```
function instantiate_eltype(ex::Expr)
    if istensor(ex)
        obj,_,_ = decomposetensor(ex)
        return Expr(:call, :eltype, obj)
    elseif ex.head == :call && (ex.args[1] == :+ || ex.args[1] == :- || ex.args[1]
== :* || ex.args[1] == :/)
        if length(ex.args) > 2
            return Expr(:call, :promote_type, map(instantiate_eltype,
ex.args[2:end])...)
        else
            return instantiate_eltype(ex.args[2])
        end
    elseif ex.head == :call && ex.args[1] == :conj
        return instantiate_eltype(ex.args[2])
    elseif isscalarexpr(ex)
        return :(typeof($ex))
    else
        # return :(eltype($ex)) # would probably lead to doing the same operation
            twice
        throw(ArgumentError("unable to determine eltype"))
    end
end
instantiate_eltype(ex) = Expr(:call,:typeof, ex)

function instantiate_scalar(ex::Expr)
    if ex.head == :call && ex.args[1] == :scalar
        @assert length(ex.args) == 2 && istensorexpr(ex.args[2])
        return :(scalar($(instantiate(nothing, 0, ex.args[2], 1, [], [], true))))
    elseif ex.head == :call
        return Expr(ex.head, ex.args[1], map(instantiate_scalar, ex.args[2:end])...)
    else
        return Expr(ex.head, map(instantiate_scalar, ex.args)...)
    end
end
instantiate_scalar(ex::Symbol) = ex
instantiate_scalar(ex) = ex

function instantiate(dst, β, ex::Expr, α, leftind::Vector{Any},
rightind::Vector{Any}, istemporary = false)
    if isgeneraltensor(ex)
        return instantiate_generaltensor(dst, β, ex, α, leftind, rightind,
istemporary)
    elseif ex.head == :call && (ex.args[1] == :+ || ex.args[1] == :-) # linear
        combination
        return instantiate_linearcombination(dst, β, ex, α, leftind, rightind,
istemporary)
    elseif ex.head == :call && ex.args[1] == :* && length(ex.args) == 3 #
        multiplication: should be pairwise by now
        if isscalarexpr(ex.args[2])
            return instantiate(dst, β, ex.args[3], Expr(:call, :*,
instantiate_scalar(ex.args[2]), α), leftind, rightind, istemporary)
        elseif isscalarexpr(ex.args[3])
            return instantiate(dst, β, ex.args[2], Expr(:call, :*, α,
```

```
instantiate_scalar(ex.args[3])), leftind, rightind, istemporary)
        else
            return instantiate_contraction(dst, β, ex, α, leftind, rightind,
istemporary)
        end
    elseif ex.head == :call && ex.args[1] == :/ && length(ex.args) == 3
        return instantiate(dst, β, ex.args[2], Expr(:call, :/, α,
instantiate_scalar(ex.args[3])), leftind, rightind, istemporary)
    elseif ex.head == :call && ex.args[1] == :\ && length(ex.args) == 3
        return instantiate(dst, β, ex.args[3], Expr(:call, :\,
instantiate_scalar(ex.args[2]), α), leftind, rightind, istemporary)
    end
    throw(ArgumentError("problem with parsing $ex"))
end

function instantiate_generaltensor(dst, β, ex::Expr, α, leftind::Vector{Any},
rightind::Vector{Any}, istemporary = false)
    src, srcleftind, srcrightind, α2, conj = decomposegeneraltensor(ex)
    srcind = vcat(srcleftind, srcrightind)
    conjarg = conj ? :(:C) : :(:N)

    p1 = (map(l->findfirst(isequal(l), srcind), leftind)...,)
    p2 = (map(l->findfirst(isequal(l), srcind), rightind)...,)

    αsym = gensym()
    if dst === nothing
        dst = gensym()
        if istemporary
            initex = quote
                $αsym = $α*$α2
                $dst = cached_similar_from_indices($(QuoteNode(dst)),
promote_type(eltype($src), typeof($αsym)), $p1, $p2, $src, $conjarg)
            end
        else
            initex = quote
                $αsym = $α*$α2
                $dst =
similar_from_indices(promote_type(eltype($src),typeof($αsym)), $p1, $p2, $src,
$conjarg)
            end
        end
    else
        initex = :($αsym = $α*$α2)
    end

    if hastraceindices(ex)
        traceind = unique(setdiff(setdiff(srcind,leftind), rightind))
        q1 = (map(l->findfirst(isequal(l), srcind), traceind)...,)
        q2 = (map(l->findlast(isequal(l), srcind), traceind)...,)
        if any(x->(x===nothing), (p1...,p2...,q1...,q2...)) ||
            !isperm((p1...,p2...,q1...,q2...)) ||
            length(srcind) != length(leftind) + length(rightind) +
2*length(traceind)
            err = "trace: $(tuple(srcleftind..., srcrightind...)) to
$(tuple(leftind..., rightind...))"
