JuMP is great, but how do I …

- add support for a new type of constraint?
- combine NLP constraints with conic constraints?
- delete a constraint or variable?
- test if a solution is feasible?
- modify coefficients in the constraint matrix?
- provide a dual warm-start?
- access the irreducible inconsistent subsystem (IIS) from Gurobi?
- distinguish between a solver that stopped because of the time limit 1) with a solution and 2) without?
- handle CPXMIP_OPTIMAL_INFEAS?
JuMP’s architecture (0.1 to 0.18)
MathProgBase
MathOptInterface (MOI)

- New problem representation
  - Extensible way to define categories of constraints and modifications
- New interface for attributes
  - Warm starts, IIS
- New status codes
- Nonconsecutive variable and constraint indices
  - For deletion
- Callbacks \textit{not} defined; they become solver-specific
- Nonlinear optimization mostly unchanged
MOI definition of an optimization problem

The standard form problem is:

\[
\begin{align*}
\min_{x \in \mathbb{R}^n} & \quad f_0(x) \\
\text{s.t.} & \quad f_i(x) \in S_i, \quad i = 1 \ldots m
\end{align*}
\]

where:

- the functions \( f_0, f_1, \ldots, f_m \) are specified by \texttt{AbstractFunction} objects
- the sets \( S_1, \ldots, S_m \) are specified by \texttt{AbstractSet} objects
Standard functions

The current function types are:

- **SingleVariable**: $x_j$, projection onto a single coordinate defined by a variable index $j$
- **VectorOfVariables**: projection onto multiple coordinates (i.e., extracting a subvector)
- **ScalarAffineFunction**: $a^T x + b$, where $a$ is a vector and $b$ is scalar
- **VectorAffineFunction**: $Ax + b$, where $A$ is a matrix and $b$ is a vector
- **ScalarQuadraticFunction**: $\frac{1}{2}x^T Q x + a^T x + b$, where $Q$ is a symmetric matrix, $a$ is a vector, and $b$ is a constant
- **VectorQuadraticFunction**: a vector of scalar-valued quadratic functions
And their definitions...

```

""

    VariableIndex

A type-safe wrapper for `Int64` for use in referencing variables in a model.
To allow for deletion, indices need not be consecutive.

""

struct VariableIndex
    value::Int64
end

struct SingleVariable <: AbstractScalarFunction
    variable::VariableIndex
end
```
struct ScalarAffineTerm{T}
    coefficient::T
    variable_index::VariableIndex
end

mutable struct ScalarAffineFunction{T} <: AbstractScalarFunction
    terms::Vector{ScalarAffineTerm{T}}
    constant::T
end
A few sets

- \text{LessThan}(\text{upper}): \{x \in \mathbb{R} : x \leq \text{upper}\}
- \text{GreaterThan}(\text{lower}): \{x \in \mathbb{R} : x \geq \text{lower}\}
- \text{EqualTo}(\text{value}): \{x \in \mathbb{R} : x = \text{value}\}
- \text{Interval}(\text{lower}, \text{upper}): \{x \in \mathbb{R} : x \in [\text{lower}, \text{upper}]\}
- \text{SecondOrderCone}(\text{dimension}): \{(t, x) \in \mathbb{R}^{\text{dimension}} : t \geq ||x||_2\}
- \text{Integer}(): \mathbb{Z}
- \text{ZeroOne}():: \{0, 1\}
And their definitions...

```text
82 struct LessThan{T <: Real} <: AbstractScalarSet
83     upper::T
84 end

126 struct SecondOrderCone <: AbstractVectorSet
127     dimension::Int
128 end

347 struct ZeroOne <: AbstractScalarSet end
```
### Linear constraints

