```
# Tools to analyze parts of tensor expressions and extract information

# analyze tensor expressions and extract information
function decomposetensor(ex::Expr)
    istensor(ex) || throw(ArgumentError("not a valid tensor: $ex"))
    if ex.head == :ref || ex.head == :typed_hcat
        if length(ex.args) == 1
            return ex.args[1], Any[], Any[]
        elseif isa(ex.args[2], Expr) && ex.args[2].head == :parameters
            return ex.args[1], ex.args[3:end], ex.args[2].args
        else
            return ex.args[1], ex.args[2:end], Any[]
        end
    else #if ex.head == :typed_vcat
        if isa(ex.args[2], Expr) && (ex.args[2].head == :row || ex.args[2].head ==
:tuple)
            leftind = ex.args[2].args
        else
            leftind = ex.args[2:2]
        end
        if isa(ex.args[3], Expr) && (ex.args[3].head == :row || ex.args[3].head ==
:tuple)
            rightind = ex.args[3].args
        else
            rightind = ex.args[3:3]
        end
        return ex.args[1], leftind, rightind
    end
end

gettensorobject(ex) = decomposetensor(ex)[1]
getleftindices(ex) = decomposetensor(ex)[2]
getrightindices(ex) = decomposetensor(ex)[3]

function decomposegeneraltensor(ex)
    if istensor(ex)
        object, leftind, rightind = decomposetensor(ex)
        return (object, leftind, rightind, true, false)
    elseif isa(ex, Expr) && ex.head == :call && ex.args[1] == :+ && length(ex.args)
        == 2 # unary plus: pass on
        return decomposegeneraltensor(ex.args[2])
    elseif isa(ex, Expr) && ex.head == :call && ex.args[1] == :- && length(ex.args)
        == 2 # unary minus: flip scalar factor
        (object, leftind, rightind, α, conj) = decomposegeneraltensor(ex.args[2])
        return (object, leftind, rightind, Expr(:call, :-, α), conj)
    elseif isa(ex, Expr) && ex.head == :call && ex.args[1] == :conj &&
        length(ex.args) == 2 # conjugation: flip conjugation flag and conjugate
        scalar factor
        (object, leftind, rightind, α, conj) = decomposegeneraltensor(ex.args[2])
        return (object, leftind, rightind, Expr(:call, :conj, α), !conj)
    elseif ex.head == :call && ex.args[1] == :* && length(ex.args) == 3 # scalar
        multiplication: muliply scalar factors
        if isscalarexpr(ex.args[2]) && isgeneraltensor(ex.args[3])
            (object, leftind, rightind, α, conj) =
```

```
decomposegeneraltensor(ex.args[3])
            return (object, leftind, rightind, Expr(:call, :*, ex.args[2], α), conj)
        elseif isscalarexpr(ex.args[3]) && isgeneraltensor(ex.args[2])
            (object, leftind, rightind, α, conj) =
decomposegeneraltensor(ex.args[2])
            return (object, leftind, rightind, Expr(:call, :*, α, ex.args[3]), conj)
        end
    elseif ex.head == :call && ex.args[1] == :/ && length(ex.args) == 3 # scalar
        multiplication: muliply scalar factors
        if isscalarexpr(ex.args[3]) && isgeneraltensor(ex.args[2])
            (object, leftind, rightind, α, conj) =
decomposegeneraltensor(ex.args[2])
            return (object, leftind, rightind, Expr(:call, :/, α, ex.args[3]), conj)
        end
    elseif ex.head == :call && ex.args[1] == :\ && length(ex.args) == 3 # scalar
        multiplication: muliply scalar factors
        if isscalarexpr(ex.args[2]) && isgeneraltensor(ex.args[3])
            (object, leftind, rightind, α, conj) =
decomposegeneraltensor(ex.args[3])
            return (object, leftind, rightind, Expr(:call, :\, ex.args[2], α), conj)
        end
    end
    throw(ArgumentError("not a valid generalized tensor expression $ex"))
end

# get left and right hand side from an expression or definition
function getlhs(ex::Expr)
    if ex.head in (:(=), :(+=), :(-=), :(:=), :(≔))
        return ex.args[1]
    else
        throw(ArgumentError("invalid assignment or definition $ex"))
    end
end
function getrhs(ex::Expr)
    if ex.head in (:(=), :(+=), :(-=), :(:=), :(≔))
        return ex.args[2]
    else
        throw(ArgumentError("invalid assignment or definition $ex"))
    end
end

# get all the tensor objects in a tensor expression (not a definition or assignment)
function gettensorobjects(ex)
    if istensor(ex)
        Any[gettensorobject(ex)]
    elseif istensorexpr(ex)
        list = Any[]
        for e in ex.args
            append!(list, gettensorobjects(e))
        end
        return list
    else
        return Any[]
    end
end
```

```
# get all the existing tensor objects which are inputs (i.e. appear in the rhs of
  assignments and definitions)
