Dyalog North America – 29-30 September 2025

Static Analysis of APL for Tooling and Compliance

Aaron Hsu *Brandon Wilson*

aaron@dyalog.com *brandon@dyalog.com*

Static Analysis

****** ********

When running your code isn’t enough

“Reading your code”

Checking

Changing

Tool Support

for Static Analysis

Getting you started now…

And a peek at the future

Make it easier to answer

questions about your code

What kind of questions, I hear you say?

Questions about your code

- Do I have singletons living in my code that might surprise me later?
- Do I have any code that depends on certain errors? (Complex, 128-bit floats, and returning nested results)
- Does my code handle empty array prototypes (Reduce, Each, Rank) correctly?
- Have I sanitized my data coming from unverified sources?
- Have I ensured data security between my client data?
- Do I have hidden dependencies on index origin?
- Do I index all my database arrays using appropriate field names?

More questions…

- Am I using some old idioms that can be replaced by newer primitives?
- Do I have naked strand assignments?
- Are my strand assignments vulnerable to being confused for modified assignments?
- Do I make dangerous assumptions about empty arrays that aren't justified?
- Can I convert functions back and forth between trad-fns and dfns safely?
- Am I passing an operator a function that unintentionally captures shadowed names in the operator?
- Do I have any strand assignments against vectors that might not be the right length?

Even more questions…

- Do I have any rank or type errors sitting around?
- Do have code whose empty prototype has the wrong element type? E.g., ‘’ vs ⍬
- Is this loop iteration independent (I can parallelize it)?
- Is it safe to reorder these statements?
- Does this function set any of my locally shadowed bindings?
- Do I have any duplicate names or labels?
- Where am I using a function via alias instead of via namespace reference? F←ns.F

Even even more questions….

- Do I have any race conditions between threads?
- Do I depend on variables that might not exist?
- Do I have dead or useless code that is sitting around unused or secretly broken?
- Are there unused variables in my function that I have shadowed?
- Do I have unshadowed variables that really should be shadowed?
- Do I bind a local variable in a dfn but use it before it is shadowed?

Crystal clear, eh?

Let’s go looking for trouble…

Looking for trouble…

```
+⌿(QUERY_MASK≠DB)[;1]
```

```
+⌿(QUERY_MASK≠DB)[;1]
                  ↑
                  1
```

1. Magic numbers for column indexing

```
+⌿(QUERY_MASK⌿DB)[;AGES]
                  ↑
                  1
```

1. Magic numbers for column indexing

```
+⌿(QUERY_MASK⌿DB)[;AGES]
                  ↑    ↑
                  2    1
```

1. Magic numbers for column indexing
2. What if we want to convert to an inverted table?

```
+⌿(QUERY_MASK≠DB)[;AGES]
          ↑  ↑     ↑
          3  2     1
```

1. Magic numbers for column indexing
2. What if we want to convert to an inverted table?
3. Is this a filter or a reduction?

```
+⌿(QUERY_MASK⌿DB)[;AGES]
        ↑       ↑ ↑   ↑
        4       3 2   1
```

1. Magic numbers for column indexing
2. What if we want to convert to an inverted table?
3. Is this a filter or a reduction?
4. What if `QUERY_MASK⌿` were a complex expression?

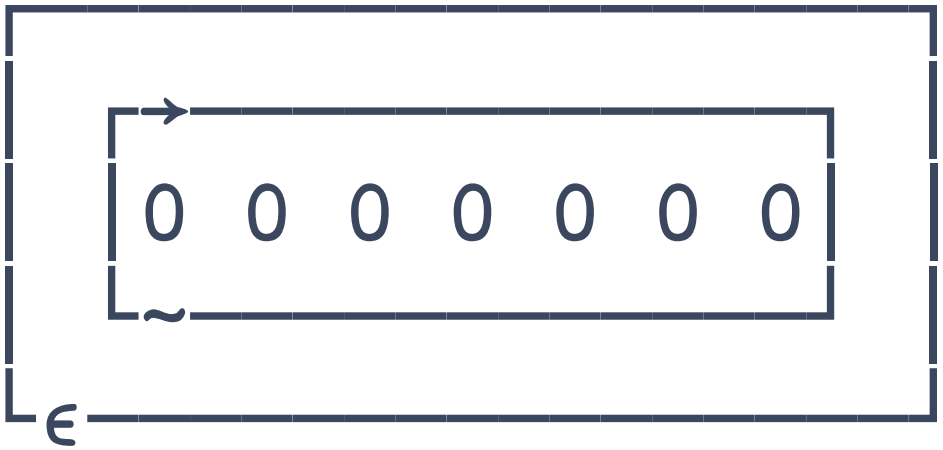
```
+⌿(QUERY_MASK⌿DB)[;AGES]
  ↑       ↑       ↑ ↑   ↑
  5       4       3 2   1
```