<table>
<thead>
<tr>
<th>Mathematical Constraint</th>
<th>MOI Function</th>
<th>MOI Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a^T x \leq u$</td>
<td>ScalarAffineFunction</td>
<td>LessThan</td>
</tr>
<tr>
<td>$a^T x \geq l$</td>
<td>ScalarAffineFunction</td>
<td>GreaterThan</td>
</tr>
<tr>
<td>$a^T x = b$</td>
<td>ScalarAffineFunction</td>
<td>EqualTo</td>
</tr>
<tr>
<td>$l \leq a^T x \leq u$</td>
<td>ScalarAffineFunction</td>
<td>Interval</td>
</tr>
<tr>
<td>$x_i \leq u$</td>
<td>SingleVariable</td>
<td>LessThan</td>
</tr>
<tr>
<td>$x_i \geq l$</td>
<td>SingleVariable</td>
<td>GreaterThan</td>
</tr>
<tr>
<td>$x_i = b$</td>
<td>SingleVariable</td>
<td>EqualTo</td>
</tr>
<tr>
<td>$l \leq x_i \leq u$</td>
<td>SingleVariable</td>
<td>Interval</td>
</tr>
<tr>
<td>$Ax + b \in \mathbb{R}^n_+$</td>
<td>VectorAffineFunction</td>
<td>Nonnegatives</td>
</tr>
<tr>
<td>$Ax + b \in \mathbb{R}^n_-$</td>
<td>VectorAffineFunction</td>
<td>Nonpositives</td>
</tr>
<tr>
<td>$Ax + b = 0$</td>
<td>VectorAffineFunction</td>
<td>Zeros</td>
</tr>
</tbody>
</table>
# Quadratic constraints

<table>
<thead>
<tr>
<th>Mathematical Constraint</th>
<th>MOI Function</th>
<th>MOI Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x^T Q x + a^T x + b \geq 0$</td>
<td>ScalarQuadraticFunction</td>
<td>GreaterThan</td>
</tr>
<tr>
<td>$x^T Q x + a^T x + b \leq 0$</td>
<td>ScalarQuadraticFunction</td>
<td>LessThan</td>
</tr>
<tr>
<td>$x^T Q x + a^T x + b = 0$</td>
<td>ScalarQuadraticFunction</td>
<td>EqualTo</td>
</tr>
<tr>
<td>Bilinear matrix inequality</td>
<td>VectorQuadraticFunction</td>
<td>PositiveSemidefiniteCone...</td>
</tr>
</tbody>
</table>
### Discrete and logical constraints

<table>
<thead>
<tr>
<th>Mathematical Constraint</th>
<th>MOI Function</th>
<th>MOI Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_i \in \mathbb{Z} )</td>
<td>SingleVariable</td>
<td>Integer</td>
</tr>
<tr>
<td>( x_i \in {0, 1} )</td>
<td>SingleVariable</td>
<td>ZeroOne</td>
</tr>
<tr>
<td>( x_i \in {0} \cup [l, u] )</td>
<td>SingleVariable</td>
<td>Semicontinuous</td>
</tr>
<tr>
<td>( x_i \in {0} \cup {l, l+1, \ldots, u-1, u} )</td>
<td>SingleVariable</td>
<td>Semiinteger</td>
</tr>
<tr>
<td>At most one component of ( x ) can be nonzero</td>
<td>VectorOfVariables</td>
<td>SOS1</td>
</tr>
<tr>
<td>At most two components of ( x ) can be nonzero, and if so they must be adjacent components</td>
<td>VectorOfVariables</td>
<td>SOS2</td>
</tr>
</tbody>
</table>
Adding a constraint at the MOI level

```python
model = OSQPObjective()
var_index = MOI.addvariable!(model)
constr_index = MOI.addconstraint!(model,
    MOI.SingleVariable(var_index),
    MOI.LessThan(10.0))
```
addconstraint! returns an index

```
""

ConstraintIndex{F,S}

A type-safe wrapper for `Int64` for use in referencing `F`-in-`S` constraints in
a model.
The parameter `F` is the type of the function in the constraint, and the
parameter `S` is the type of set in the constraint. To allow for deletion,
indices need not be consecutive. Indices within a constraint type (i.e. `F`-in-`S`)  
must be unique, but non-unique indices across different constraint types are allowed.
""

struct ConstraintIndex{F,S}
    value::Int64
end
```
Deleting variables and constraints

```python
MOI.isvalid(model, variable_index)  # True
MOI.isvalid(model, constraint_index)  # True

MOI.delete!(model, variable_index)
MOI.delete!(model, constraint_index)

MOI.isvalid(model, variable_index)  # False
MOI.isvalid(model, constraint_index)  # False
```
Adding a constraint at the JuMP level

```python
m = JuMP.Model(optimizer = GurobiOptimizer())

@variable(m, x <= 10)  # SingleVariable-LessThan
@variable(m, y)

@constraint(m, x + y >= 10)  # ScalarAffineFunction-GreaterThan
@constraint(m, [x,y] in MOI.SecondOrderCone(2))  # VectorOfVariables-

# Hypothetical
@constraint(m, [x,1-x] in MOI.Complements())  # VectorAffineFunction-
```
@constraint returns a reference

```plaintext
269  struct ConstraintRef{M<:AbstractModel,C}
270       m::M
271       index::C
272  end
```

