Modul:Set

Vorlage:Lua

This module includes a number of set operations for Lua tables. It currently has union, intersection and complement functions for both key/value pairs and for values only. It is a meta-module, meant to be called from other Lua modules, and should not be called directly from #invoke.

Inhaltsverzeichnis

1 Loading the module 1
2 union 1
3 valueUnion 1
4 intersection 2
5 valueIntersection 2
6 complement 2
7 valueComplement 2

Loading the module

To use any of the functions, first you must load the module.

```
local set = require('Module:Set')
```

union

```
set.union(t1, t2, ...)
```

Returns the union of the key/value pairs of n tables. If any of the tables contain different values for the same table key, the table value is converted to an array holding all of the different values. For example, for the tables `{foo = "foo", bar = "bar"}` and `{foo = "foo", bar = "baz", qu` , union will return `{foo = "foo", bar = {"bar", "baz"}, qux = "qux"}`. An error is raised if the function receives less than two tables as arguments.

valueUnion

```
set.valueUnion(t1, t2, ...)
```

Returns the union of the values of n tables, as an array. For example, for the tables `{1, 3, 4, 5,` and `{2, bar = 3, 5, 6}`, valueUnion will return `{1, 2, 3, 4, 5, 6, 7}`. An error is raised if the function receives less than two tables as arguments.

intersection

```
set.intersection(t1, t2, ...)
```

Returns the intersection of the key/value pairs of n tables. Both the key and the value must match to be included in the resulting table. For example, for the tables `{foo = "foo", bar = "ba` and `{foo = "foo", bar = "baz", qux = "qux"}`, intersection will return `{foo = "foo"}`. An error is raised if the function receives less than two tables as arguments.

valueIntersection

```
set.valueIntersection(t1, t2, ...)
```

Returns the intersection of the values of n tables, as an array. For example, for the tables `{1, 3,` and `{2, bar = 3, 5, 6}`, valueIntersection will return `{3, 5}`. An error is raised if the function receives less than two tables as arguments.

complement

```
set.complement(t1, t2, ..., tn)
```

Returns the relative complement of *t1*, *t2*, ..., in *tn*. The complement is of key/value pairs. This is equivalent to all the key/value pairs that are in *tn* but are not in any of *t1*, *t2*, ... *tn-1*. For example, for the tables `{foo = "foo", bar = "bar", baz = "baz"}` and `{foo = "foo", bar =` , complement would return `{bar = "baz", qux = "qux"}`. An error is raised if the function receives less than two tables as arguments.

