Ruby on Rails master@1c81926

Module

ActiveRecord::Enum

activerecord/lib/active_record/enum.rb

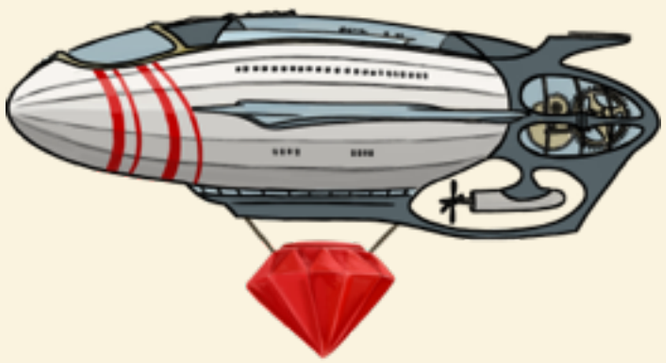
Declare an enum attribute where the values map to integers in the database,

```ruby
class Conversation < ActiveRecord::Base
  enum status: [ :active, :archived ]
end

# conversation.update! status: 0
conversation.active!
conversation.active? # => true
conversation.status  # => "active"
```

```
class Video < ActiveRecord::Base
  enum status: [ :active, :archived ]
end
```

```ruby
class Video < ActiveRecord::Base
  enum status: [
    :uploaded,
    :encoded,
    :encoding_failed,
    :uploading_to_s3,
    :uploaded_to_s3,
    :upload_failed,
    :ready,
    :archived
  ]
end
```

Bodo Tasche
@bitboxer



@bitcrowd

```ruby
class Video < ActiveRecord::Base
  enum status: [
    :uploaded,
    :encoded,
    :encoding_failed,
    :uploading_to_s3,
    :uploaded_to_s3,
    :upload_failed,
    :ready,
    :archived
]
```

```
video = Video.create(state: uploaded)
...
video.state = :ready
```

```ruby
validates :state_is_correct

def state_is_correct
  if self.ready? && file_not_uploaded?
    errors.add(:state, "can't be ready")
  elsif ...
    # 50 more ugly lines to protect the state
  end
end
```

```ruby
def transcode_video
  if FFmpeg.new(self).transcode_video
    self.state = :encoded
  else
    self.state = :encoding_failed
  end
end
```

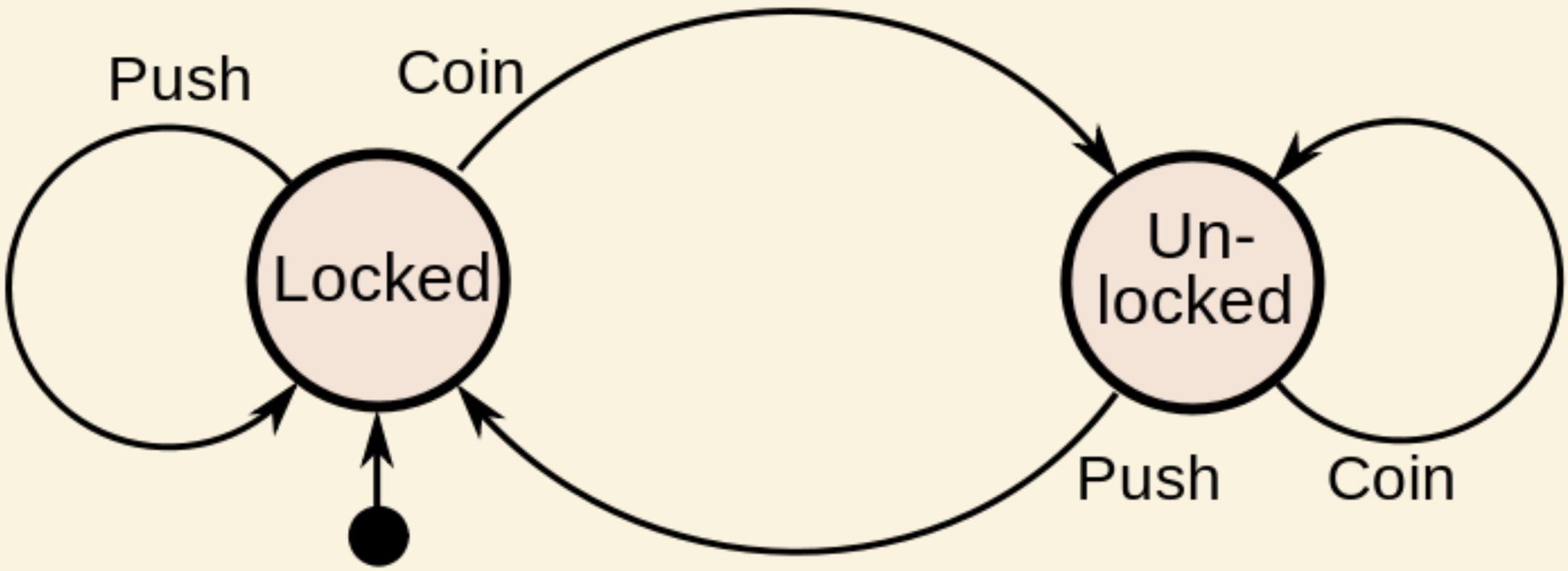
```ruby
def transcode_video
  if self.uploaded?
    if FFmpeg.new(self).transcode_video
      self.state = :encoded
    else
      self.state = :encoding_failed
    end
  end
end
```

