```
# Verifiers: tools to verify if an expression is a valid tensor expression of
  certain type

# test for a valid index
const prime = Symbol("'")
function isindex(ex)
    if isa(ex, Symbol) || isa(ex, Int)
        return true
    elseif isa(ex, Expr) && ex.head == prime && length(ex.args) == 1
        return isindex(ex.args[1])
    else
        return false
    end
end

# test for a simple tensor object indexed by valid indices
istensor(ex) = false
function istensor(ex::Expr)
    if ex.head == :ref || ex.head == :typed_hcat
        if length(ex.args) == 1
            return true
        elseif isa(ex.args[2], Expr) && ex.args[2].head == :parameters
            return all(isindex, ex.args[2].args) || all(isindex, ex.args[3:end])
        else
            return all(isindex, ex.args[2:end])
        end
    elseif ex.head == :typed_vcat && length(ex.args) == 3
        length(ex.args) == 3 || return false
        if isa(ex.args[2], Expr) && (ex.args[2].head == :row || ex.args[2].head ==
:tuple)
            all(isindex, ex.args[2].args) || return false
        else
            isindex(ex.args[2]) || return false
        end
        if isa(ex.args[3], Expr) && (ex.args[3].head == :row || ex.args[3].head ==
:tuple)
            all(isindex, ex.args[3].args) || return false
        else
            isindex(ex.args[3]) || return false
        end
        return true
    end
    return false
end

# test for a generalized tensor, i.e. with scalar multiplication and conjugation
isgeneraltensor(ex) = false
function isgeneraltensor(ex::Expr)
    if istensor(ex)
        return true
    elseif ex.head == :call && ex.args[1] == :+ && length(ex.args) == 2
        # unary plus
        return isgeneraltensor(ex.args[2])
    elseif ex.head == :call && ex.args[1] == :- && length(ex.args) == 2
```

```
        # unary minus
        return isgeneraltensor(ex.args[2])
    elseif ex.head == :call && ex.args[1] == :conj && length(ex.args) == 2
        # conjugation
        return isgeneraltensor(ex.args[2])
    elseif ex.head == :call && ex.args[1] == :adjoint && length(ex.args) == 2
        # adjoint
        return isgeneraltensor(ex.args[2])
    elseif ex.head == prime && length(ex.args) == 1
        # adjoint
        return isgeneraltensor(ex.args[1])
    elseif ex.head == :call && ex.args[1] == :transpose && length(ex.args) == 2
        # conjugation
        return isgeneraltensor(ex.args[2])
    elseif ex.head == :call && ex.args[1] == :*
        # scalar multiplication
        count = 0
        for i = 2:length(ex.args)
            if isgeneraltensor(ex.args[i])
                count += 1
            elseif !isscalarexpr(ex.args[i])
                return false
            end
        end
        return count == 1
    elseif ex.head == :call && ex.args[1] == :/ && length(ex.args) == 3
        # scalar multiplication
        if isscalarexpr(ex.args[3]) && isgeneraltensor(ex.args[2])
            return true
        end
    elseif ex.head == :call && ex.args[1] == :\ && length(ex.args) == 3
        # scalar multiplication
        if isscalarexpr(ex.args[2]) && isgeneraltensor(ex.args[3])
            return true
        end
    end
    return false
end

function hastraceindices(ex)
    obj, leftind, rightind, = decomposegeneraltensor(ex)
    allind = vcat(leftind, rightind)
    return length(allind) != length(unique(allind))
end

# test for a scalar expression, i.e. no indices
function isscalarexpr(ex::Expr)
    if ex.head == :call && ex.args[1] == :scalar
        return istensorexpr(ex.args[2])
    elseif ex.head in (:ref, :typed_vcat, :typed_hcat)
        return false
    else
        return all(isscalarexpr, ex.args)
    end
end
```

```
isscalarexpr(ex::Symbol) = true
isscalarexpr(ex::Number) = true
isscalarexpr(ex) = true

# test for a tensor contraction expression
function istensorcontraction(ex)
    if isa(ex, Expr) && ex.head == :call && ex.args[1] == :*
        return count(istensorexpr, ex.args[2:end]) >= 2
    end
    return false
end

# test for a tensor expression, i.e. something that can be evaluated to a tensor
function istensorexpr(ex)
    if isgeneraltensor(ex)
        return true
    elseif isa(ex, Expr) && ex.head == :call && (ex.args[1] == :+ || ex.args[1] ==
:-)
        return all(istensorexpr, ex.args[2:end]) # all arguments should be tensor
            expressions (we are not checking matching indices yet)
    elseif isa(ex, Expr) && ex.head == :call && ex.args[1] == :*
        count = 0
        for i = 2:length(ex.args)
            if istensorexpr(ex.args[i])
                count += 1
            elseif !isscalarexpr(ex.args[i])
                return false
            end
        end
        return count > 0
    elseif isa(ex, Expr) && ex.head == :call && ex.args[1] == :/ && length(ex.args)
== 3
        return istensorexpr(ex.args[2]) && isscalarexpr(ex.args[3])
    elseif isa(ex, Expr) && ex.head == :call && ex.args[1] == :\ && length(ex.args)
== 3
        return istensorexpr(ex.args[3]) && isscalarexpr(ex.args[2])
    elseif isa(ex, Expr) && ex.head == :call && ex.args[1] == :conj &&
length(ex.args) == 2
        return istensorexpr(ex.args[2])
    elseif isa(ex, Expr) && ex.head == :call && ex.args[1] == :adjoint &&
length(ex.args) == 2
        return istensorexpr(ex.args[2])
    elseif isa(ex, Expr) && ex.head == prime
        return istensorexpr(ex.args[1])
    end
    return false
end

# test for assignment (copy into existing tensor) or definition (create new tensor)
isassignment(ex) = false
isdefinition(ex) = false
isassignment(ex::Expr) = ex.head == :(=) || ex.head == :(+=) || ex.head == :(-=)
isdefinition(ex::Expr) = (ex.head == :(:=) || ex.head == :(≔)) &&
istensor(ex.args[1])
```