```
function optdata(ex::Expr)
    allindices = getallindices(ex)
    cost = Power{:χ}(1,1)
    return Dict{Any, typeof(cost)}(i=>cost for i in allindices)
end

function optdata(optex::Expr, ex::Expr)
    if optex.head == :tuple
        isempty(optex.args) && return nothing
        args = optex.args
        if all(x -> isa(x, Expr) && x.head == :call && x.args[1] == :(=>), args)
            indices, costs = _optdata(map(x -> x.args[2], args), map(x ->
x.args[3], args))
            costtype = promote_type(typeof.(costs)...)
            costs = convert(Vector{costtype}, costs)
        elseif all(x -> isa(x, Expr) && x.head == :(=), args)
            indices, costs = _optdata(map(x -> x.args[1], args), map(x ->
x.args[2], args))
            costtype = promote_type(typeof.(costs)...)
            costs = convert(Vector{costtype}, costs)
        else
            indices = map(normalizeindex, args)
            costtype = Power{:χ,Int}
            costs = fill(Power{:χ,Int}(1,1), length(args))
        end
        return Dict{Any, costtype}(k=>v for (k,v) in zip(indices, costs))
    elseif optex.head == :call && optex.args[1] == :!
        allindices = unique(getallindices(ex))
        excludeind = map(normalizeindex, optex.args[2:end])
        cost = Power{:χ}(1,1)
        d = Dict{Any, typeof(cost)}(i=>cost for i in allindices)
        for i in excludeind
            d[i] = 1
        end
        return d
    else
        error("invalid index cost specification")
    end
end

function _optdata(indexvec, costvec)
    indices = Vector{Any}()
    costs = Vector{Any}()
    for (index, cost) in zip(indexvec, costvec)
        if typeof(index) != Symbol && index.head == :tuple
            for index_ in index.args
                push!(indices, normalizeindex(index_))
                push!(costs, parsecost(cost))
            end
        else
            push!(indices, normalizeindex(index))
            push!(costs, parsecost(cost))
        end
    end
```

```
    indices, costs
end

# Process index cost specification for @tensoropt and friends
function parsecost(ex::Expr)
    if ex.head == :call && ex.args[1] == :*
        return *(map(parsecost, ex.args[2:end])...)
    elseif ex.head == :call && ex.args[1] == :+
        return +(map(parsecost, ex.args[2:end])...)
    elseif ex.head == :call && ex.args[1] == :-
        return -(map(parsecost, ex.args[2:end])...)
    elseif ex.head == :call && ex.args[1] == :^
        return ^(map(parsecost, ex.args[2:end])...)
    elseif ex.head == :call && ex.args[1] == :/
        return /(map(parsecost, ex.args[2:end])...)
    elseif ex.head == :call && ex.args[1] == :big
        return big(map(parsecost, ex.args[2:end])...)
    elseif ex.head == :call && ex.args[1] == :float
        return float(map(parsecost, ex.args[2:end])...)
    elseif ex.head == :call && ex.args[1] == :Int128
        return Int128(map(parsecost, ex.args[2:end])...)
    else
        error("invalid index cost specification: $ex")
    end
end
parsecost(ex::Number) = ex
parsecost(ex::Symbol) = Power{ex}(1,1)
```