```
'use strict'

function isArguments (thingy) {
  return thingy != null && typeof thingy === 'object' && thingy.hasOwnProperty('callee')
}

var types = {
  '*': {label: 'any', check: function () { return true }},
  A: {label: 'array', check: function (thingy) { return Array.isArray(thingy) ||
isArguments(thingy) }},
  S: {label: 'string', check: function (thingy) { return typeof thingy === 'string' }},
  N: {label: 'number', check: function (thingy) { return typeof thingy === 'number' }},
  F: {label: 'function', check: function (thingy) { return typeof thingy === 'function' }},
  O: {label: 'object', check: function (thingy) { return typeof thingy === 'object' &&
thingy != null && !types.A.check(thingy) && !types.E.check(thingy) }},
  B: {label: 'boolean', check: function (thingy) { return typeof thingy === 'boolean' }},
  E: {label: 'error', check: function (thingy) { return thingy instanceof Error }},
  Z: {label: 'null', check: function (thingy) { return thingy == null }}
}

function addSchema (schema, arity) {
  var group = arity[schema.length] = arity[schema.length] || []
  if (group.indexOf(schema) === -1) group.push(schema)
}

var validate = module.exports = function (rawSchemas, args) {
  if (arguments.length !== 2) throw wrongNumberOfArgs(['SA'], arguments.length)
  if (!rawSchemas) throw missingRequiredArg(0, 'rawSchemas')
  if (!args) throw missingRequiredArg(1, 'args')
  if (!types.S.check(rawSchemas)) throw invalidType(0, ['string'], rawSchemas)
  if (!types.A.check(args)) throw invalidType(1, ['array'], args)
  var schemas = rawSchemas.split('|')
  var arity = {}

  schemas.forEach(function (schema) {
    for (var ii = 0; ii < schema.length; ++ii) {
      var type = schema[ii]
      if (!types[type]) throw unknownType(ii, type)
    }
    if (/E.*E/.test(schema)) throw moreThanOneError(schema)
    addSchema(schema, arity)
    if (/E/.test(schema)) {
      addSchema(schema.replace(/E.*$/, 'E'), arity)
      addSchema(schema.replace(/E/, 'Z'), arity)
      if (schema.length === 1) addSchema('', arity)
    }
  })
  var matching = arity[args.length]
  if (!matching) {
    throw wrongNumberOfArgs(Object.keys(arity), args.length)
  }
  for (var ii = 0; ii < args.length; ++ii) {
    var newMatching = matching.filter(function (schema) {
      var type = schema[ii]
      var typeCheck = types[type].check
      return typeCheck(args[ii])
    })
    if (!newMatching.length) {
      var labels = matching.map(function (schema) {
        return types[schema[ii]].label
      }).filter(function (schema) { return schema != null })
      throw invalidType(ii, labels, args[ii])
    }
    matching = newMatching
```

```
  }
}

function missingRequiredArg (num) {
  return newException('EMISSINGARG', 'Missing required argument #' + (num + 1))
}

function unknownType (num, type) {
  return newException('EUNKNOWNTYPE', 'Unknown type ' + type + ' in argument #' + (num + 1))
}

function invalidType (num, expectedTypes, value) {
  var valueType
  Object.keys(types).forEach(function (typeCode) {
    if (types[typeCode].check(value)) valueType = types[typeCode].label
  })
  return newException('EINVALIDTYPE', 'Argument #' + (num + 1) + ': Expected ' +
    englishList(expectedTypes) + ' but got ' + valueType)
}

function englishList (list) {
  return list.join(', ').replace(/, ([^,]+)$/, ' or $1')
}

function wrongNumberOfArgs (expected, got) {
  var english = englishList(expected)
  var args = expected.every(function (ex) { return ex.length === 1 })
    ? 'argument'
    : 'arguments'
  return newException('EWRONGARGCOUNT', 'Expected ' + english + ' ' + args + ' but got ' +
got)
}

function moreThanOneError (schema) {
  return newException('ETOOMANYERRORTYPES',
    'Only one error type per argument signature is allowed, more than one found in "' +
schema + '"')
}

function newException (code, msg) {
  var e = new Error(msg)
  e.code = code
  if (Error.captureStackTrace) Error.captureStackTrace(e, validate)
  return e
}
```