State Machines!

A **finite-state machine** (FSM) or finite-state automaton (plural: automata), or simply a **state machine**, is a mathematical model of computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of states. The machine is in only one state at a time; the state it is in at any given time is called the current state.

https://en.wikipedia.org/wiki/Finite-state_machine

https://en.wikipedia.org/wiki/Turnstile



https://en.wikipedia.org/wiki/Finite-state_machine

```
gem "transitions", require: [
        "transitions",
        „active_model/transitions"
]
```

```ruby
class Turnstile < ActiveRecord::Base
  include ActiveModel::Transitions

  state_machine initial: :locked do
    state :locked
    state :unlocked
  end

end
```

```
state_machine initial: :locked do
  state :locked
  state :unlocked

  event :insert_coin do
    transitions from: :locked,   to: :unlocked
    transitions from: :unlocked, to: :unlocked
  end

end
```

```
t = Turnstile.new
t.state #= :locked
t.insert_coin
t.state #= :unlocked
t.insert_coin
t.state #= :unlocked
```

```
event :push do
  transitions from: :unlocked, to: :locked
  transitions from: :locked,   to: :locked
end
```

```
t = Turnstile.new
t.state #= :locked
t.push
t.state #= :locked
t.insert_coin
t.state #= :unlocked
t.push
t.state #= :locked
```

```ruby
class Turnstile < ActiveRecord::Base
  include ActiveModel::Transitions

  state_machine initial: :locked do
    state :locked
    state :unlocked

    event :push do
      transitions from: :unlocked, to: :locked
      transitions from: :locked,   to: :locked
    end

    event :insert_coin do
      transitions from: :locked,   to: :unlocked
      transitions from: :unlocked, to: :unlocked
    end
  end

end
```

Hooks and Guards

```
event :sell_out, success: :reorder do
  transitions from: :available,
              to:   :out_of_stock
end
```

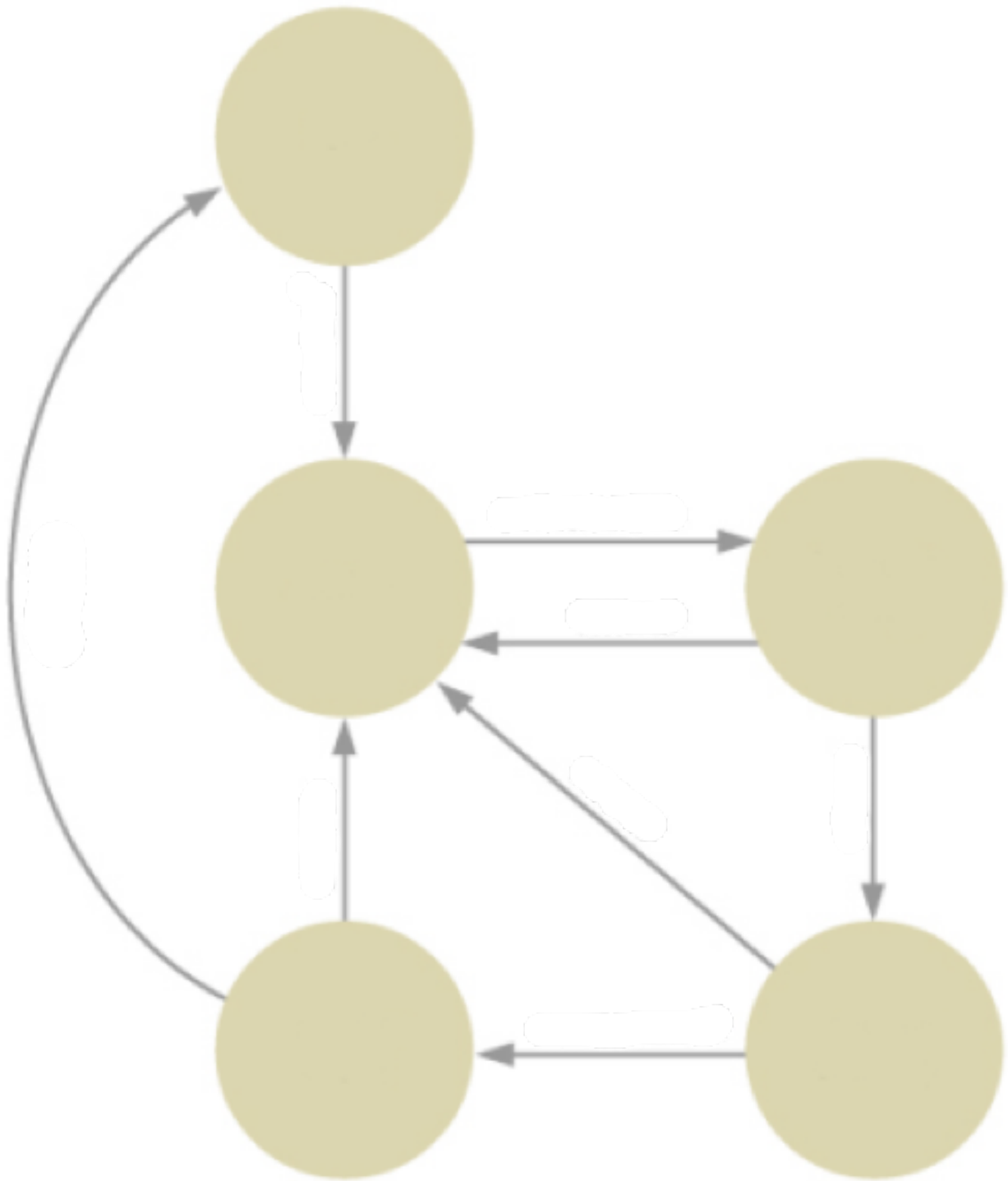
```
state :out_of_stock, exit: :inform_users
state :discontinued, enter: lambda do |product|
  product.cancel_orders
end
```