1. Magic numbers for column indexing
2. What if we want to convert to an inverted table?
3. Is this a filter or a reduction?
4. What if `QUERY_MASK⌿` were a complex expression?
5. …

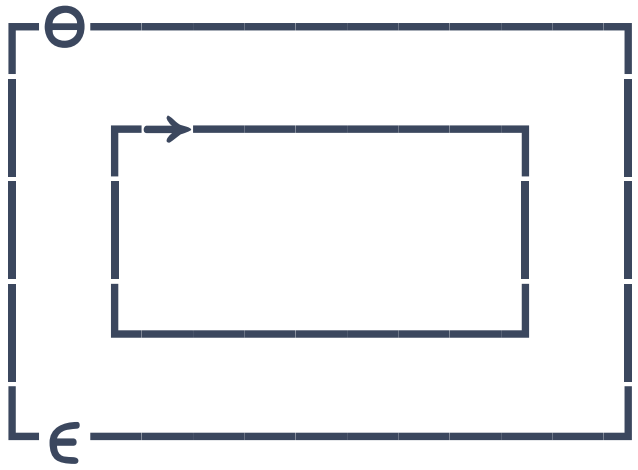
```
      DB←3 2⍴'brandon' 2 'aaron' 12
      QUERY_MASK←DB[;0]∊'brandon' 'aaron'
      (QUERY_MASK≠DB)[;1]
2 12 2
      +⌿(QUERY_MASK≠DB)[;1]
16
```

```
QUERY_MASK←DB[;0]∈⊂'morten'
+⌿(QUERY_MASK⌿DB)[;1]
```

```
QUERY_MASK←DB[;0]∈⊂'morten'
+⌿(QUERY_MASK≠DB)[;1]
```



```
QUERY_MASK←DB[;0]∈⊂'morten'
(QUERY_MASK≠DB)[;1]
```



```
+⌿(QUERY_MASK⌿DB)[;AGES]
  ↑       ↑       ↑ ↑   ↑
  5       4       3 2   1
```

1. Magic numbers for column indexing
2. What if we want to convert to an inverted table?
3. Is this a filter or a reduction?
4. What if `QUERY_MASK⌿` were a complex expression?
5. **Empty-array prototypes must be considered**

