Posix-Nexus AWK

Canine-Table

March 24, 2025

Contents

1 Algorithms

2 Boolean

Boolean Operations

The following functions provide utilities for logical and comparative operations, enabling versatile Boolean checks across various conditions.

- **NOT__(D)**: Returns the logical NOT of D.
- **NULL__(D)**: Returns the logical NOT of D (equivalent to **NOT__**).
- **FULL__(D)**: Determines whether D is full (non-empty).
- **TRUE__(D, B)**: Returns 1 if D is full or valid based on B.
- **FALSE__(D, B)**: Returns the logical NOT of **TRUE__**.
- **OR__(B1, B2, B3)**: Logical OR operation between B1 and B2 based on B3.
- **NOR__(B1, B2, B3)**: Logical NOR operation, the NOT of **OR__**.
- **AND__(B1, B2, B3)**: Logical AND operation between B1 and B2 based on B3.
- **NAND__(B1, B2, B3)**: Logical NAND operation, the NOT of **AND__**.
- **XOR__(B1, B2, B3)**: Logical XOR operation, true if exactly one of B1 or B2 is true.
- **XNOR__(B1, B2, B3)**: Logical XNOR operation, the NOT of **XOR__**.
- **CMP__(B1, B2, B3, B4)**: Compares B1 and B2 based on conditions B3 and B4.
- **NCMP__(B1, B2, B3, B4)**: Logical NOT of **CMP__**.
- **LOR__(B1, B2, B3, M)**: Logical OR based on modes specified in M.
- **EQ__(B1, B2, B3)**: Determines equality between B1 and B2 based on B3.
- **NEQ__(B1, B2, B3)**: Determines inequality (NOT equal) between B1 and B2 based on B3.
- **IEQ__(B1, B2, B3)**: Case-insensitive equality comparison between B1 and B2.

Boolean Operations

- **INEQ__(B1, B2, B3)**: Logical NOT of **IEQ__**.
- **GT__(B1, B2, B3)**: Returns `true` if B1 is greater than B2.
- **LT__(B1, B2, B3)**: Returns `true` if B1 is less than B2.
- **LE__(B1, B2, B3)**: Returns `true` if B1 is less than or equal to B2.
- **GE__(B1, B2, B3)**: Returns `true` if B1 is greater than or equal to B2.
- **IN__(V, D, B)**: Determines if D is an element of array V and satisfies **TRUE__**.
- **ORFT__(B1, B2, B3)**: Returns true if B1 is false or B2 is true, based on B3.

NOT__(D)

```
function NOT__(D)
{
    return ! D
}
```

function NOT$_{(D)return!D}$

NULL__(D)

```
function NULL__(D)
{
    return NOT__(D)
}
```

function NULL$_{(D)returnNOT_{(D)}}$

FULL__(D)

```
function FULL__(D)
{
    return CMP__(D, "", "", 1)
}
```

function FULL$_{(D)returnCMP_{(D,"","",1)}}$

TRUE__(D, B)

```
function TRUE__(D, B)
{
    if (B)
        return FULL__(D)
    else if (NOT__(NULL__(D)))
        return 1
    return 0
}
```

function TRUE$_{(D,B)if(B)returnFULL_{(D)elseif(NOT_{(NULL_{(D)))return1return0})}}}$

IN__(V, D, B)

```
function IN__(V, D, B)
{
    return D in V && TRUE__(V[D], B)
}
```

function IN$_{(V,D,B)returnDinVTRUE_{(V[D],B)}}$

FALSE__(D, B)

```
function FALSE__(D, B)
{
    return NOT__(TRUE\_\_(D, B))
}
```

function FALSE$_{(D,B)returnNOT_{(TRUE\_(D,B))}}$

OR__(B1, B2, B3)

```
function OR__(B1, B2, B3)
{
        return TRUE__(B1, B3) || TRUE\_\_(B2, B3)
}
```

function OR$_{(B1,B2,B3)returnTRUE_{(B1,B3)||TRUE\_(B2,B3)}}$

NOR__(B1, B2, B3)

```
function NOR__(B1, B2, B3)
{
        return NOT__(OR__(B1, B2, B3))
}
```

function NOR$_{(B1,B2,B3)returnNOT_{(OR_{(B1,B2,B3))}}}$

ORFT__(B1, B2, B3)

```
function ORFT__(B1, B2, B3)
{
        # Return the result of OR__ with the negation of B1 and the truth value
↪of B2
        return OR__(FALSE__(B1, B3), TRUE__(B2, B3))
}
```

function ORFT$_{(B1,B2,B3)ReturntheresultofOR_{withthenegationofB1andthetruthvalueofB2returnOR_{(FALSE_{(B1,B3),TRUE_{(B2,B3))}}}}}$

