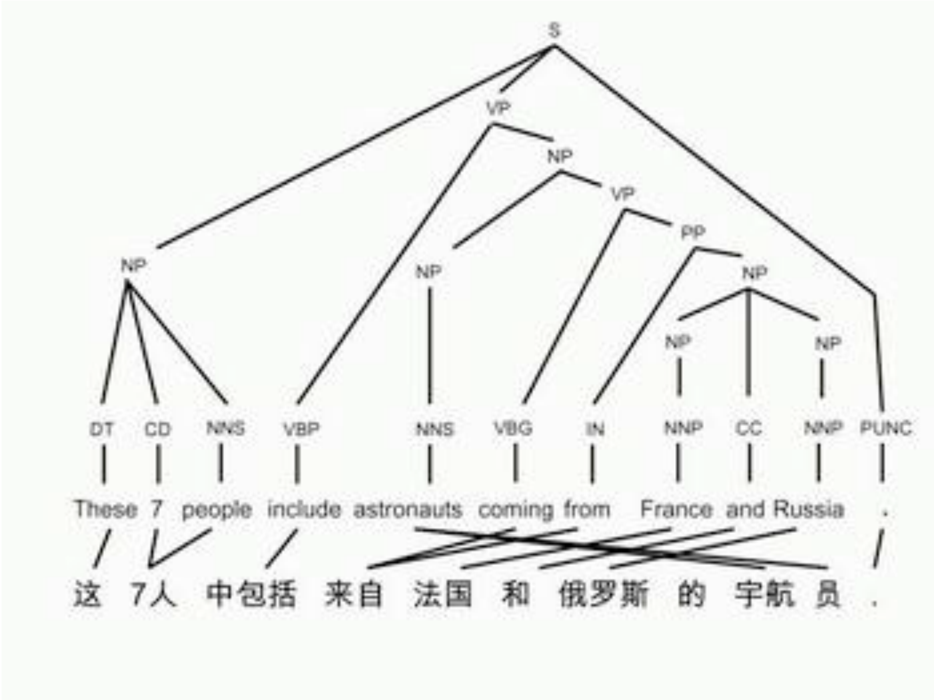


Object Grammars

Compositional & Bidirectional Mapping Between Text and Graphs

Tijs van der Storm, William R. Cook, Alex Loh

CWI Centrum Wiskunde & Informatica

Universiteit van Amsterdam

INRIA Lille - Nord-Europe

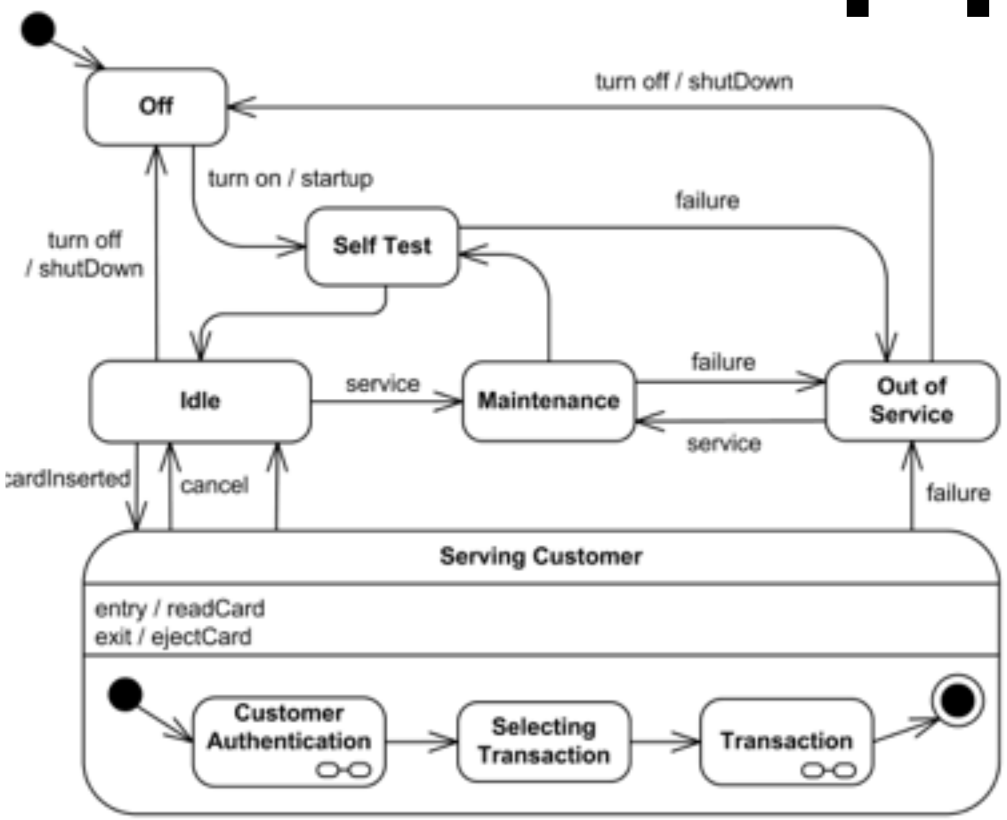
The University of Texas at Austin

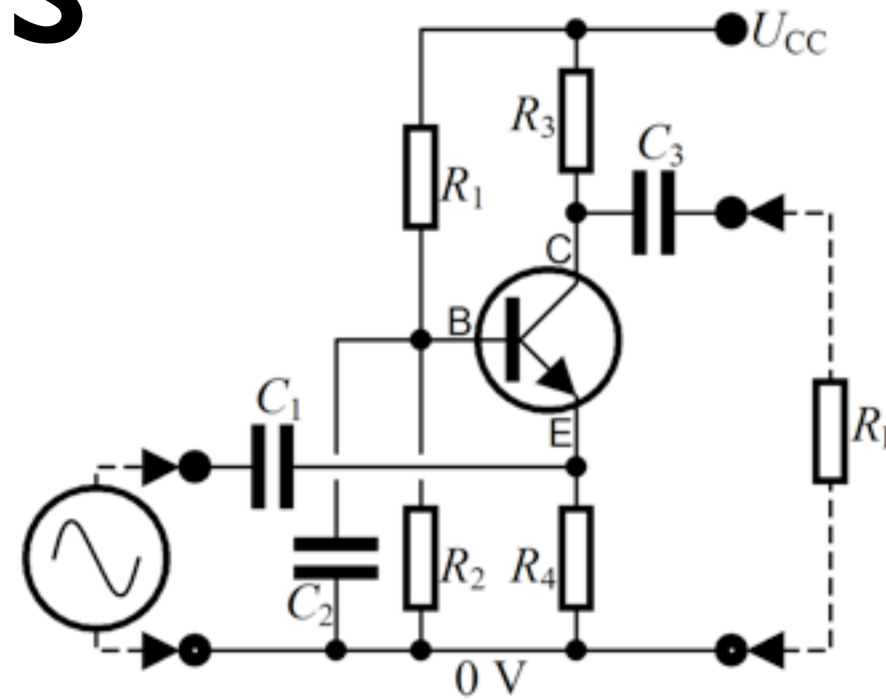
Ensō 円相

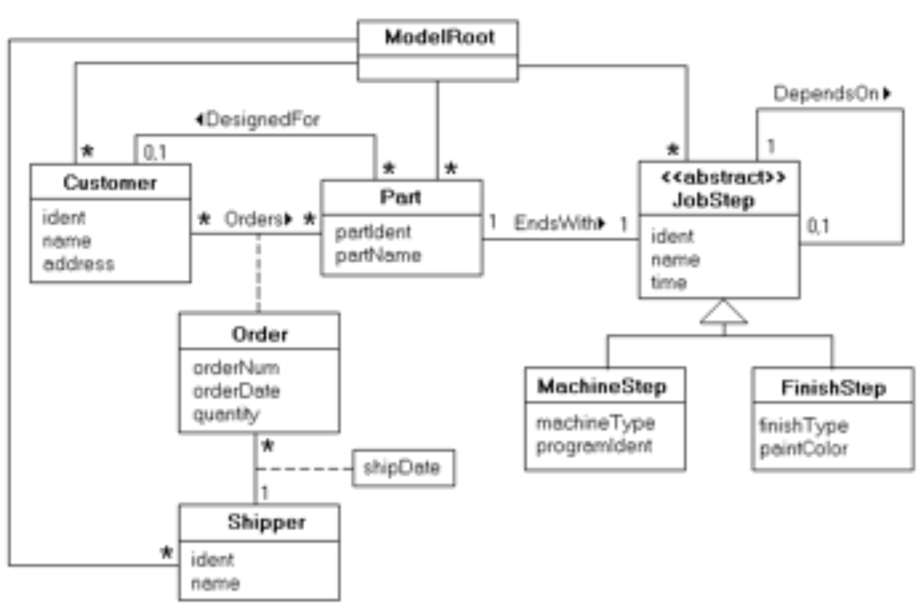