For example,

```
ConstraintRef{Model, MOI.ConstraintIndex{MOI.ScalarAffineFunction, MOI.LessThan}}
```
Discussion

There are multiple ways to write down the same constraint. Which should a solver support? *Who should do the work of transforming the problem?*

- **VectorAffineFunction-in-Zeros vs. multiple ScalarAffineFunction-in-EqualsTo**?
- **GemetricMeanCone versus PowerCone versus SecondOrderCone**?

MOI provides a single framework in which to experiment with different representations of a problem at *both the model and solver level*. 
Attributes in MOI

```python
model = OSQPOptimizer()
var_index = MOI.addvariable!(model)
constr_index = MOI.addconstraint!(model,
                MOI.SingleVariable(var_index),
                MOI.LessThan(10.0))

MOI.set!(model, MOI.ObjectiveSense(), MOI.MaxSense)

MOI.set!(model, MOI.ObjectiveFunction(), [...])

MOI.set!(model, var_index, MOI.VariablePrimalStart(), 5.0)

MOI.set!(model, constr_index, MOI.ConstraintDualStart(), 10.0)
```
Attributes in JuMP

```julia
model = JuMP.Model(optimizer = OSQPOptimizer())
@variable(model, x >= 10.0, start = 5.0)
@objective(model, Max, [...])

bound_ref = JuMP.LowerBoundRef(x)
MOI.set!(model, bound_ref, MOI.ConstraintDualStart(), 10.0)
```
From MOI to JuMP

87     ""
88     VariableRef <: AbstractVariableRef
89
90     Holds a reference to the model and the corresponding MOI.VariableIndex.
91     ""
92     struct VariableRef <: AbstractVariableRef
93         m::Model
94         index::MOIVAR
95     end
96
94     function MOI.set!(m::Model, attr::MOI.AbstractVariableAttribute, v::VariableRef, value)
95         @assert m === v.m
96         MOI.set!(m.moibackend, attr, index(v), value)
97     end
98
475    startvalue(v::VariableRef) = MOI.get(v.m, MOI.VariablePrimalStart(), v)
476    setstartvalue(v::VariableRef, val::Number) = MOI.set!(v.m, MOI.VariablePrimalStart(), v, val)
Status codes

1. Why did the solver stop? `TerminationStatus()`
2. Does the solver have vectors to return? `ResultCount()`
3. What do the result vectors mean? `PrimalStatus()` and `DualStatus()`
# Example situations with primal-dual solvers

