```
# replace all indices by a function of that index
function replaceindices((@nospecialize f), ex::Expr)
    if istensor(ex)
        if ex.head == :ref || ex.head == :typed_hcat
            if length(ex.args) == 1
                return ex
            elseif isa(ex.args[2], Expr) && ex.args[2].head == :parameters
                arg2 = ex.args[2]
                return Expr(ex.head, ex.args[1],
                            Expr(arg2.head, map(f, arg2.args)...),
                            (f(ex.args[i]) for i = 3:length(ex.args))...)
            else
                return Expr(ex.head, ex.args[1],
                            (f(ex.args[i]) for i = 2:length(ex.args))...)
            end
            return ex
        else #if ex.head == :typed_vcat
            arg2, arg3 = map((ex.args[2], ex.args[3])) do arg
                if isa(arg, Expr) && (arg.head == :row || arg.head == :tuple)
                    return Expr(arg.head, map(f, arg.args)...)
                else
                    return f(arg)
                end
            end
            return Expr(ex.head, ex.args[1], arg2, arg3)
        end
    else
        return Expr(ex.head, (replaceindices(f, e) for e in ex.args)...)
    end
end
replaceindices((@nospecialize f), ex) = ex

function normalizeindex(ex)
    if isa(ex, Symbol) || isa(ex, Int)
        return ex
    elseif isa(ex, Expr) && ex.head == prime && length(ex.args) == 1
        return Symbol(normalizeindex(ex.args[1]), "′")
    else
        error("not a valid index: $ex")
    end
end

normalizeindices(ex::Expr) = replaceindices(normalizeindex, ex)

# replace all tensor objects by a function of that object
function replacetensorobjects(f, ex::Expr)
    # first try to replace ex completely
    ex2 = f(ex, nothing, nothing)
    ex2 !== ex && return ex2
    if istensor(ex)
        obj, leftind, rightind = decomposetensor(ex)
        return Expr(ex.head, f(obj, leftind, rightind), ex.args[2:end]...)
    else
        return Expr(ex.head, (replacetensorobjects(f, e) for e in ex.args)...)
```

```
    end
end
replacetensorobjects(f, ex) = f(ex, nothing, nothing)

# expandconj: conjugate individual terms or factors instead of a whole expression
function expandconj(ex::Expr)
    if isgeneraltensor(ex) || isscalarexpr(ex)
        return ex
    elseif ex.head == :call && ex.args[1] == :conj
        @assert length(ex.args) == 2
        return conjexpr(expandconj(ex.args[2]))
    else
        return Expr(ex.head, map(expandconj, ex.args)...)
    end
end
expandconj(ex) = ex

function conjexpr(ex::Expr)
    if ex.head == :call && ex.args[1] == :conj
        return ex.args[2]
    elseif isgeneraltensor(ex) || isscalarexpr(ex)
        return Expr(:call, :conj, ex)
    elseif ex.head == :call && (ex.args[1] == :* || ex.args[1] == :+ || ex.args[1]
== :-)
        return Expr(ex.head, ex.args[1], map(conjexpr, ex.args[2:end])...)
    elseif ex.head == :call && (ex.args[1] == :/ || ex.args[1] == :\)
        return Expr(ex.head, ex.args[1], map(conjexpr, ex.args[2:end])...)
    else
        error("cannot conjugate expression: $ex")
    end
end
conjexpr(ex::Number) = conj(ex)
conjexpr(ex::Symbol) = Expr(:call, :conj, ex)
conjexpr(ex) = ex

# explicitscalar: wrap all tensor expressions with zero output indices in scalar
  call
function explicitscalar(ex::Expr)
    ex = Expr(ex.head, map(explicitscalar, ex.args)...)
    if istensorexpr(ex) && isempty(getindices(ex))
        return Expr(:call, :scalar, ex)
    else
        return ex
    end
end
explicitscalar(ex) = ex

# extracttensorobjects: replace all tensor objects with newly generated symbols,
  and assign
# them before the expression and after the expression as necessary
function extracttensorobjects(ex::Expr)
    inputtensors = getinputtensorobjects(ex)
    outputtensors = getoutputtensorobjects(ex)
    newtensors = getnewtensorobjects(ex)
    existingtensors = unique!(vcat(inputtensors, outputtensors))
```

```
    alltensors = unique!(vcat(existingtensors, newtensors))
    tensordict = Dict{Any,Any}(a => gensym() for a in alltensors)
    pre = Expr(:block, [Expr(:(=), tensordict[a], a) for a in existingtensors]...)
    ex = replacetensorobjects((obj,leftind,rightind)->get(tensordict, obj, obj), ex)
    post = Expr(:block, [Expr(:(=), a, tensordict[a]) for a in newtensors]...)
    pre2 = Expr(:macrocall, Symbol("@notensor"), LineNumberNode(@__LINE__,
Symbol(@__FILE__)), pre)
    post2 = Expr(:macrocall, Symbol("@notensor"), LineNumberNode(@__LINE__,
Symbol(@__FILE__)), post)
    return Expr(:block, pre2, ex, post2)
end
```