Find this code…

+⌿(QUERY_MASK⌿DATABASE)[;AGES]

```
(ast _ symbol)←3↑⎕codfns.PS src
(parent _ type kind name _ vardef _ _)←ast

expr←type∊Strand Expr
roots←⍸parent=⍳≠parent
dexpr←(type=Expr)∧kind=2
moxpr←(type=Oper)∧kind∊1 2
defs←(type=Var)∧type[parent]=Header
namespace←(type=Trad)∧kind=0
ns←{⍺[⍵]}⍣{~namespace[⍵]}⍤≡⍨parent
prmfn←(type=Prim)∧kind=2
prmop←(type=Prim)∧kind=3
named←{name=symbol⍳⊂,⍵}
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍸defs^(ns∊roots)^named'DB'
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace

db←⍳⍣defs⍣(ns∊roots)⍣named'DB'
```

```
:Namespace
  ⍝DB←LD'DYNA'
  ⍝QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  ⍝+⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace

db←⍳defs∧(ns∊roots)∧named'DB'
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace

db←⍳defs⍎(ns∊roots)⍎named'DB'
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace

db←⍸defs∧(ns∊roots)∧named'DB'
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∈(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳defs∧(ns∈roots)∧named'DB'
dbex←(type=Var)∧(vardef∈db)∧expr[parent]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳defs⍤(ns∊roots)⍤named'DB'
dbex←(type=Var)⍤(vardef∊db)⍤expr[parent]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳defs∧(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∈(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳defs∧(ns∈roots)∧named'DB'
dbex←(type=Var)∧(vardef∈db)∧expr[parent]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳defs⍤(ns∊roots)⍤named'DB'
dbex←(type=Var)⍤(vardef∊db)⍤expr[parent]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍸defs∧(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∈(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK≠DB)[;AGES]
:EndNamespace
```

```
db←⍳⍨defs∧(ns∈roots)∧named'DB'
dbex←(type=Var)∧(vardef∈db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍸defs∧.(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∈(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK≠DB)[;AGES]
:EndNamespace
```

```
db←⍳defs∧(ns∈roots)∧named'DB'
dbex←(type=Var)∧(vardef∈db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∈(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳defs⍤(ns∈roots)⍤named'DB'
dbex←(type=Var)⍤(vardef∈db)⍤expr[parent]
{dbex[parent]∨←dbex⍤expr[parent]}⍣≡⍬
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∈(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳defs⍤(ns∈roots)⍤named'DB'
dbex←(type=Var)⍤(vardef∈db)⍤expr[parent]
{dbex[parent]∨←dbex⍤expr[parent]}⍣≡⍬
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳⍨defs∧.(ns∊roots)∧named'DB'
dbex←(type=Var)∧.(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳⍨prmfn∧.(named'[')∧dexpr[parent]]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳⍨defs∧.(ns∊roots)∧.named'DB'
dbex←(type=Var)∧.(vardef∊db)∧.expr[parent]
{dbex[parent]∨←dbex∧.expr[parent]}⍣≡⍬
idxpr←parent[⍳⍨prmfn∧.(named'[')∧.dexpr[parent]]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳⍨defs∧.(ns∊roots)∧named'DB'
dbex←(type=Var)∧.(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳⍨prmfn∧.(named'[')∧dexpr[parent]]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳defs∧(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳prmfn∧(named'[')∧dexpr[parent]]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳defs∧(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳prmfn∧(named'[')∧dexpr[parent]]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳defs∧(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳prmfn∧(named'[')∧dexpr[parent]]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK≠DB)[;AGES]