<table>
<thead>
<tr>
<th>What happened?</th>
<th>TerminationStatus</th>
<th>Result Count</th>
<th>PrimalStatus</th>
<th>DualStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proved optimality*</td>
<td>Success</td>
<td>1</td>
<td>FeasiblePoint</td>
<td>FeasiblePoint</td>
</tr>
<tr>
<td>Proved primal infeasible</td>
<td>Success</td>
<td>1</td>
<td>error</td>
<td>InfeasibilityCertificate</td>
</tr>
<tr>
<td>Optimal within relaxed tolerances</td>
<td>AlmostSuccess</td>
<td>1</td>
<td>FeasiblePoint or AlmostFeasiblePoint</td>
<td>FeasiblePoint or AlmostFeasiblePoint</td>
</tr>
<tr>
<td>Stall</td>
<td>SlowProgress</td>
<td>1</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

* within numerical tolerances or given optimality gap
## Example situations with MIP solvers

<table>
<thead>
<tr>
<th>What happened?</th>
<th>TerminationStatus</th>
<th>ResultCount</th>
<th>PrimalStatus</th>
<th>DualStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proved optimality</td>
<td>Success</td>
<td>1</td>
<td>FeasiblePoint</td>
<td>error</td>
</tr>
<tr>
<td>Proved infeasible or unbounded</td>
<td>InfeasibleOrUnbounded</td>
<td>0</td>
<td>error</td>
<td>error</td>
</tr>
<tr>
<td>Proved infeasible</td>
<td>InfeasibleNoResult</td>
<td>0</td>
<td>error</td>
<td>error</td>
</tr>
<tr>
<td>Timed out (no solution)</td>
<td>TimeLimit</td>
<td>0</td>
<td>error</td>
<td>error</td>
</tr>
<tr>
<td>Timed out (with solution)</td>
<td>TimeLimit</td>
<td>1</td>
<td>FeasiblePoint</td>
<td>error</td>
</tr>
<tr>
<td>CPXMIP_OPTIMAL_INFEAS</td>
<td>Success</td>
<td>1</td>
<td>InfeasiblePoint</td>
<td>error</td>
</tr>
</tbody>
</table>
## Example situations with NLP solvers

<table>
<thead>
<tr>
<th>What happened?</th>
<th>TerminationStatus</th>
<th>Result Count</th>
<th>PrimalStatus</th>
<th>DualStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Converged to feasible point</td>
<td>Success</td>
<td>1</td>
<td>FeasiblePoint</td>
<td>FeasiblePoint</td>
</tr>
<tr>
<td>Converged to infeasible point</td>
<td>Success</td>
<td>1</td>
<td>InfeasiblePoint</td>
<td>FeasiblePoint</td>
</tr>
<tr>
<td>Iteration limit</td>
<td>IterationLimit</td>
<td>1</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Diverging</td>
<td>NormLimit or ObjectiveLimit</td>
<td>1</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Modifications

- $x \leq 1 \Rightarrow x \leq 2$: set ConstraintSet attribute
- $x \leq 1 \Rightarrow x \geq 2$: call MOI.transform!
- $2x + y \leq 10 \Rightarrow 3x + y \leq 10$: set ConstraintFunction attribute or call MOI.modify! with ScalarCoefficientChange
The state of MOI

MOI 0.4 released this week. 700+ commits, 400 issues/PRs.

Jun 25, 2017 – Jun 27, 2018

Contributions to master, excluding merge commits
# Status of solver wrappers

<table>
<thead>
<tr>
<th>Released version supports MOI:</th>
<th>MOI support in PR or master branch:</th>
<th>Up for grabs:</th>
</tr>
</thead>
<tbody>
<tr>
<td>● CSDP</td>
<td>● Cbc</td>
<td>● AmplNLWriter</td>
</tr>
<tr>
<td>● ECOS</td>
<td>● Clp</td>
<td>● CPLEX</td>
</tr>
<tr>
<td>● Ipopt</td>
<td>● Gurobi</td>
<td>● Knitro</td>
</tr>
<tr>
<td>● Mosek</td>
<td>● GLPK</td>
<td>● NLopt</td>
</tr>
<tr>
<td>● OSQP</td>
<td>● Xpress</td>
<td>● Pajarito</td>
</tr>
<tr>
<td></td>
<td>● SDPA</td>
<td>● SCIP</td>
</tr>
<tr>
<td></td>
<td>● SCS</td>
<td></td>
</tr>
</tbody>
</table>