valueComplement

```
set.valueComplement(t1, t2, ..., tn)
```

This returns an array containing the relative complement of *t1*, *t2*, ..., in *tn*. The complement is of values only. This is equivalent to all the values that are in tn but are not in t1, t2, ... tn-1. For example, for the tables `{1, 2}`, `{1, 2, 3}` and `{1, 2, 3, 4, 5}`, valueComplement would return `{4, 5}`. An error is raised if the function receives less than two tables as arguments.

```
--[[
--------------------------------------------------------------------------------
--                                    Set
--
-- This module includes a number of set operations for dealing with Lua tables.
-- It currently has union, intersection and complement functions for both
-- key/value pairs and for values only.
--------------------------------------------------------------------------------
--]]
```

```
-- Get necessary libraries and functions
local libraryUtil = require('libraryUtil')
local checkType = libraryUtil.checkType
local tableTools = require('Module:TableTools')

local p = {}

--[[
--------------------------------------------------------------------------------
-- Helper functions
--------------------------------------------------------------------------------
--]]

-- Makes a set from a table's values. Returns an array of all values with
-- duplicates removed.
local function makeValueSet(t)
	local isNan = tableTools.isNan
	local ret, exists = {}, {}
	for k, v in pairs(t) do
		if isNan(v) then
			-- NaNs are always unique, and they can't be table keys,
			-- check for existence.
			ret[#ret + 1] = v
		elseif not exists[v] then
			exists[v] = true
			ret[#ret + 1] = v
		end
	end
	return ret
end

--[[
--------------------------------------------------------------------------------
-- union
--
-- This returns the union of the key/value pairs of n tables. If any of the table
-- contain different values for the same table key, the table value is converted
-- to an array holding all of the different values.
--------------------------------------------------------------------------------
--]]
function p.union(...)
	local lim = select('#', ...)
	if lim < 2 then
		error("too few arguments to 'union' (minimum is 2, received " ..
	end
	local ret, trackArrays = {}, {}
	for i = 1, lim do
		local t = select(i, ...)
		checkType('union', i, t, 'table')
		for k, v in pairs(t) do
			local retKey = ret[k]
			if retKey == nil then
				ret[k] = v
			elseif retKey ~= v then
				if trackArrays[k] then
					local array = ret[k]
					local valExists
					for _, arrayVal in ipairs(array) do
						if arrayVal == v then
							valExists = true
							break
						end
					end
```

```
                                        if not valExists then
                                                array[#array + 1] = v
                                                ret[k] = array
                                        end
                                else
                                        ret[k] = {ret[k], v}
                                        trackArrays[k] = true
                                end
                        end
                end
        end
        return ret
end

--[[
--------------------------------------------------------------------------------
-- valueUnion
--
-- This returns the union of the values of n tables, as an array. For example, fo
-- the tables {1, 3, 4, 5, foo = 7} and {2, bar = 3, 5, 6}, union will return
-- {1, 2, 3, 4, 5, 6, 7}.
--------------------------------------------------------------------------------
--]]
function p.valueUnion(...)
        local lim = select('#', ...)
        if lim < 2 then
                error("too few arguments to 'valueUnion' (minimum is 2, received
        end
        local isNan = tableTools.isNan
        local ret, exists = {}, {}
        for i = 1, lim do
                local t = select(i, ...)
                checkType('valueUnion', i, t, 'table')
                for k, v in pairs(t) do
                        if isNan(v) then
                                ret[#ret + 1] = v
                        elseif not exists[v] then
                                ret[#ret + 1] = v
                                exists[v] = true
                        end
                end
        end
        return ret
end

--[[
--------------------------------------------------------------------------------
-- intersection
--
-- This returns the intersection of the key/value pairs of n tables. Both the key
-- and the value must match to be included in the resulting table.
--------------------------------------------------------------------------------
--]]
function p.intersection(...)
        local lim = select('#', ...)
        if lim < 2 then
                error("too few arguments to 'intersection' (minimum is 2, receive
        end
        local ret, track, pairCounts = {}, {}, {}
        for i = 1, lim do
                local t = select(i, ...)
                checkType('intersection', i, t, 'table')
                for k, v in pairs(t) do
                        local trackVal = track[k]
```

```
                        if trackVal == nil then
                                track[k] = v
                                pairCounts[k] = 1
                        elseif trackVal == v then
                                pairCounts[k] = pairCounts[k] + 1
                        end
                end
        end
        for k, v in pairs(track) do
                if pairCounts[k] == lim then
                        ret[k] = v
                end
        end
        return ret
end

--[[
--------------------------------------------------------------------------------
-- valueIntersection
--
-- This returns the intersection of the values of n tables, as an array. For
-- example, for the tables {1, 3, 4, 5, foo = 7} and {2, bar = 3, 5, 6},
-- intersection will return {3, 5}.
--------------------------------------------------------------------------------
--]]

function p.valueIntersection(...)
