BINARY

comma.types = [

["A", "A", "A"], // zipApplyTo (3 4),(+1 +)

[["X", "A"], "A", ["X", "A"]], // unaryZipApplyTo +1@,(\*2 /2)

[["X", "Y", "A"], "A", ["X", "Y", "A"]], // binaryZipApplyTo :,(+1 -1)

["X", ["X", "Y"], "Y"], // applyTo 3,+1

["X", ["X", "Y", "Z"], ["Y", "Z"]], // applyToBinary 3,+ =1,?

[["X", "Y", "Z"], [["Y", "Z"], "W"], ["X", "W"]], // binaryUnaryApply =,'(1 2 3)

];

dot.types = [

["V", "A", "A"], // applyToArray (1 2 3).(# ])

[["V", "V"], "A", ["V", "A"]], // pipeToArray [.(+1 -2) -- OLD (??)A(?A)

[["V", "V", "V"], "A", ["V", "V", "A"]], // pipeBinaryToArray :.(+$ -$) -- OLD (???)A(??A)

[["X", "Y", "Z"], ["Z", "W"], ["X", "Y", "W"]], // binaryUnaryPipe :.+$

[["X", "Y"], ["Y", "Z", "W"], ["X", ["Z", "W"]]], // unaryBinaryPipe +1./

[["X", "Y"], ["Y", "Z"], ["X", "Z"]], // pipe +1./2

];

plus.types = [

["N", "V", "N"], // add 2+3

["S", "V", "S"], // stringConcat ""+4

["A", "A", "A"], // arrayConcat (1 2 3)+(4 5 6)

["O", "O", "O"], // merge {"{a: 1}"+({"{b: 2}")

];

slash.types = [

["N", "N", "N"], // divide 8/2

[["V", "S"], "A", "O"], // groupBy [/("ann" "ben" "ade")

];

less.types = [

["N", "N", "B"], // lessThan 2<3

["S", "S", "B"], // lessThanString "abc"<"bcd"

[["V", "S"], "A", "A"], // sort ;<("dan" "sue" "alan")

[["V", "N"], "A", "A"], // sort ;<(1 2 3)

];

greater.types = [

["N", "N", "B"], // greaterThan 3>2

["S", "S", "B"], // greaterThanString "bcd">"abc"

["A", "O", "O"], // over ((1 ) +1)>(3 5 7)

["A", "A", "A"], // over (("a" ) +1)>{({"a": 1})

["V", ["V", "V"], "V"], // tap 3>({"console.log")

];

minus.types = [

["N", "N", "N"], // subtract 5-2

["S", "O", "O"], // omitKey "a"-({"{a: 1}")

["A", "O", "O"], // omitKeys ("a" "b")-({"{a: 1, b: 2}")

["A", "A", "A"], // splice (1 2 3 4)-(5 6 7 8)=(5 3 4 8)

["A", "S", "S"], // splice (3 2 "le")-"nucular"="nuclear"

];

colon.types = [

["?", "?", "A"], // pair +:2

];

question.types = [

["A", "V", "V"], // cond ((<10 +1) -1)?15

["N", "N", "N"], // random 1#100

[["V", "V"], "A", "N"], // findIndex (%2.=0)?(1 2 3 4)

];

atsign.types = [

[["V", "V"], "A", "A"], // map \*2@(3 4 5)

[["V", "V", "V"], "A", "A"], // mapBinary =@(2 3 4)

[["V", "V"], "O", "O"], // mapObject \*2@({"{a: 1, b: 2, c: 3}")

["A", "S", "S"], // stringReplace ("\_" "-")@"1 0 \_1"

["V", "A", "N"], // indexOf 2@(6 8 2 3)

["S", "S", "N"], // indexOf "bc"@"abcd"

//[["V", "S", "V"], "O", "O"] // mapObjectIndexed

];

asterisk.types = [

["N", "N", "N"], // times 2\*3

["A", "O", "O"], // pick ("a" "c" "d")\*(\(("a" 1) ("b" 2) ("c" 3)))

[["V", "B"], "A", "A"], // filter <5\*(4 9 2 7 3)

