DYALOG

Glasgow 2024

ullu: Tests for Dyalog APL

Aarush Bhat

Dyalog Ltd.

Coming up...

- What is ullu?
- How has Dyalog been tested so far?
- Why and How ullu was created?
- What is the future of ullu?

What is ullu

Pre-ullu

- Standard QAs
- Shuffle QAs
- Unit Tests
- Fuzz tests
- Code Coverage tests

Versions and Platforms

- 6½ platforms – Windows, Mac, Mac/Arm, AIX, Pi, Linux, Linux/Arm(under dev)
- 4 versions × 6½ platforms × 32/64 × Classic/Unicode
- That is… 63 interpreters of Dyalog APL

With ullu

What does it do?

- Verify correct results using fundamental scenarios
- Test edge cases
- Test all the data types possible, being reproducible and truly randomised
- Finally, inspect code coverage to find missing test cases

Battle testing

Image generated by GPT-4o

Primitives Covered – as of 1/9/24 - 11

- add (dyadic +)
- divide (dyadic ÷)
- floor (monadic ⌊)
- magnitude (monadic |)
- residue (dyadic |)
- subtract (dyadic -)
- unique (monadic ∪)
- unique mask (monadic ≠)
- index of (dyadic ⍳)
- membership (dyadic ∈)
- multiply (dyadic ×)

Next up

- greater (dyadic >)
- less (dyadic <)
- exponential (dyadic *)
- grade (⍋ ⍒)
- not equal (dyadic ≠)
- not (monadic ~)
- union (dyadic ∪)

Process

- Define fundamental scenarios
- Explore the codebase
- Check code coverage
- Check user issues
- Verify interpreter limits
- Check displayed errors
- Remove dead code

Model functions

- Unique R←∪Y
  - `{0=≢⍵:⍵ ⋄ ↑,⊃{⍺,(∧/⍺≢¨⍵)/⍵}⍨⌸⊢⍵}`
- Add R←X+Y
  - `{⍺--⍵}`
- Magnitude R←|Y
  - `{⍵×(¯1@(∊∘0)(⍵>0))}`

Tests

- Unique R←∪Y
  - `Assert (≢data)≥≢∪data`
- Add R←X+Y
  - `RunVariations (2×data) data data`
- Magnitude R←|Y
  - `RunVariations (modelMag data) data`

RunVariations

- Normal array (one or two datatypes mixed)
- Scalar
- Empty array
- Differently shaped array
- Testing with model
- Testing with completely randomised data with model

Data, more data and results

- 84,274 tests
- Lots of numbers, arrays and data elements
- Good news, ullu is now integrated with the standard and coverage QAs

Code coverage

```
16  switch (mult)
16  {
 0  case 1: b_times_scal(bound, type);
 0    break;
 2  case 2: if (Larg==Bool0)
 2      zero_each(bound);
 2    else
 2      Rslt = zap(Rarg);
 2    break;
14  case 3: calc_rslt(type, type, APLBOOL, bound,
14        mulbsfns[type-APLSINT]);
14    break;
16  }
```

	Function Coverage	Line Coverage	Region Coverage	Branch Coverage
src/same_ibeam.c	66.67% (2/3)	39.73% (29/73)	38.67% (58/150)	32.00% (16/50)
src/same_ibeam.h	100.00% (2/2)	100.00% (6/6)	100.00% (2/2)	- (0/0)
src/same_template.h	100.00% (3/3)	86.05% (259/301)	53.18% (643/1209)	45.33% (340/750)
src/scalar.h	100.00% (1/1)	82.35% (14/17)	100.00% (4/4)	100.00% (2/2)
src/scalarscalar.cpp	92.16% (94/102)	83.90% (469/559)	80.72% (1394/1727)	82.71% (220/266)
src/scald.cpp	100.00% (36/36)	98.61% (995/1009)	96.88% (1953/2016)	93.40% (835/894)
src/scalm.cpp	95.35% (123/129)	97.17% (789/812)	96.33% (1522/1580)	89.72% (506/564)
src/scan.c	100.00% (60/60)	95.74% (1820/1901)	93.34% (1304/1397)	89.33% (804/900)

Future plans

- Cover all primitives
- Work with more data
- Find cases that people might not hit in a 100 years
- Make Dyalog APL more reliable

Find ullu at: github.com/Dyalog/ullu

Reach out: aarush@dyalog.com

In short, ullu is...

- New
- Structured
- Independent
- A 2nd coat of paint

It is not just about finding things wrong about Dyalog APL but making us more confident when making enhancements