        local lim = select('#', ...)
        if lim < 2 then
                error("too few arguments to 'valueIntersection' (minimum is 2, re
        end
        local isNan = tableTools.isNan
        local vals, ret = {}, {}
        local isSameTable = true -- Tracks table equality.
        local tableTemp -- Used to store the table from the previous loop so that
        for i = 1, lim do
                local t = select(i, ...)
                checkType('valueIntersection', i, t, 'table')
                if tableTemp and t ~= tableTemp then
                        isSameTable = false
                end
                tableTemp = t
                t = makeValueSet(t) -- Remove duplicates
                for k, v in pairs(t) do
                        -- NaNs are never equal to any other value, so they can't
                        -- Which is lucky, as they also can't be table keys.
                        if not isNan(v) then
                                local valCount = vals[v] or 0
                                vals[v] = valCount + 1
                        end
                end
        end
        if isSameTable then
                -- If all the tables are equal, then the intersection is that tab
                -- All we need to do is convert it to an array and remove duplica
                return makeValueSet(tableTemp)
        end
        for val, count in pairs(vals) do
                if count == lim then
                        ret[#ret + 1] = val
                end
        end
        return ret
end
```

```
--[[
--------------------------------------------------------------------------------
-- complement
--
-- This returns the relative complement of t1, t2, ..., in tn. The complement
-- is of key/value pairs. This is equivalent to all the key/value pairs that are
-- tn but are not in t1, t2, ... tn-1.
--------------------------------------------------------------------------------
--]]
function p.complement(...)
	local lim = select('#', ...)
	if lim < 2 then
		error("too few arguments to 'complement' (minimum is 2, received
	end
	--[[
	-- Now we know that we have at least two sets.
	-- First, get all the key/value pairs in tn. We can't simply make ret equ
	-- as that will affect the value of tn for the whole module.
	--]]
	local tn = select(lim, ...)
	checkType('complement', lim, tn, 'table')
	local ret = tableTools.shallowClone(tn)
	-- Remove all the key/value pairs in t1, t2, ..., tn-1.
	for i = 1, lim - 1 do
		local t = select(i, ...)
		checkType('complement', i, t, 'table')
		for k, v in pairs(t) do
			if ret[k] == v then
				ret[k] = nil
			end
		end
	end
	return ret
end

--[[
--------------------------------------------------------------------------------
-- valueComplement
--
-- This returns an array containing the relative complement of t1, t2, ..., in tn
-- The complement is of values only. This is equivalent to all the values that ar
-- in tn but are not in t1, t2, ... tn-1.
--------------------------------------------------------------------------------
--]]
function p.valueComplement(...)
	local lim = select('#', ...)
	if lim < 2 then
		error("too few arguments to 'valueComplement' (minimum is 2, rece
	end
	local isNan = tableTools.isNan
	local ret, exists = {}, {}
	for i = 1, lim - 1 do
		local t = select(i, ...)
		checkType('valueComplement', i, t, 'table')
		t = makeValueSet(t) -- Remove duplicates
		for k, v in pairs(t) do
			if not isNan(v) then
				-- NaNs cannot be table keys, and they are also u
				exists[v] = true
			end
		end
	end
	local tn = select(lim, ...)
```

```
        checkType('valueComplement', lim, tn, 'table')
        tn = makeValueSet(tn) -- Remove duplicates
        for k, v in pairs(tn) do
                if isNan(v) or exists[v] == nil then
                        ret[#ret + 1] = v
                end
        end
        return ret
end

--[[
--------------------------------------------------------------------------------
-- symmDiff
--
-- This returns the symmetric difference of key/value pairs of t1, t2, ..., tn.
-- The symmetric difference of two tables consists of the key/value pairs
-- that appear in set 1 but not set 2, together with the key/value pairs that
-- appear in set 2 but not in set 1. This is the same as the union of the two
-- minus the intersection. If either of the tables contain different values for t
-- same table key, the table value is converted to an array holding all of the
-- different values.For more than two tables, this can get confusing - see the
-- "Symmetric difference" article for details.
--------------------------------------------------------------------------------
--]]

--[[ -- This is a rough work in progress.
function p.symmDiff(...)
        local lim = select('#', ...)
        if lim < 2 then
                error("too few arguments to 'symmDiff' (minimum is 2, received "
        end

        local tremove = table.remove
        local trackArrays = {}

        local function symmDiffTwo(t1, t2)
                local ret = {}
                for k, v in pairs(t1) do
                        local t2val = t2[k]
                        if t2val == nil then
                                ret[k] = v
                        elseif trackArrays[k] then
                                local array = ret[k]
                                local valExists
                                for i, arrayVal in ipairs(array) do
                                        if arrayVal == v then
                                                valExists = true
                                                break
                                        end
                                end
                                if not valExists then
                                        array[#array + 1] = v
                                end
                        elseif v ~= t2val then
                                ret[k] = {t2val, v}
                                trackArrays[k] = true
                        end
--]]

return p
```