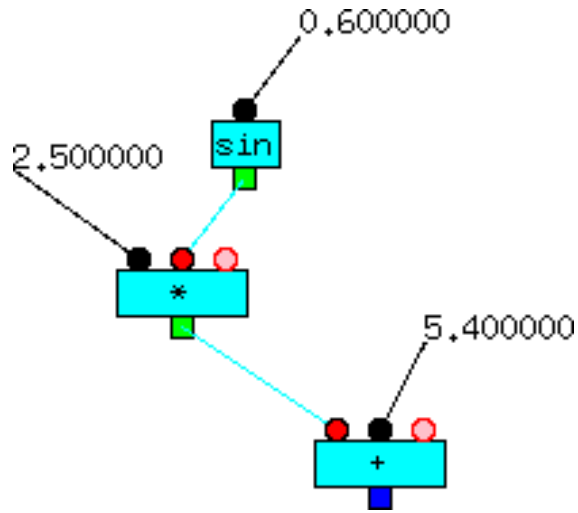
(from `http://sites.uci.edu/computermusic/category/msp-tutorials/page/8/`)
uzi is designed for rapid-fire output of a specified number of bang messages.

Emblem

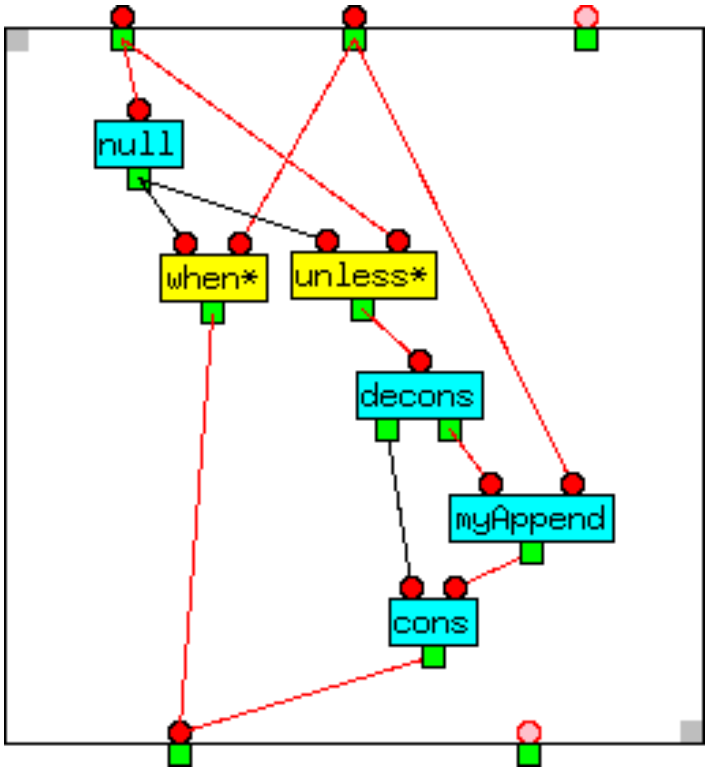
- Designed as the implementation language for Full Metal Jacket.
- Simpler, but not gratuitously different from Common Lisp.
- Object oriented, single inheritance.
- Compiles to byte code.
- Has HTTP server; can browse Lisp objects from a web browser.
- X11 library code.
- OpenGL library code.
- Other library code (some AI, data mining).