AND__(B1, B2, B3)

```
function AND\_\_(B1, B2, B3)
{
        return TRUE__(B1, B3) && TRUE__(B2, B3)
}
```

function AND__(B1, B2, B3) return TRUE$_{(B1,B3)TRUE_{(B2,B3)}}$

NAND__(B1, B2, B3)

```
function NAND__(B1, B2, B3)
{
        return NOT__(AND__(B1, B2, B3))
}
```

function NAND$_{(B1,B2,B3)returnNOT_{(AND_{(B1,B2,B3))}}}$

XOR__(B1, B2, B3)

```
function XOR__(B1, B2, B3)
{
	# Return the result of OR__ with the combination of AND__ and AND__
	return OR__(AND__(TRUE__(B1, B3), FALSE__(B2, B3)),
		    AND__(FALSE__(B1, B3), TRUE__(B2, B3)))
}
```

function XOR$_{(B1,B2,B3)ReturntheresultofOR_{withthecombinationofAND_{andAND_{returnOR_{(AND_{(TRUE_{(B1,B3),FALSE_{(B2,B3)),AND_{(FALSE}}}}}}}}}$

XNOR__(B1, B2, B3)

```
function XNOR__(B1, B2, B3)
{
	return NOT__(XOR__(B1, B2, B3))
}
```

function XNOR$_{(B1,B2,B3)returnNOT_{(XOR_{(B1,B2,B3))}}}$

CMP__(B1, B2, B3, B4)

```
function CMP__(B1, B2, B3, B4)
{
	# If B3 is true
	if (B3) {
		if (B4)
			return B1 > B2
		if (length(B4))
			return B1 ~ B2
		return B1 == B2
	# Else if B3 has a length
	} else if (length(B3)) {
		if (B4)
			return length(B1) > length(B2)
		if (length(B4))
			return length(B1) ~ length(B2)
		return length(B1) == length(B2)
	} else if (is_digit(B1, 1) && is_digit(B2, 1)) {
		if (B4)
			return +B1 > +B2
		if (length(B4))
			return +B1 ~ +B2
		return +B1 == +B2
	} else {
		if (B4)
			return "a" B1 > "a" B2
```

```
                if (length(B4))
                        return "a" B1 ~ "a" B2
                return "a" B1 == "a" B2
        }
}
```

function CMP$_{(B1,B2,B3,B4) If B3 is true if (B3) if (B4) return B1 > B2 if (length(B4)) return B1 \ B2 return B1 == B2 Else if B3 has a length el}$

NCMP__(B1, B2, B3, B4)

```
function NCMP__(B1, B2, B3, B4)
{
        return NOT__(CMP__(B1, B2, B3, B4))
}
```

function NCMP$_{(B1,B2,B3,B4) return NOT_{(CMP_{(B1,B2,B3,B4))}}}$

LOR__(B1, B2, B3, M)

```
function LOR__(B1, B2, B3, M,   t)
{
        # Determine mode based on M pattern: length or default
        if (M ~ /^(l(e(n(g(t(h)?)?)?)?)?)?$/) # Regex for 'length'
                t = 0
        else if (M ~ /^(d(e(f(a(u(l(t)?)?)?)?)?)?)$/) # Regex for 'default'
                t = 1
        # Full comparison based on t
        if (FULL__(t)) {
                if (B3)
                        return GT__(B1, B2, t) # Greater than comparison
                else
                        return LT__(B1, B2, t) # Less than comparison
        } else {
                # Check if B1 and B2 are digits or M is 'string'
                if (! (is_digit(B1, 1) && is_digit(B2, 1)) || M ~
↪/^(s(t(r(i(n(g)?)?)?)?)?)$/)
                        t = "a" # Set t to 'a' for ASCII comparison
                if (B3)
                        return GT__(t B1, t B2) # Concatenate t with B1 and B2,
↪compare
                else
                        return LT__(t B1, t B2)
        }
}
```

function LOR$_{(B1,B2,B3,M,t) Determine mode based on M pattern: length or default if (M \ /^{(} l(e(n(g(t(h)?)?)?)?)?)^{/})}$ Regex for 'length' t = 0 else if (M $/^{(} d(e(f(a(u(l(t)?)?)?)?)?)?)/)$ Regex for 'default' t = 1 Full comparison based on t

if (FULL$_{(t))if(B3)returnGT_{(B1,B2,t)GreaterthancomparisonelsereturnLT_{(B1,B2,t)Lessthancomparison}}}^{elseCheckifB1andB2aredigitsorM is' string' if(}$
t = "a" Set t to 'a' for ASCII comparison if (B3) return GT$_{(tB1,tB2)ConcatenatetwithB1andB2,compareelsereturnLT_{(tB1,tB2)}}$