SemidefiniteOptInterface and LinQuadOptInterface are optional helper layers for implementing MOI wrappers.
JuMP/MOI

What hasn’t changed:

- Macro syntax
  - Except for `norm()`
- Variable construction syntax
- Automatic differentiation (ReverseDiffSparse moved into JuMP repo.)
What *has* changed

- Containers
- Names
- Data structures for AffExpr and QuadExpr
- Interacting with solvers
- Software engineering improvements
- Documenter and docstrings for documentation
JuMP containers

- **JuMPDict replaced by** `Base.Dict`
- **JuMPArray rewritten, inspired by** `AxisArrays`
Review of JuMP containers

```julia
@variable(m, [1:5, 1:5])  # Array
set_1 = Base.OneTo(5)
@variable(m, [set_1, 1:5])  # Array
set_2 = 1:5
@variable(m, [1:5, set_2])  # JuMPArray
set_3 = [a, b, c]
@variable(m, [set_2, set_3])  # JuMPArray
@variable(m, [i=set_2, 1:i])  # Dict
@variable(m, [i = 1:5; isodd(i)])  # Dict
```

Same applies for @constraint, @expression, @NLconstraint, @NLexpression
Now possible to request a container type

```python
@variable(m, [1:5, 1:5], container=JuMPArray)
set_1 = 1:5
@variable(m, [set_1, 1:5], container=Array)
set_2 = 2:3
@variable(m, [set_2, 1:5], container=Array) # Error
```
New JuMPArrays

\[ \text{set}_1 = [\text{a}, \text{b}, \text{c}] \]
\[ \text{set}_2 = [\text{x}, \text{y}, \text{z}] \]

# 0.18
```
julia> @variable(m, x[set_1, set_2])
x[i,j] \forall i \in \{a,b,c\}, j \in \{x,y,z\}
```

julia> x[:, :, z]
ERROR: Failed attempt to index JuMPArray ...
   along dimension 1: Colon() ∈ Symbol[\text{a}, \text{b}, \text{c}]

# 0.19
```
julia> @variable(m, x[set_1, set_2])
2-dimensional JuMPArray{JuMP.VariableRef, 2, ...} with index sets:
   Dimension 1, Symbol[\text{a}, \text{b}, \text{c}]
   Dimension 2, Symbol[\text{x}, \text{y}, \text{z}]
And data, a 3×3 Array{JuMP.VariableRef, 2}:
x[a,x] x[a,y] x[a,z]
x[b,x] x[b,y] x[b,z]
x[c,x] x[c,y] x[c,z]
```
```
julia> x[:, :, z]
1-dimensional JuMPArray{JuMP.VariableRef, 1, ...} with index sets:
   Dimension 1, Symbol[\text{a}, \text{b}, \text{c}]
And data, a 3-element Array{JuMP.VariableRef, 1}:
x[a,z]
x[b,z]
x[c,z]
Names

- Variables and constraints now have string names. The set of nonempty names is unique. You can lookup by name.
- Model scope (`model[:x]`) still exists, useful for non-scalar variables.
AffExpr and QuadExpr

0.18

```c
mutable struct GenericAffExpr{CoefType, VarType}
    vars::Vector{VarType}
    coeffs::Vector{CoefType}
    constant::CoefType
end
```

0.19

```c
mutable struct GenericAffExpr{CoefType, VarType}
    constant::CoefType
    terms::OrderedDict{VarType, CoefType}
end
```