Simple Examples



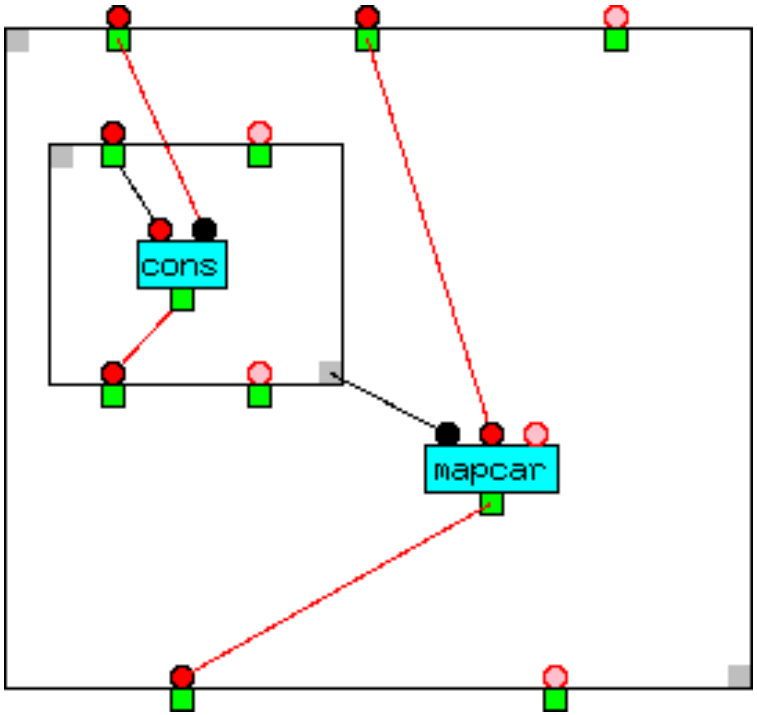


Recursion




```
(defun myAppend (x y)
  (if (null x)
      y
    (cons (car x)
          (myAppend (cdr x)
                    y))))
```

Functional Arguments




```
(defun foo (x y)
  (mapcar (lambda (z)
            (cons z x))
          y))
```

Dataflow

- Computations can proceed in parallel if all their inputs are available. Execution doesn't have to depend on a program counter.
- This suggests programs should be directed graphs.
- Vertices should transform data, edges should transmit data.
- We gain:
  - Parallel programming.
  - Underpinned by graph theory.
  - Homoiconicity.
- We lose:
  - Flow of control.
  - Variables.
  - The need to parse programs before executing them.

Adding Edges

- Outputs can only be connected to inputs.
- Makes sense to restrict on type (e.g. we shouldn't be allowed to add one to a string).
- Requires composite types to work (or we won't be able to add one to the first of a list of integers).
- Suggests type inference (Hindley-Milner).
- We gain:
  - Strong types, checked by a smart editor.
- We lose:
  - The need to declare types.
  - Runtime type errors.
  - Compile time type errors.

Conditional Dataflow

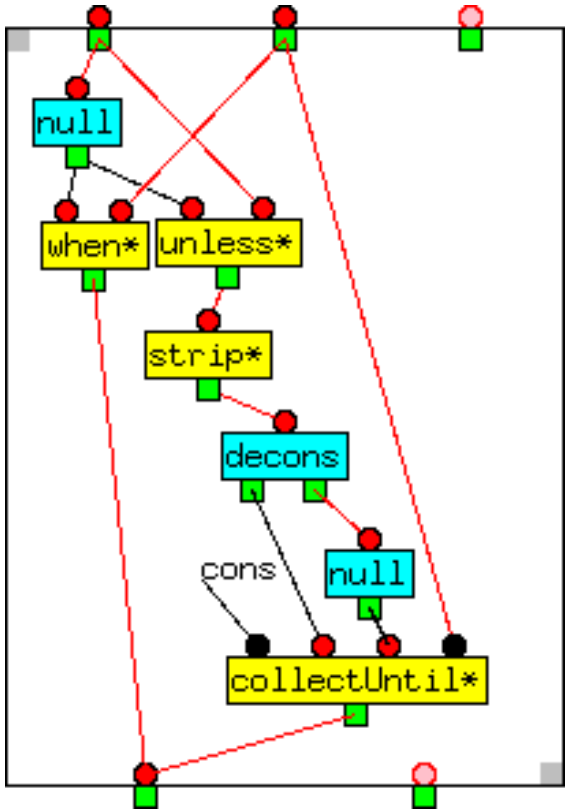
- If `when` receives the value `T` on its first input, it outputs the value received on its second input.
- If it receives `NIL`, it doesn't output anything, so any computation depending on the value output does not proceed.
- `unless` outputs only if it receives `NIL`.
- This suggests vertices don't need to output every time they receive inputs (Filters).
- We gain:
  - a generalization of function;
  - conditional execution without a special construct.

Iteration Without Loops

We can generalize further.

- Vertices should be able to output *more than once* after receiving inputs (Iterators). This is useful when iterating through, e.g.,
  - integers;
  - lists;
  - data received over a connexion.
- Vertices should be able to output once after receiving inputs one or more times (Collectors). This allows the accumulation of results, e.g. for
  - summation;
  - storing in lists or arrays.
- We gain:
  - iteration without loops or special constructs; still pure dataflow.
- We lose:
  - loops.