EQ__(B1, B2, B3)

```
function EQ__(B1, B2, B3)
{
        return CMP__(B1, B2, B3)
}
```

function EQ$_{(B1,B2,B3)returnCMP_{(B1,B2,B3)}}$

NEQ__(B1, B2, B3)

```
function NEQ__(B1, B2, B3)
{
        return NCMP__(B1, B2, B3)
}
```

function NEQ$_{(B1,B2,B3)returnNCMP_{(B1,B2,B3)}}$

3 Math

4 Strings

5 Type Validation

Type Validation

The following functions provide utilities to classify and manipulate numeric inputs in various formats. The examples demonstrate how to use these functions in practice.

- **__is_signed(N)**: Checks if the input number N has a + or – prefix.
- **__get_sign(N)**: Retrieves the sign (+ or –) of N if it is signed.
- **is_integral(N, B)**: Verifies if N is an integer. The parameter B specifies whether to allow a sign prefix.
- **is_signed_integral(N)**: Checks if N is a signed integer.
- **is_float(N, B)**: Determines if N is a floating-point number, with B controlling the allowance of a sign.
- **is_signed_float(N)**: Validates whether N is a signed floating-point number.
- **is_digit(N, B)**: Checks if N is any numeric value (integer or float).
- **is_signed_digit(N)**: Checks if N is a signed numeric value (integer or float).

__is_signed(N)

```
function __is_signed(N)
{
        return N ~ /^([-]|[+])/
}
```

function $_{i}s_{s}igned(N)returnN\ /^{(}[-]|[+])/$

__get_sign(N)

```
function __get_sign(N)
{
        if (__is_signed(N)) {
                return substr(N, 1, 1)
        }
}
```

function ${}_{get_sign(N)if(_{is_signed(N))returnsubstr(N,1,1)}}$

is_integral(N)

```
function is_integral(N, B,      e)
{
    if ((B && N ~ /^([-]|[+])?[0-9]+$/) || (! B && N ~ /^[0-9]+$/))
                e = 1
        return e
}
```

function is$_integral(N, B, e)if((BN\ /^([-]|[+])?[0-9]+/)$ || (! B N $/^[0-9]+/)$) e = 1 return e

is_signed_integral(N)

```
function is_signed_integral(N,          e)
{
        if (__is_signed(N) && is_integral(N, 1))
                e = 1
        return e
}
```

function is$_signed_integral(N, e)if(_{is_signed(N)is_integral(N,1))e=1returne}$

is_float(N, B)

```
function is_float(N, B,         e)
{
        if ((B && N ~ /^([-]|[+])?[0-9]+[.][0-9]+$/) || (! B && N ~
↪/^[0-9]+[.][0-9]+$/))
                e = 1
        return e
}
```

function is$_float(N, B, e)if((BN\ /^([-]|[+])?[0-9]+[.][0-9]+/)$ || (! B N $/^[0-9]+[.][0-9]+/)$) e = 1 return e

is_signed_float(N)

```
function is_signed_float(N,              e)
{
        if (__is_signed(N) && is_float(N, 1))
                e = 1
        return e
}
```

function is$_{s}igned_{f}loat(N, e)if(_{is_{s}igned(N)is_{f}loat(N,1))e=1returne}$

is_digit(N, B)

```
function is_digit(N, B,         e)
{
        if (is_integral(N, B) || is_float(N, B))
                e = 1
        return e
}
```

function is$_{d}igit(N, B, e)if(is_{i}ntegral(N, B)||is_{f}loat(N, B))e = 1returne$

is_signed_digit(N)

```
function is_signed_digit(N,     e)
{
        if (__is_signed(N) && is_digit(N, 1))
                e = 1
        return e
}
```

function is$_{s}igned_{d}igit(N, e)if(_{is_{s}igned(N)is_{d}igit(N,1))e=1returne}$

6 Numbers Base

7 Internet

8 Miscellaneous

9 Standard Output

10 Structures