];

dollar.types = [

[["?", "?", "X"], "A", "X"], // insert +$(1 2)

["S", "A", "S"], // join ","$(1 2 3)

["A", "A", "?"], // reduce (+ 0)$(1 2 3)

//[[["X", "Y"], "X", "Y"], "A", "Y"],

//[[["X", "Y"], ["X", "Y"], ["X", "Y"]], "A", ["X", "Y"]],

];

apostrophe.types =[

["N", "N", "N"], // round 3'3.14196

["N", "A", "?"], // at 1'(1 2 3)

["N", "S", "S"], // at 1'"abc"

["S", "O", "?"], // prop "a"'{({"a": 1})

["A", "A", "?"], // path (1 )'(5 6 7)

["A", "O", "?"], // path ("a" )'{({"a": 1})

[["V", "V"], "A", "V"], // find (%2.=0)'(1 2 3)

];

bar.types = [

["V", "V", "V"], // orValue !()|()

[["V", "V"], ["V", "V"], ["V", "V"]], // orPredicate >0|(%2.=0)

[["V", "V", "V"], ["V", "V", "V"], ["V", "V", "V"]] // orBinary <|=

];

percent.types = [

["N", "N", "N"], // modulo 7%2

["N", "A", "A"], // split 2%(1 2 3 4 5)

["N", "S", "A"], // split 2%"abcde"

["A", "A", "A"], // chunk (1 2 0)%(1 2 3 4 5)

["A", "S", "A"], // chunk (1 2 0)%"abcde"

["S", "S", "A"], // chunkWithDelimiter ", "%"1, 2, 3, 4"

[["V", "B"], "A", "A"], // chunkWhenPredicate =2%(1 2 3 2 1)

[["S", "B"], "S", "A"], // chunkWhenPredicate ="b"%"abcbe"

[["V", "V", "B"], "A", "A"], // chunkWhenComparator <%(1 2 3 2 1)

[["S", "S", "B"], "S", "A"], // chunkWhenComparator <%"abcba"

];

hat.types = [

["N", "N", "N"], // power 2^3

[["N", "?"], "N", "A"], // generate ;^3

["A", "A", "A"], // scan (#.<5 #.+1)^( )

[["X", "V"], ["X", "Y"], ["X", "Y"]], // while 1,(<10^(\*2))

];

ampersand.types = [

["V", "V", "V"], // andValue !()&()

[["V", "V"], ["V", "V"], ["V", "V"]], // andPredicate >2&(<6)

];

backtick.types = [

["X", "?", "X"], // constant 2`3

];

UNARY

tilde.types = [

["A", "A"], // transpose ~((1 2) (3 4))

[["X", "Y", "Z"], ["Y", "X", "Z"]], // flip ~/

];

underscore.types = [

["N", "N"], // negative \_5

["A", "A"], // reverse \_(1 2 3)

["S", "S"], // reverse \_"Hello"

];

bracketleft.types = [

["A", "?"], // first [(1 2 3)

["S", "S"], // firstInString ["abc"

["N", "N"], // floor [1.8

];

bracketright.types = [

["A", "?"], // last ](1 2 3)

["S", "S"], // lastInString ]"abc"

["N", "N"], // ceiling ]1.2

];

hash.types = [

["A", "N"], // arrayLength #(4 5 6)

["S", "N"], // stringLength #"abcd"

["O", "N"], // keyLength #({"{a: 1}")

["N", "N"], // modulus #(\_1.5)

];

backslash.types = [

["A", "O"], // fromPairs \(("a" 1) ("b" 2))

["O", "A"], // toPairs \({"{a: 1, b: 2}")

];

braceleft.types = [

["S", "?"], // eval {"Math.sqrt(2)"

["A", "A"], // unnest {(1 (2 3))

];

semicolon.types = [

["X", "X"], // identity ;1

];

braceright.types = [

["?", "S"], // typeof }3

];

bang.types = [

["V", "B"], // not !2

[["V", "V", "V"], ["V", "V", "B"]], // not !<

[["V", "V"], ["V", "B"]], // not !(<2)

]