:EndNamespace
```

```
db←⍳⍨defs∧.(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳⍨prmfn∧(named'[')∧dexpr[parent]]
idxpr_db←parent[⍳⍨dbex∧(≠parent)∧parent∊idxpr]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∈(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳⍨defs∧.(ns∈roots)∧named'DB'
dbex←(type=Var)∧(vardef∈db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳⍨prmfn∧(named'[')∧dexpr[parent]]
idxpr_db←parent[⍳⍨dbex∧(≠parent)∧parent∈idxpr]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳defs∧(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳prmfn∧(named'[')∧dexpr[parent]]
idxpr_db←parent[⍳dbex∧(≠parent)∧parent∊idxpr]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳⍨defs∧(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳⍨prmfn∧(named'[')∧dexpr[parent]]
idxpr_db←parent[⍳⍨dbex∧(≠parent)∧parent∊idxpr]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳defs∧(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳prmfn∧(named'[')∧dexpr[parent]]
idxpr_db←parent[⍳dbex∧(≠parent)∧parent∊idxpr]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK≠DB)[;AGES]
:EndNamespace
```

```
db←⍳⍨defs∧.(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳⍨prmfn∧(named'[')∧dexpr[parent]]
idxpr_db←parent[⍳⍨dbex∧(≠parent)∧parent∊idxpr]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK≠DB)[;AGES]
:EndNamespace
```

```
db←⍳defs∧(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳prmfn∧(named'[')∧dexpr[parent]]
idxpr_db←parent[⍳dbex∧(≠parent)∧parent∊idxpr]
redox←parent[⍳prmop∧(named'≠')∧moxpr[parent]]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK≠DB)[;AGES]
:EndNamespace
```

```
db←⍳⍨defs∧.(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳⍨prmfn∧(named'['⍤)∧dexpr[parent]]
idxpr_db←parent[⍳⍨dbex∧(≠parent)∧parent∊idxpr]
redox←parent[⍳⍨prmop∧(named'≠'⍤)∧moxpr[parent]]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK≠DB)[;AGES]
:EndNamespace
```

```
db←⍳defs∧(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳prmfn∧(named'[')∧dexpr[parent]]
idxpr_db←parent[⍳dbex∧(≠parent)∧parent∊idxpr]
redox←parent[⍳prmop∧(named'≠')∧moxpr[parent]]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK≠DB)[;AGES]
:EndNamespace
```

```
db←⍳defs∧(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳prmfn∧(named'[')∧dexpr[parent]]
idxpr_db←parent[⍳dbex∧(≠parent)∧parent∊idxpr]
redox←parent[⍳prmop∧(named'≠')∧moxpr[parent]]
parent[redox]∩parent[idxpr_db]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK≠DB)[;AGES]
:EndNamespace
```

```
db←⍳⍨defs∧.(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳⍨prmfn∧(named'[')∧dexpr[parent]]
idxpr_db←parent[⍳⍨dbex∧(≠parent)∧parent∊idxpr]
redox←parent[⍳⍨prmop∧(named'≠')∧moxpr[parent]]
parent[redox]∩parent[idxpr_db]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳defs∧(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳prmfn∧(named'[')∧dexpr[parent]]
idxpr_db←parent[⍳dbex∧(≠parent)∧parent∊idxpr]
redox←parent[⍳prmop∧(named'⌿')∧moxpr[parent]]
parent[redox]∩parent[idxpr_db]
```

```
:Namespace
  DB←LD'DYNA'
  QUERY_MASK←DB[;EMAIL]∊(LD'DYALOG')[;EMAIL]
  +⌿(QUERY_MASK⌿DB)[;AGES]
:EndNamespace
```

```
db←⍳⍨defs∧.(ns∊roots)∧named'DB'
dbex←(type=Var)∧(vardef∊db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍳⍨prmfn∧(named'['⍱.)∧dexpr[parent]]
idxpr_db←parent[⍳⍨dbex∧(≠parent)∧parent∊idxpr]
redox←parent[⍳⍨prmop∧(named'⌿')∧moxpr[parent]]
parent[redox]∩parent[idxpr_db]
```

