In [1]:
```
using DynamicPolynomials,LinearAlgebra
using JuMP
import HiGHS
```

Match Polynomials

In [2]:
```
function getLP(poly,arrayOfPolynomials,var)
    d = maxdegree(poly)
    base=monomials(var,d)
    b=coefficients(poly,base)
    c=[]
    nVars=[]
    for f in arrayOfPolynomials
        deg=d - maxdegree(f)
        push!(nVars,size(monomials(var,deg))[1])
        l = []
        for g in monomials(var,deg)
            basepoly=g*f
            push!(l,coefficients(basepoly,monomials(var,d)))
        end
        l=transpose(reduce(vcat,transpose.(l)))
        push!(c,l)
    end
    c=transpose(reduce(vcat,transpose.(c)))

    #Create Linear Program
    m = size(b,1) #Number of Monomials in Polynom to Match
    LP = Model(HiGHS.Optimizer) # Initialize Model
    set_optimizer_attribute(LP, "log_to_console", "false") # disable debug info
    n=size(c,2) # Number of Variables
    @variable(LP, u[1:n]>=0)
    for i in 1:m
        @constraint(LP, sum(dot(u,c[i,:])) == b[i])
    end
    # print(LP)
    return (LP,u,b,c,nVars)
end
```

Out[2]:
```
getLP (generic function with 1 method)
```

In [3]:
```
function getSolution(poly,arrayOfPolynomials,var)
    LP,u,b,c,n = getLP(poly,arrayOfPolynomials,var)
    # print(LP)
    JuMP.optimize!(LP)
    if JuMP.has_values(LP)
        sol=JuMP.value.(u)
        walk=1
        c=[]
        solPoly=[]
        for (j,i) in enumerate(n)
            coeff = view(sol,walk:walk+i-1)
            base = monomials(var,maxdegree(poly)-maxdegree(arrayOfPolynomials[j]))
            push!(solPoly,polynomial(coeff,base))
            push!(c,coeff)
            walk+=i
        end
        return (solPoly,c)
    else
        return false
```

```
    end
end
```

Out[3]:
```
getSolution (generic function with 1 method)
```

Example Functions

In [4]:
```
@polyvar x[1:3]
h = 3x[1] - 2x[2] - 2x[3]
f1 = 1 + 0*x[1]
f2 = x[1]-x[2]
f3 = x[1]-x[3]
f4 = x[1]^2 - 4x[2]x[3]
F=[f1,f2,f3,f4]
e=polynomial(monomials(x,1))*h
```

Out[4]:
$$ 3x_{1}^{2} + x_{1}x_{2} + x_{1}x_{3} - 2x_{2}^{2} - 4x_{2}x_{3} - 2x_{3}^{2} $$

In [6]:
```
function Run(poly,arrayOfPolynomials,maxeval)
    var = variables([arrayOfPolynomials...,poly])
    for i in 1:maxeval
        e=polynomial(monomials(var,1))^i*poly
        Sol,c = getSolution(e,arrayOfPolynomials,var)
        if Sol != false
            return Sol,c
        end
    end
    return false
end
```

Out[6]:
```
Run (generic function with 1 method)
```

Solution

In [7]:
```
Sol, c = Run(h,F,5)
Sol
```

Out[7]:
```
4-element Vector{Any}:
 0.0
 x₁ + 2.0x₂
 x₁ + 2.0x₃
 1.0
```

In [ ]:
```
```