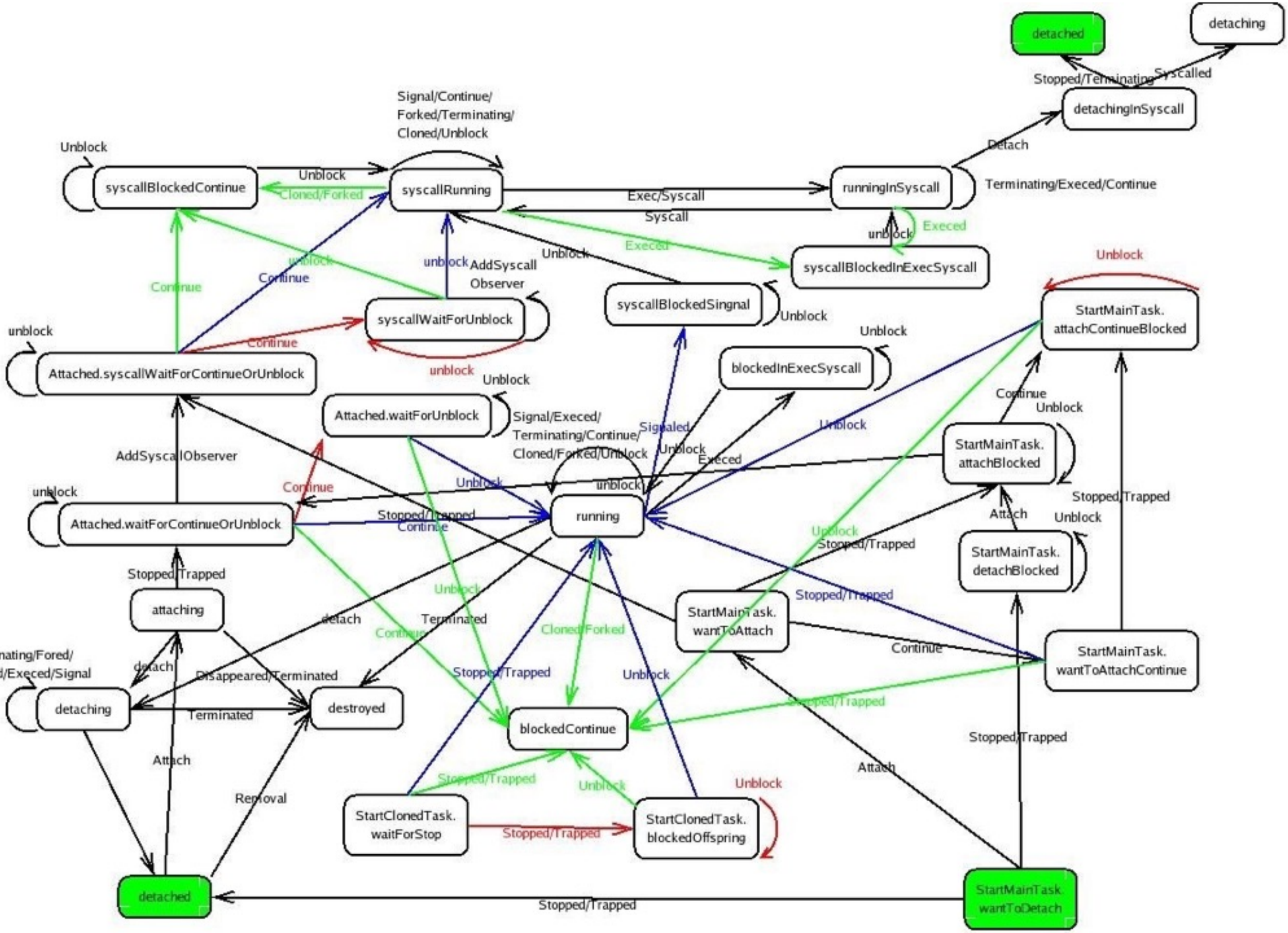
```
event :restock do
  transitions from:  :out_of_stock,
              to:    :available,
              guard: lambda do |product|
                product.in_stock > 0
              end
end
```

```ruby
class Order < ActiveRecord::Base
  include ActiveModel::Transitions

  state_machine initial: :init do
    state :init
    state :billed
    state :billing_failed
  end

end
```

```
event :bill_customer do
  transitions from: :init,
              to:   :billed,
              guard: lambda do |customer|
    customer.credit_card?
  end

  transitions from: :init,
              to:   :billing_failed
  end
end
```

```
order.bill_customer
order.billed?
```



