```julia
macro cutensor(ex::Expr)
    cuwrapdict = Dict{Any,Any}()
    parser = TensorParser()
    parser.preprocessors[end] = ex->extracttensorobjects(ex, cuwrapdict)
    push!(parser.postprocessors, ex->addcutensorwraps(ex, cuwrapdict))
    return esc(parser(ex))
end

macro cutensor(ex::Expr, orderex::Expr)
    cuwrapdict = Dict{Any,Any}()
    parser = TensorParser()
    if !(orderex.head == :(=) && orderex.args[1] == :order &&
            orderex.args[2] isa Expr && orderex.args[2].head == :tuple)
        throw(ArgumentError("unkown first argument in @tensor, should be `order =
(...,)`"))
    end
    indexorder = map(normalizeindex, orderex.args[2].args)
    parser.contractiontreebuilder = network->indexordertree(network, indexorder)
    parser.preprocessors[end] = ex->extracttensorobjects(ex, cuwrapdict)
    push!(parser.postprocessors, ex->addcutensorwraps(ex, cuwrapdict))
    return esc(parser(ex))
end

function extracttensorobjects(ex, cuwrapdict)
    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]...)
    cutensordict = Dict{Any,Any}(a => gensym() for a in alltensors)
    ex = replacetensorobjects((obj,leftind,rightind)->get(cutensordict, obj, obj),
ex)
    post = Expr(:block,
                [Expr(:call, :copyto!, a, cutensordict[a]) for a in
outputtensors]...,
                [Expr(:(=), a, cutensordict[a]) for a in newtensors]...)
    for k in inputtensors
        a = tensordict[k]
        b = cutensordict[k]
        push!(cuwrapdict, b=>(a,:($b = CuArray($a))))
    end
    for k in setdiff(outputtensors, inputtensors)
        a = tensordict[k]
        b = cutensordict[k]
        push!(cuwrapdict, b=>(a,:($b = CuArray{eltype($a)}(undef, size($a)))))
    end
    return Expr(:block, pre, ex, post)
end

function addcutensorwraps(ex, cuwrapdict)
    for (b,(a,assignb)) in cuwrapdict
        ex = _replace_in_eltype(ex, b, a)
        ex = _splice(ex, b, assignb)
    end
    return ex
end

function _replace_in_eltype(ex::Expr, b, a)
    if ex.head == :call && ex.args[1] == :eltype && ex.args[2] == b
        return Expr(:call, :eltype, a)
    else
        return Expr(ex.head, (_replace_in_eltype(arg, b, a) for arg in ex.args)...)
    end
end
_replace_in_eltype(ex, b, a) = ex

# check if a subexpression contains/uses the variable s, but ignore `eltype` calls
_contains(ex::Expr, s) = any(e->_contains(e, s), ex.args)
_contains(ex::Symbol, s) = ex == s
_contains(ex, s) = false

function _splice(ex::Expr, s1, assign)
    if ex.head == :block
        args = copy(ex.args)
        i = 1
        while i <= length(args)
            if _contains(args[i], s1)
                insert!(args, i, assign)
                break
            end
            i += 1
        end
        return Expr(ex.head, args...)
    elseif !(ex.head == :call && ex.args[1] == :throw)
        error("unexpected expression: $ex")
    end
end
```