DYALOG
Eastbourne 2026

Dyalog Static Analysis Tool

In Practice and in the Future



Brandon Wilson

brandon@dyalog.com

Who?

Yadda yadda

- Lead developer of Dyalog's static analysis tool
- Japan for 16 years
- APL for ~5
- Dyalog for ~2.5
- Interested in data-driven design in APL
- and formal program analysis

Static What? Why?

We all have questions

We all have questions about our code

We all have questions
about our code
and our development processes

Static analysis is a tool in your arsenal

Like what?

Development Concerns

- Is this use of Execute always safe?
- Does this APL+Win code also work with Dyalog?
- Does my code have runtime syntax errors?
- Where are these semi-globals defined?
- Am I using cryptographic primitives correctly?
- Is there a better way to write this?

But wait. There's more!

Development Concerns

- 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?

Development Concerns

- 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?

Development Concerns

- 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`

Development Concerns

- 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?

You get the point

But those are just developer issues

Management Concerns

- Is my codebase healthy?
- Do we have lots of duplicate code?
- What parts of our codebase are the most complex?
- Where are our choke points?
- Are we respecting module boundaries?

There are bigger questions, too

Executive Concerns

- What security standards do we meet?
- Where is my audit log?
- Do we meet compliance requirements?
- Are we ensuring maintainable code?
- What remediation process do we have?

But what about the African elephants?

AI Concerns

- Can I make Claude write better APL?
- How can I reduce token usage?
- What can I do about context drift?
- Where are my verified sanity checks?
- Will this output hold up to policy?

That's why

Presenting...

Dyalog Static Analysis Tool

Dyalog Static Analysis Tool

DSAT

Are there other tools?

Competing Products

- SonarQube? No APL support
- OpenText Fortify? No APL support
- Veracode? No APL support
- CodeQL? Semgrep? Codacy? No APL support

Competing Solutions?

- SonarQube? No APL support
- OpenText Fortify? No APL support
- Veracode? No APL support
- CodeQL? Semgrep? Codacy? No APL support

Competing Solutions?

- Some impressive in-house tooling
- *Ad hoc* regex searches
- Yeet it at AI
- Tell the auditors that APL really, truly is special
- Development culture and discipline
- …

Are there other tools?
Sort of?

So, what can DSAT do now?

Current Ruleset—Security

- DSAT-0101 Insecure Connections
- DSAT-0102 Insecure Cryptography
- DSAT-0103 Hard-Coded Secrets
- DSAT-0104 Code Injection

Current Ruleset—Cleanliness

- DSAT-0105 Leaked Locals
- DSAT-0106 Duplicate Functions
- DSAT-0107 Shadow After Usage
- DSAT-0108 Naked strand assignments
- DSAT-0109 Unbound Variables

Current Ruleset—Errors

- DSAT-0001 UNBALANCED STRING
- DSAT-0020 INVALID SYSTEM NAME
- DSAT-0030 UNBALANCED OPEN PARENS/BRACKETS
- DSAT-0033 EMTPY ASSIGNMENT TARGET/VALUE
- …
- 43 in total

Provisional Ruleset—Migration

- Hybrid operators, *e.g.*, `a b/¨c d ↠ a b∘/¨c d`
- Bracket Indexing, *e.g.*, `a b[i] ↠ a (b[i])`
- Multiassignment, *e.g.*, `a b←c d ↠ a (b←c d)`

Coverage

- 54 DSAT rules in total
- 1 DSAT ~ 5 SonarQube
  - DSAT-0102 ≡ 2 SonarQube rules
  - DSAT-0104 ≡ 18 SonarQube rules
- ~40 security, ~25 maintenance, ~150 linting equiv.
- ~1/5 SonarQube coverage

Show me, don't tell me!

Summary Reports

```
      SUMMARY
10   DSAT-0001   UNBALANCED STRING
2    DSAT-0002   INVALID CHARACTER IN SOURCE
0    DSAT-0003   MULTIPLE . IN FLOAT
101  DSAT-0004   ¯ CANNOT APPEAR BETWEEN DIGITS
⍝ ∘∘∘
```

Audit Logs

```
out←'rules.csv' 'issues.csv'
out CSV REPORT NSDIR 'src'

json←SONARΔQUBE REPORT NSDIR 'src'
json □NPUT 'report.json'
```

Investigative Tooling

```
      2↑VIEW'DSAT-0020'
src/f.aplf[2]
y←⎕INVALID input

src/g.aplf[2]
y←⍬,⎕INVALID ⍬
```

Ruleset Tailoring

```
included←'DSAT-0020' 'DSAT-0039'
AST 1 included REPORT NSDIR 'src'

excluded←'DSAT-0001' 'DSAT-0040'
AST 0 excluded REPORT NSDIR 'src'
```

Whitelisting code

```
x←⍬⍬⍬      ⍝DSAT ignore
y←9 ⋄ z←   ⍝DSAT ignore DSAT-0033
f←{ ⍝DSAT ignore block
  ⍵⍵⍵
}
```

Some Upcoming Functionality

Investigative Tooling

```
      COMPLEXITY
Cyclomatic          14
Cognitive           14
Halstead            12
Function Length     103
Duplication %       0.3
```

Investigative Tooling

```
      COMPLEXITY 'someFun'
Cyclomatic        9
Cognitive         9
Halstead          7
Function Length   37
Duplication %     0.0
```

Investigative Tooling

```
      '#.entryFn' CALLGRAPH 'ns.calledFn'
#.entryFn[31] #.thingify[2] ns.soup[9]
#.entryFn[31] #.doX[101] ns.f[3] ns.g[5]
#.entryFn[42]
```

Investigative Tooling

```
      '#.entryFn' GLOBALSΔSET 'ns.calledFn'
#.entryFn[17] someVar←'Enough is a feast'
#.thingify[1] another←'Lagom är bäst'
#.thingify[2] varVarX←'中庸之道'
```

Future Directions

- Interpreter affordances
- VSCode integration
- Mechanical migration tooling
- Array notation support
- .NET analysis
- Expanded ruleset

DSAT provides insights

DSAT is a tool in your arsenal

But I've got lots of code, and
A million warnings is useless.

Glad you asked

Incremental Adoption

- Filter ruleset as needed
- Whitelist code with ignore directives
- Custom tailored rules
- Non-invasive investigative tools
- Happy to provide support

Soft adoption

Simple Adoption

- Dyalog 18.2+ interpreter
- Codebase as text
- Computer or Docker container
- Tutorials, Reference documentation, HowTos, *etc.*
- One-on-one support as needed

Just give us a shout

Acknowlegements

- **Moris Zucca**, *Simcorp*, for pointed advice
- **Nunzio Turtulici**, *Simcorp*, for detailed feedback
- **Lars Villadsen**, *Simcorp*, for bold initial trials and feedback
- **Gilgamesh Athoraya**, *Tiamatica*, for insightful suggestions
- **Morten Kromberg**, *Dyalog*, for clear direction and steering
- **Aaron Hsu**, *Dyalog*, for detailed technical discussions