We could refactor…

- `(QUERY_MASK≠DATABASE)[;AGES]`
- `QUERY_MASK≠DATABASE[;AGES]`
- `AGES⊃QUERY_MASK∘≠¨DATABASE`

- C.f. - Asher on Trees/Parent Vectors

```
QUERY_SAVE←{'Q1'STORE ⍺⍵}
RESULTS←MASK QUERY_SAVE DATABASE
AVG_AGES←(+⌿÷≢)(LOAD 'Q1')[;AGES]
```

```
QUERY_SAVE←{'Q1'STORE ⍺≠⍵}
RESULTS←MASK QUERY_SAVE DATABASE
AVG_AGES←(+⌿÷≢)(LOAD 'Q1')[;AGES]
```

```
db←⍸(ns∈roots)∧defs∧named'DB'
dbex←(type=Var)∧(vardef∈db)∧expr[parent]
{dbex[parent]∨←dbex∧expr[parent]}⍣≡⍬
idxpr←parent[⍸prmfn∧(named'[')∧dexpr[parent]]
idxpr_db←parent[⍸dbex∧(≠parent)∧parent∈idxpr]
redox←parent[⍸prmop∧(named'⌿')∧moxpr[parent]]
parent[redox]∩parent[idxpr_db]
```

```
QUERY_SAVE←{'Q1'STORE α≠ω}
RESULTS←MASK QUERY_SAVE DATABASE
AVG_AGES←(+⌿÷≢)(LOAD 'Q1')[;AGES]
```

```
db←⍳(ns∈roots)^defs^named'DB'
dbex←(type=Var)^(vardef∈db)^expr[parent]
{dbex[parent]∨←dbex^expr[parent]}⍣≡⍬
idxpr←parent[⍳prmfn^(named'[')^dexpr[parent]]
idxpr_db←parent[⍳dbex^(≠parent)^parent∈idxpr]
redox←parent[⍳prmop^(named'⌿')^moxpr[parent]]
parent[redox]∩parent[idxpr_db]
```

```
QUERY_SAVE←{'Q1'STORE ⍺≠⍵}
RESULTS←MASK QUERY_SAVE DB
AVG_AGES←(+⌿÷≢)(LOAD 'Q1')[;AGES]

db←⍸(ns∊roots)∧defs∧named'DB'
dbex←expr∧uses∨.∧used∊db

idxpr←parent[⍸prmfn∧(named'[')∧dexpr[parent]]
idxpr_db←parent[⍸dbex∧(≠parent)∧parent∊idxpr]
redox←parent[⍸prmop∧(named'⌿')∧moxpr[parent]]
⍸(uses∊redox)∧uses∨.∧used∊idxpr_db
```

```
QUERY_SAVE←{'Q1'STORE α≠ω}
RESULTS←MASK QUERY_SAVE DB
AVG_AGES←(+⌿÷≢)(LOAD 'Q1')[;AGES]
```

```
db←⍳(ns∊roots)∧defs∧named'DB'
dbex←expr∧uses∨.∧used∊db

idxpr←parent[⍳prmfn∧(named'[')∧dexpr[parent]]
idxpr_db←parent[⍳dbex∧(≠parent)∧parent∊idxpr]
redox←parent[⍳prmop∧(named'⌿')∧moxpr[parent]]
⍳(uses∊redox)∧uses∨.∧used∊idxpr_db
```

```
QUERY_SAVE←{'Q1'STORE α≠ω}
RESULTS←MASK QUERY_SAVE DB
AVG_AGES←(+⌿÷≢)(LOAD 'Q1')[;AGES]
```

```
db←⍳(ns∊roots)^defs^named'DB'
dbex←expr^uses∨.^used∊db
```

```
idxpr←parent[⍳prmfn^(named'[')^dexpr[parent]]
idxpr_db←parent[⍳dbex^(≠parent)^parent∊idxpr]
redox←parent[⍳prmop^(named'⌿')^moxpr[parent]]
⍳(uses∊redox)^uses∨.^used∊idxpr_db
```

```
QUERY_SAVE←{'Q1'STORE α≠ω}
RESULTS←MASK QUERY_SAVE DB
AVG_AGES←(+⌿÷≢)(LOAD 'Q1')[;AGES]

db←⍳(ns∊roots)^defs^named'DB'
dbex←expr^uses⍣.^used∊db

idxpr←parent[⍳prmfn^(named'[')^dexpr[parent]]
idxpr_db←parent[⍳dbex^(≠parent)^parent∊idxpr]
redox←parent[⍳prmop^(named'⌿')^moxpr[parent]]
⍳(uses∊redox)^uses⍣.^used∊idxpr_db
```

```
QUERY_SAVE←{'Q1'STORE α≠ω}
RESULTS←MASK QUERY_SAVE DB
AVG_AGES←(+⌿÷≢)(LOAD 'Q1')[;AGES]

db←⍸(ns∊roots)∧defs∧named'DB'
dbex←expr∧uses∨.∧used∊db

idxpr←parent[⍸prmfn∧(named'[')∧dexpr[parent]]
idxpr_db←parent[⍸dbex∧(≠parent)∧parent∊idxpr]
redox←parent[⍸prmop∧(named'⌿')∧moxpr[parent]]
⍸(uses∊redox)∧uses∨.∧used∊idxpr_db
```

```
QUERY_SAVE←{'Q1'STORE ⍺≠⍵}
RESULTS←MASK QUERY_SAVE DB
AVG_AGES←(+⌿÷≢)(LOAD 'Q1')[;AGES]
```

```
db←⍸(ns∊roots)∧defs∧named'DB'
dbex←expr∧uses∨.∧used∊db

idxpr←parent[⍸prmfn∧(named'[')∧dexpr[parent]]
idxpr_db←parent[⍸dbex∧(≠parent)∧parent∊idxpr]
redox←parent[⍸prmop∧(named'⌿')∧moxpr[parent]]
⍸(uses∊redox)∧uses∨.∧used∊idxpr_db
```

Static Analysis Tool for APL Notation

Special Analyses

Code Quality Rules

Security Rules

Tracing data flow for you

Migration Assistance

A set of common rules and checks

Security Vulnerabilities

Support Our Customers

around Compliance and Code Quality Audits

Standardization of Security Habits

Integration with Industry Products

(SonarQube, GitGuardian, Coverity, *etc*.)

Make things easier…

Easy customization

Easy application integration

Easy training, onboarding

Easy deployment

Provide your customers

with additional confidence and trust

Thank You.

aaron@dyalog.com brandon@dyalog.com

https://github.com/Co-dfns/Co-dfns