```

```
            return :(throw(IndexError($err)))
        end
        return quote
            $initex
            trace!($α * $α2, $src, $conjarg, $β, $dst, $p1, $p2, $q1, $q2)
            # $dst
        end
    else
        if any(x->(x===nothing), (p1..., p2...)) || !isperm((p1...,p2...)) ||
            length(srcind) != length(leftind) + length(rightind)
            err = "add: $(tuple(srcleftind..., srcrightind...)) to
$(tuple(leftind..., rightind...)))"
            return :(throw(IndexError($err)))
        end
        return quote
            $initex
            add!($α * $α2, $src, $conjarg, $β, $dst, $p1, $p2)
            # $dst
        end
    end
end
function instantiate_linearcombination(dst, β, ex::Expr, α, leftind::Vector{Any},
rightind::Vector{Any}, istemporary = false)
    if ex.head == :call && (ex.args[1] == :+ || ex.args[1] == :-) # addition: add
        one by one
        if dst === nothing
            αnew = Expr(:call, :*, α, Expr(:call, :one, instantiate_eltype(ex)))
            ex1 = instantiate(dst, β, ex.args[2], αnew, leftind, rightind,
istemporary)
            dst = gensym()
            returnex = :($dst = $ex1)
        else
            returnex = instantiate(dst, β, ex.args[2], α, leftind, rightind,
istemporary)
        end
        αnew = (ex.args[1] == :-) ? Expr(:call, :-, α) : α
        for k = 3:length(ex.args)
            ex1 = instantiate(dst, true, ex.args[k], αnew, leftind, rightind)
            returnex = quote
                $returnex
                $ex1
            end
        end
        return quote
            $returnex
            # $dst
        end
    else
        throw(ArgumentError("unable to instantiate linear combination: $ex"))
    end
end
function instantiate_contraction(dst, β, ex::Expr, α, leftind::Vector{Any},
rightind::Vector{Any}, istemporary = false)
    @assert ex.head == :call && ex.args[1] == :* && length(ex.args) == 3 &&
        istensorexpr(ex.args[2]) && istensorexpr(ex.args[3])
```

```
    exA = ex.args[2]
    exB = ex.args[3]

    indA = getindices(exA)
    indB = getindices(exB)
    cind = intersect(indA, indB)
    indC = vcat(leftind, rightind)
    oindA = intersect(indA, indC) # in the order they appear in A
    oindB = intersect(indB, indC) # in the order they appear in B

    symA = gensym()
    symB = gensym()
    symC = gensym()
    symTC = gensym()

    # prepare tensors or tensor expressions
    if dst === nothing
        TA = instantiate_eltype(exA)
        TB = instantiate_eltype(exB)
        TC = Expr(:call, :promote_type, TA, TB, :(typeof($α)))
    else
        TC = Expr(:call, :eltype, dst)
    end

    if !isgeneraltensor(exA) || hastraceindices(exA)
        initA = instantiate(nothing, false, exA, true, oindA, cind, true)
        poA = ((1:length(oindA))...,)
        pcA = length(oindA) .+ ((1:length(cind))...,)
        conjA = :(:N)
        initA = Expr(:(=), symA, initA)
        αA = 1
    else
        A, indlA, indrA, αA, conj = decomposegeneraltensor(exA)
        indA = vcat(indlA, indrA)
        poA = (map(l->findfirst(isequal(l), indA), oindA)...,)
        pcA = (map(l->findfirst(isequal(l), indA), cind)...,)
        TA = dst === nothing ? :(float(eltype($A))) : :(eltype($dst))
        conjA = conj ? :(:C) : :(:N)
        initA = Expr(:(=), symA, A)
    end

    if !isgeneraltensor(exB) || hastraceindices(exB)
        initB = instantiate(nothing, false, exB, true, cind, oindB, true)
        poB = length(cind) .+ ((1:length(oindB))...,)
        pcB = ((1:length(cind))...,)
        conjB = :(:N)
        initB = Expr(:(=), symB, initB)
        αB = 1
    else
        B, indlB, indrB, αB, conj = decomposegeneraltensor(exB)
        indB = vcat(indlB, indrB)
        poB = (map(l->findfirst(isequal(l), indB), oindB)...,)
        pcB = (map(l->findfirst(isequal(l), indB), cind)...,)
        conjB = conj ? :(:C) : :(:N)
        initB = Expr(:(=), symB, B)
```

```
    end

    oindAB = vcat(oindA, oindB)
    p1 = (map(l->findfirst(isequal(l), oindAB), leftind)...,)
    p2 = (map(l->findfirst(isequal(l), oindAB), rightind)...,)
    if any(x->(x===nothing), (poA..., pcA..., poB..., pcB..., p1..., p2...)) ||
        !(isperm((poA...,pcA...)) && length(indA) == length(poA)+length(pcA)) ||
        !(isperm((pcB...,poB...)) && length(indB) == length(poB)+length(pcB)) ||
        !(isperm((p1...,p2...)) && length(oindAB) == length(p1)+length(p2))
        err = "contraction: $(tuple(leftind..., rightind...)) from
$(tuple(indA...,)) and $(tuple(indB...,)))"
        return :(throw(IndexError($err)))
    end
    if dst === nothing
        if istemporary
            initC = :($symC = cached_similar_from_indices($(QuoteNode(symC)),
$symTC, $poA, $poB, $p1, $p2, $symA, $symB, $conjA, $conjB))
        else
            initC = :($symC = similar_from_indices($symTC, $poA, $poB, $p1, $p2,
$symA, $symB, $conjA, $conjB))
        end
    else
        initC = :($symC = $dst)
    end

    return quote
        $symTC = $TC
        $initA
        $initB
        $initC
        contract!($α*$αA*$αB, $symA, $conjA, $symB, $conjB, $β, $symC,
                    $poA, $pcA, $poB, $pcB, $p1, $p2,
                    $((gensym(),gensym(),gensym())))
        # $symC
    end
end
```