function getinputtensorobjects(ex)
    list = Any[]
    if isdefinition(ex)
        append!(list, gettensorobjects(getrhs(ex)))
    elseif isassignment(ex)
        if ex.head == :(+=) || ex.head == :(-=)
            lhs = getlhs(ex)
            if istensor(lhs)
                push!(list, gettensorobject(lhs))
            end
        end
        append!(list, gettensorobjects(getrhs(ex)))
    elseif isa(ex, Expr) && ex.head == :block
        for i = 1:length(ex.args)
            list2 = getinputtensorobjects(ex.args[i])
            for j = 1:i-1
                # if objects have previously been defined or assigned to, they are
                  not inputs
                list2 = setdiff(list2, getnewtensorobjects(ex.args[j]))
                list2 = setdiff(list2, getoutputtensorobjects(ex.args[j]))
            end
            append!(list, list2)
        end
    elseif isa(ex, Expr) && ex.head in (:for, :while)
        append!(list, getinputtensorobjects(ex.args[2]))
    elseif isa(ex, Expr) && ex.head == :call && ex.args[1] == :scalar
        append!(list, gettensorobjects(ex.args[2]))
    end
    return unique!(list)
end
# get all the existing tensor objects which are outputs (i.e. appear in the lhs of
  assignments)
function getoutputtensorobjects(ex)
    list = Any[]
    if isassignment(ex)
        lhs = getlhs(ex)
        if istensor(lhs)
            push!(list, gettensorobject(lhs))
        end
    elseif isa(ex, Expr) && ex.head == :block
        for i = 1:length(ex.args)
            list2 = getoutputtensorobjects(ex.args[i])
            for j = 1:i-1
                # if objects have previously been defined, they are not existing
                  outputs
                list2 = setdiff(list2, getnewtensorobjects(ex.args[j]))
            end
            append!(list, list2)
        end
    elseif isa(ex, Expr) && ex.head in (:for, :while)
        append!(list, getoutputtensorobjects(ex.args[2]))
    end
```

```
    return unique!(list)
end
# get all the existing tensor objects which are newly created (i.e. appear in the
  lhs of definition)
function getnewtensorobjects(ex)
    list = Any[]
    if isdefinition(ex)
        lhs = getlhs(ex)
        if istensor(lhs)
            push!(list, gettensorobject(lhs))
        end
    elseif isa(ex, Expr) && ex.head == :block
        for e in ex.args
            append!(list, getnewtensorobjects(e))
        end
    elseif isa(ex, Expr) && ex.head in (:for, :while)
        append!(list, getnewtensorobjects(ex.args[2]))
    end
    return unique!(list)
end

# for any tensor expression, get the list of uncontracted indices that would remain
  after evaluating that expression
function getindices(ex::Expr)
    if istensor(ex)
        _,leftind,rightind = decomposetensor(ex)
        return unique2(vcat(leftind, rightind))
    elseif ex.head == :call && (ex.args[1] == :+ || ex.args[1] == :-)
        return getindices(ex.args[2]) # getindices on any of the args[2:end] should
            yield the same result
    elseif ex.head == :call && ex.args[1] == :*
        indices = getindices(ex.args[2])
        for k = 3:length(ex.args)
            append!(indices, getindices(ex.args[k]))
        end
        return unique2(indices)
    elseif ex.head == :call && ex.args[1] == :/
        return getindices(ex.args[2])
    elseif ex.head == :call && ex.args[1] == :\
        return getindices(ex.args[3])
    elseif ex.head == :call && length(ex.args) == 2
        return getindices(ex.args[2])
    else
        return Any[]
    end
end
getindices(ex) = Any[]

# get all unique indices appearing in a tensor expression
function getallindices(ex::Expr)
    if istensor(ex)
        _,leftind,rightind = decomposetensor(ex)
        return unique!(vcat(leftind, rightind))
    elseif isassignment(ex) || isdefinition(ex)
        return getallindices(getrhs(ex))
```

```
    else
        return unique!(mapreduce(getallindices, vcat, ex.args))
    # elseif istensorexpr(ex)
    #     return unique!(mapreduce(getallindices, vcat, ex.args))
    # else
    #     return Any[]
    end
end
getallindices(ex) = Any[]

# # auxiliary routine
# function unique2(itr)
#     out = collect(itr)
#     i = 1
#     while i < length(out)
#         inext = findnext(isequal(out[i]), out, i+1)
#         if inext == nothing
#             i += 1
#             continue
#         end
#         while inext !== nothing
#             deleteat!(out, inext)
#             inext = findnext(isequal(out[i]), out, i+1)
#         end
#         deleteat!(out, i)
#     end
#     out
# end
```