```c
mutable struct GenericQuadExpr{CoefType, VarType}
    qvars1::Vector{VarType}
    qvars2::Vector{VarType}
    qcoeffs::Vector{CoefType}
    aff::GenericAffExpr{CoefType, VarType}
end
```

- No duplicates by construction
- Faster on 0.7
The multi-backend problem

JuMP now supports many kinds of problem modifications. Solvers support a subset of these. We want to keep the solver in memory and pass modifications efficiently when possible.
JuMP stores the problem data only in MOI

```julia
addconstraint(m::Model, c::AbstractConstraint, name::String="")

Add a constraint `c` to `Model m` and sets its name.

```julia
function addconstraint(m::Model, c::AbstractConstraint, name::String="")
    cindex = MOI.addconstraint!(m.moibackend, moi_function_and_set(c)...)  
    cref = ConstraintRef(m, cindex)  
    if !isempty(name)
        setname(cref, name)
    end
    end
    return cref
end
```
JuMP solver modes

- **Direct**: the `moibackend` field is a solver (e.g., Gurobi)
  - This is the mode for using callbacks.

- **Manual**: the `moibackend` field is a `CachingOptimizer` in Manual mode
  - A solver is “attached” or “empty”. When the solver is attached, error if user attempts to make a modification that the solver doesn’t support.

- **Automatic**: the `moibackend` field is a `CachingOptimizer` in Automatic mode
  - Solver is attached and emptied when needed without notice.
Are my incremental modifications efficiently passed to the solver?

Direct: Yes, you’ll always get an error when you make a change that the solver doesn’t support.

Manual: Yes, you control when the model is reloaded into the solver.

Automatic: Maybe, this will happen silently.
Software engineering improvements

JuMP’s tests no longer depend on a solver!

- MOI lightweight text format for testing model generation
- Mock solvers for testing communication with a solver
@testset "Generation and solve with fake solver" begin
@testset "LP" begin
    m = Model()
    @variable(m, x <= 2.0)
    @variable(m, y >= 0.0)
    @objective(m, Min, -x)
    c = @constraint(m, x + y <= 1)
    JuMP.setname(JuMP.UpperBoundRef(x), "xub")
    JuMP.setname(JuMP.LowerBoundRef(y), "ylb")
    JuMP.setname(c, "c")

    modelstring = ""
    variables: x, y
    minobjective: -1.0*x
    xub: x <= 2.0
    ylb: y >= 0.0
    c: x + y <= 1.0
    ""

    model = JuMP.JuMPMODModel{Float64}()
    MOIU.loadfromstring!(model, modelstring)
    MOIU.test_models_equal(m.moibackend.model_cache, model, ["x","y"], ["c", "xub", "ylb"])
"
Style guide and design principles

Style guide

TODO: A style guide for JuMP, JuMP models and surrounding Julia code. Formatting, naming, use of macros, comments, TODOs, docstrings, etc.

Design principles

TODO: How to structure and test large JuMP models, libraries that use JuMP.

For how to write a solver, see MOI.
Remaining JuMP TODOs for 0.19

- Model printing
- Clean up API
  - Modifications not exposed
  - Names not well exposed
  - Getting/setting the solver
- Documentation
  - Guide for updating from 0.18
  - Update examples
- Callbacks
- Support 0.7
Maintenance plans for JuMP/MPB

MPB and MOI wrappers will co-exist for some time.

We would like to release a version of JuMP 0.18 that’s compatible with Julia 0.7.
Yesterday’s announcement

JuMP will be a NumFOCUS Sponsored Project!
NumFOCUS will help with:

- Receiving Donations
- Receiving Grants
- Google Summer of Code
- Holding funds for the JuMP-dev workshop
- Facilitating contracts for open-source work
New JuMP branding

- JuMP needs a (new) website
- Low priority: New name for the GitHub organization?
Thanks!