Iteration



Output of strip*	2nd Input	3rd Input of collectUntil*	Accumulator	Output
(a b c)	a	NIL	NIL (a)	
(b c)	b	NIL	(b a)	
(c)	c	T	(c b a)	(c b a)

Tags

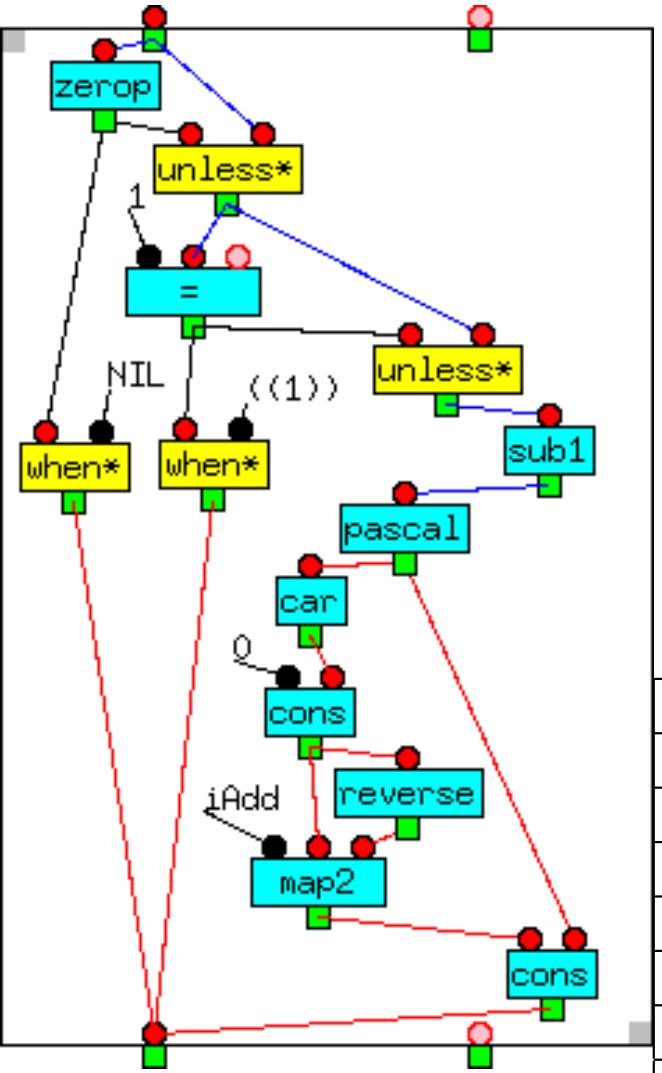
- Values must be tagged.
- Values with the same tag are processed together.
- A function can be called when it has values with the same tag for all its arguments.
- When an enclosure is entered (e.g. when a function is called), its arguments are assigned new tags, until the enclosure is left.

Interpreter Code

- `runNextTask ()`
- `executeVertex (vertex tag argList)`
- `sendValueToInput (inputOfDest tag value)`
- `everyInputHasAValueP (inputs tag)`
- `extractValuesFromInputs (inputs tag)`
- `importArgsIntoEnclosure (enclosure tag args)`
- `applyEnclosure (enclosure args)`

```
(defun myAppend (x y)
  (applyEnclosure (get myAppend enclosure)
                  (list x y)))
```

Type Inference



Input Types	Function	Output Types
Int	pascal	?a
(List ?b)	car	?b
?c (List ?c)	$cons_a$	(List ?c)
(List ?d)	reverse	(List ?d)
(?e ?f) $\rightarrow$ (?g) (List ?e) (List ?f)	map2	(List ?g)
?h (List ?h)	$cons_b$	(List ?h)

Table: Vertex types in pascal

Function	Output Type	Input Type	Function
pascal	?a	(List ?b)	car
car	?b	(List ?c)	$cons_a$
$cons_a$	(List ?c)	(List ?d)	reverse
$cons_a$	(List ?c)	(List ?e)	map2
reverse	(List ?d)	(List ?f)	map2
map2	(List ?g)	?h	$cons_b$
pascal	?a	(List ?h)	$cons_b$

