```
# FusionTreeIterator:
# iterate over fusion trees for fixed coupled and uncoupled sector labels
#==============================================================================#
function fusiontrees(uncoupled::NTuple{N,G}, coupled::G = one(G),
                         isdual::NTuple{N,Bool} = ntuple(n->false, Val(N))) where
                         {N,G<:Sector}
    FusionTreeIterator{G,N}(uncoupled, coupled, isdual)
end

struct FusionTreeIterator{G<:Sector,N}
    uncoupled::NTuple{N,G}
    coupled::G
    isdual::NTuple{N,Bool}
end

Base.IteratorSize(::FusionTreeIterator) = Base.HasLength()
Base.IteratorEltype(::FusionTreeIterator) = Base.HasEltype()
Base.eltype(T::Type{FusionTreeIterator{G,N}}) where {G<:Sector, N} =
    fusiontreetype(G, StaticLength(N))

Base.length(iter::FusionTreeIterator) = _fusiondim(iter.uncoupled, iter.coupled)
_fusiondim(u::Tuple{}, c::G) where {G<:Sector} = Int(one(c) == c)
_fusiondim(u::Tuple{G}, c::G) where {G<:Sector} = Int(u[1] == c)
_fusiondim((a,b)::Tuple{G,G}, c::G) where {G<:Sector} = Int(Nsymbol(a, b, c))
function _fusiondim(u::Tuple{G,G,Vararg{G}}, c::G) where {G<:Sector}
    a = u[1]
    b = u[2]
    d = 0
    for c′ in a ⊗ b
        d += Nsymbol(a, b, c′)*_fusiondim((c′, TupleTools.tail2(u)...), c)
    end
    return d
end

# * Iterator methods:
#   Start with special cases:
function Base.iterate(it::FusionTreeIterator{G,0},
                         state = (it.coupled != one(G))) where {G<:Sector}
    state && return nothing
    T = vertex_labeltype(G)
    tree = FusionTree{G,0,0,0,T}((), one(G), (), (), ())
    return tree, true
end

function Base.iterate(it::FusionTreeIterator{G,1},
                         state = (it.uncoupled[1] != it.coupled)) where {G<:Sector}
    state && return nothing
    T = vertex_labeltype(G)
    tree = FusionTree{G,1,0,0,T}(it.uncoupled, it.coupled, it.isdual, (), ())
    return tree, true
end

#   General case:
function Base.iterate(it::FusionTreeIterator{G,N} where {N}) where {G<:Sector}
```

```
    next = _iterate(it.uncoupled, it.coupled)
    next === nothing && return nothing
    lines, vertices, states = next
    vertexlabels = labelvertices(it.uncoupled, it.coupled, lines, vertices)
    f = FusionTree(it.uncoupled, it.coupled, it.isdual, lines, vertexlabels)
    return f, (lines, vertices, states)
end
function Base.iterate(it::FusionTreeIterator{G,N} where {N}, state) where
{G<:Sector}
    next = _iterate(it.uncoupled, it.coupled, state...)
    next === nothing && return nothing
    lines, vertices, states = next
    vertexlabels = labelvertices(it.uncoupled, it.coupled, lines, vertices)
    f = FusionTree(it.uncoupled, it.coupled, it.isdual, lines, vertexlabels)
    return f, (lines, vertices, states)
end

labelvertices(uncoupled::NTuple{2,G}, coupled::G, lines::Tuple{},
                vertices::Tuple{Int}) where {G<:Sector} =
    (vertex_ind2label(vertices[1], uncoupled..., coupled),)

function labelvertices(uncoupled::NTuple{N,G}, coupled::G, lines,
                       vertices) where {G<:Sector,N}
    c = lines[1]
    resttree = tuple(c, TupleTools.tail2(uncoupled)...)
    rest = labelvertices(resttree, coupled, tail(lines), tail(vertices))
    l = vertex_ind2label(vertices[1], uncoupled[1], uncoupled[2], c)
    return (l, rest...)
end

# Actual implementation
@inline function _iterate(uncoupled::NTuple{2,G}, coupled::G, lines = (),
                          vertices = (0,), states = ()) where {G<:Sector}
    a, b = uncoupled
    n = vertices[1] + 1
    n > Nsymbol(a,b, coupled) && return nothing
    return (), (n,), ()
end

function _iterate(uncoupled::NTuple{N,G}, coupled::G) where {N, G<:Sector}
    a, b, = uncoupled
    it = a ⊗ b
    next = iterate(it)
    next === nothing && return nothing
    # this should not happen: there should always be at least one fusion output
    c, s = next
    resttree = tuple(c, TupleTools.tail2(uncoupled)...)
    rest = _iterate(resttree, coupled)
    while rest === nothing
        next = iterate(it, s)
        next === nothing && return nothing
        c, s = next
        resttree = tuple(c, TupleTools.tail2(uncoupled)...)
        rest = _iterate(resttree, coupled)
```

```
    end
    n = 1
    restlines, restvertices, reststates = rest
    lines = (c, restlines...)
    vertices = (n, restvertices...)
    states = (s, reststates...)
    return lines, vertices, states
end

function _iterate(uncoupled::NTuple{N,G}, coupled::G, lines, vertices,
                  states) where {N, G<:Sector}
    a, b, = uncoupled
    it = a ⊗ b
    c = lines[1]
    n = vertices[1]
    s = states[1]
    restlines = tail(lines)
    restvertices = tail(vertices)
    reststates = tail(states)
    if n < Nsymbol(a, b, c)
        n += 1
        return lines, (n, restvertices...), states
    end
    n = 1
    resttree = tuple(c, TupleTools.tail2(uncoupled)...)
    rest = _iterate(resttree, coupled, restlines, restvertices, reststates)
    while rest === nothing
        next = iterate(it, s)
        next === nothing && return nothing
        c, s = next
        resttree = tuple(c, TupleTools.tail2(uncoupled)...)
        rest = _iterate(resttree, coupled)
    end
    restlines, restvertices, reststate = rest
    lines = (c, restlines...)
    vertices = (n, restvertices...)
    states = (s, reststate...)
    return lines, vertices, states
end
```