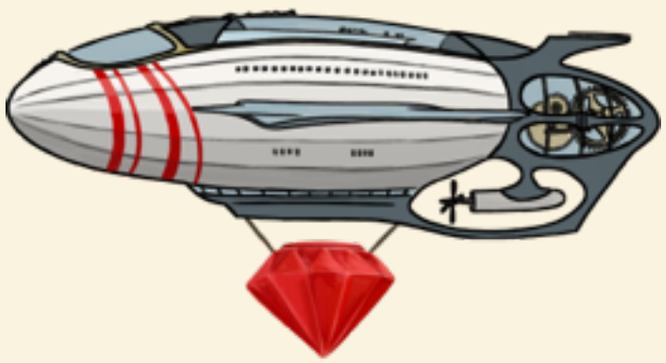

detached
detaching
Stopped/Terminating
Syscalled
detachingInSyscall
Detach
runningInSyscall
Terminating/Execed/Continue
Exec/Syscall
Syscall
Execed
unblock
Signal/Continue/
Forked/Terminating/
Cloned/Unblock
Unblock
syscallBlockedContinue
Cloned/Forked
syscallRunning
Unblock
syscallBlockedInExecSyscall
Unblock
Continue
unblock
Continue
unblock
AddSyscall
Observer
syscallWaitForUnblock
Continue
unblock
syscallBlockedSingnal
Unblock
Unblock
StartMainTask.
attachContinueBlocked
unblock
Attached.syscallWaitForContinueOrUnblock
Unblock
blockedInExecSyscall
Unblock
Attached.waitForUnblock
Signal/Execed/
Terminating/Continue/
Cloned/Forked/Unblock
Signaled
Unblock
Unblock
Execed
Continue
Unblock
StartMainTask.
attachBlocked
AddSyscallObserver
Continue
Unblock
unblock
Stopped/Trapped
Continue
Attached.waitForContinueOrUnblock
Stopped/Trapped
running
Attach
Unblock
Stopped/Trapped
Stopped/Trapped
StartMainTask.
detachBlocked
Stopped/Trapped
attaching
detach
Terminated
Unblock
Continue
Cloned/Forked
StartMainTask.
wantToAttach
Stopped/Trapped
Continue
StartMainTask.
wantToAttachContinue
Terminating/Fored/
Cloned/Execed/Signal
detach
Disappeared/Terminated
detaching
destroyed
Terminated
Stopped/Trapped
Unblock
Stopped/Trapped
blockedContinue
Stopped/Trapped
Attach
Attach
Removal
Unblock
StartClonedTask.
waitForStop
Stopped/Trapped
StartClonedTask.
blockedOffspring
Unblock
detached
Stopped/Trapped
StartMainTask.
wantToDetach

Recursions => :(

GEMs

bit.ly/statemachines

Bodo Tasche
@bitboxer



@bitcrowd