Type Inference

Function	Output Type	Input Type	Function
pascal	?a	(List ?b)	car
car	?b	(List ?c)	cons_a
cons_a	(List ?c)	(List ?d)	reverse
cons_a	(List ?c)	(List ?e)	map2
reverse	(List ?d)	(List ?f)	map2
map2	(List ?g)	?h	cons_b
pascal	?a	(List ?h)	cons_b

Table: Edge types in pascal

Type Variable	Type Type
?a	(List (List Int))
?b	(List Int)
?c	Int
?d	Int
?e	Int
?f	Int
?g	Int
?h	(List Int)

Table: Inferred types in pascal

Value	Type	Input Type	Fn.
0	Int	?c	cons_a
iAdd	(Int Int) $\rightarrow$ (Int)	(?e ?f) $\rightarrow$ (?g)	map2

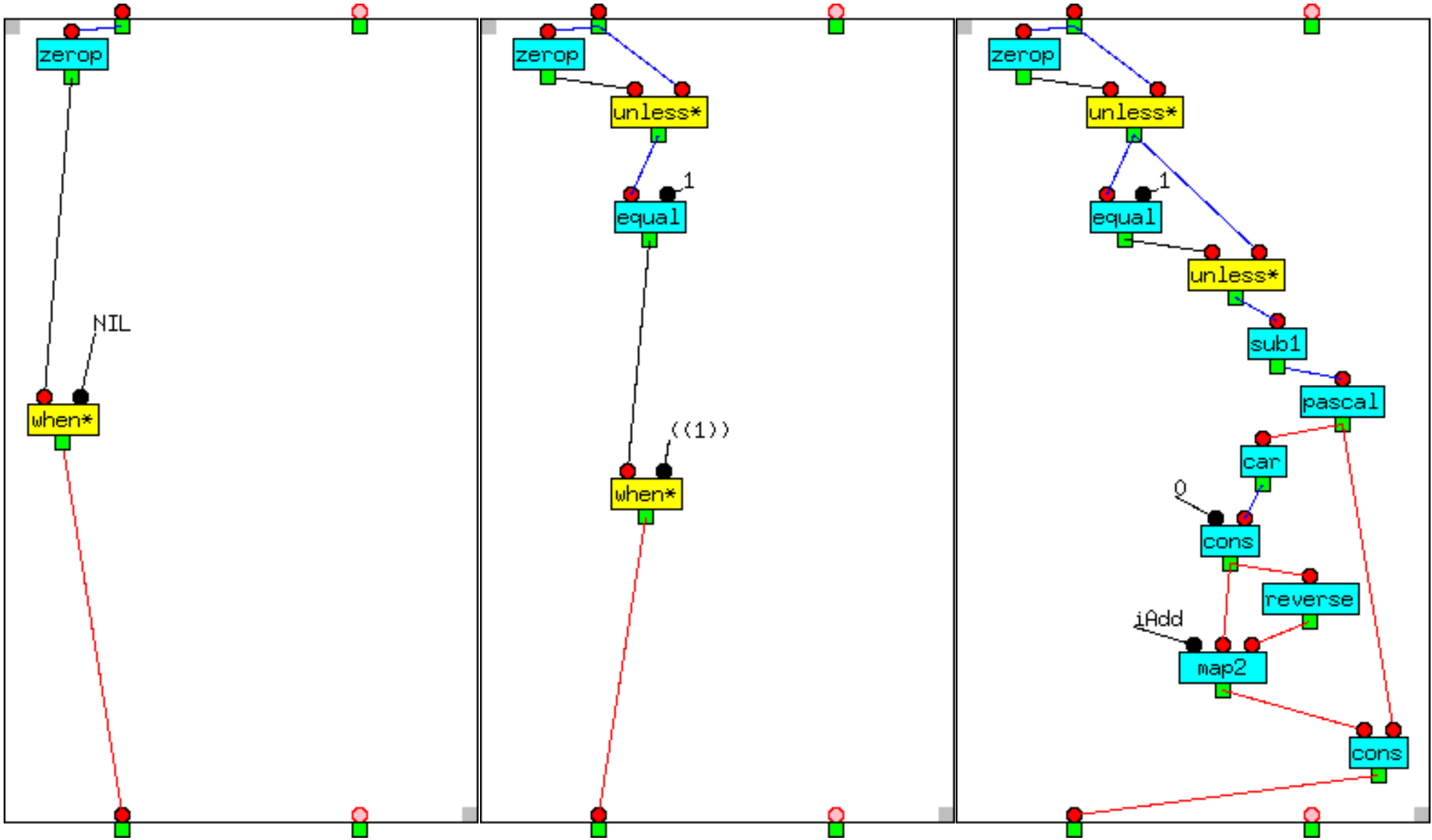
Table: Constant types in pascal

(pascal 5) $\rightarrow$ ((1 4 6 4 1) (1 3 3 1) (1 2 1) (1 1) (1))

Race Conditions

- The result of a function should not depend on the order its vertices are executed.
- If two or more edges converge on the same input, there might be a race condition.
- It is therefore important to ensure that data only travels down one edge whenever the function is called.
- Data flows through one of several mutually exclusive *streams*.
- The stream containing a vertex is found by following edges downstream, then back upstream at every vertex encountered.
- If data flows through any edge in a stream, it flows through all edges in that stream.
- So, if a stream has more than one edge converging on the same input, a race condition exists.

Streams


zerop
NIL
when*
zerop
unless*
1
equal
((1))
when*
zerop
unless*
1
equal
unless*
sub1
pascal
car
0
cons
reverse
iAdd
map2
cons

Type Extension

```
(deftype (List ?x) (or NIL (Pair ?x (List ?x))))
(deftype (AList ?x ?y) (List (Pair ?x ?y)))
(deftype (Bag ?x) (AList ?x Int))
```

These resemble function definitions.

This suggests types could be defined like functions:

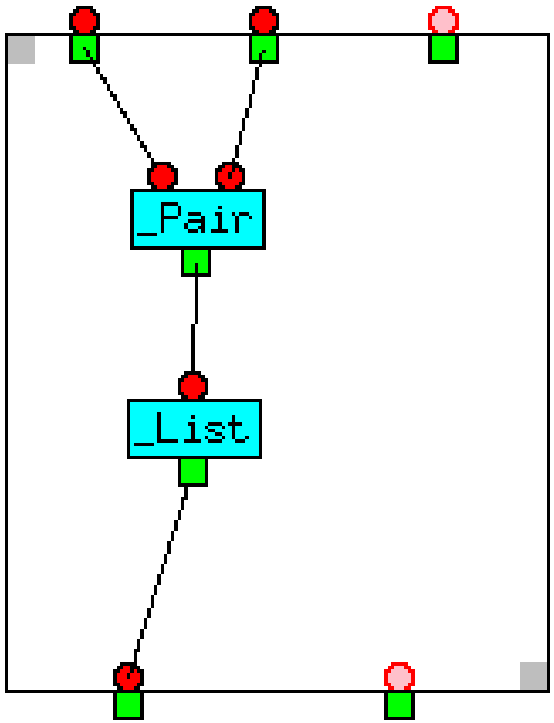


Figure: AList

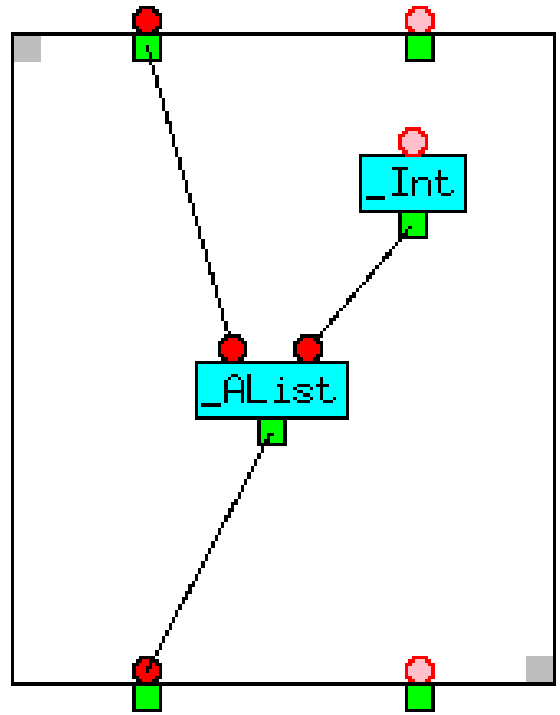


Figure: Bag

What Else Needs to be Done

- Experiment with Filters, Iterators and Collectors to find the best ones.
- Debugger.
- Merge classes and types.
- Macros.

The End