Don't Design Your Programs, Program Your Designs

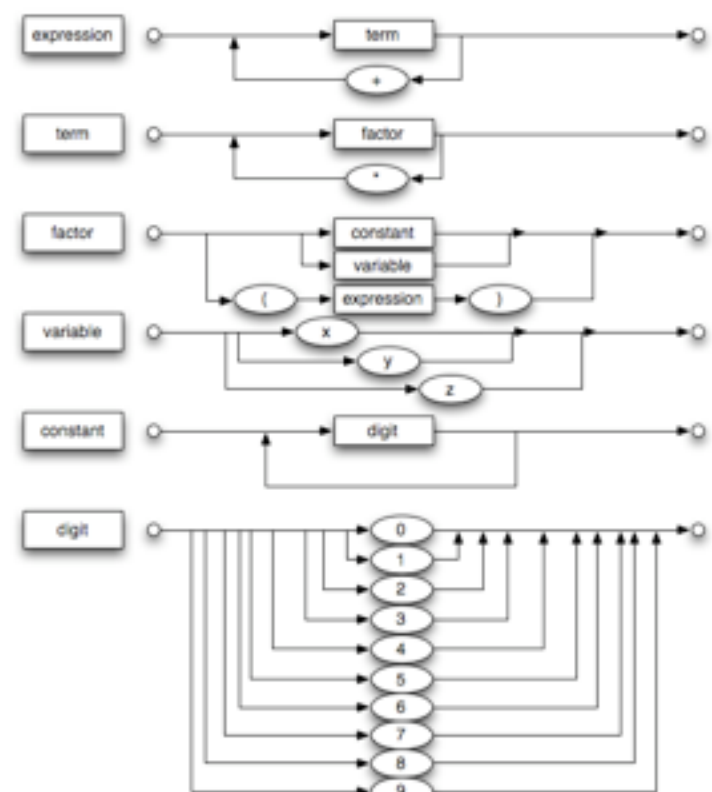
http://www.enso-lang.org/

Models


Off
turn off / shutDown
turn on / startup
Self Test
failure
turn off
/ shutDown
Idle
service
Maintenance
failure
Out of Service
service
cardInserted
cancel
failure
Serving Customer
entry / readCard
exit / ejectCard
Customer Authentication
Selecting Transaction
Transaction

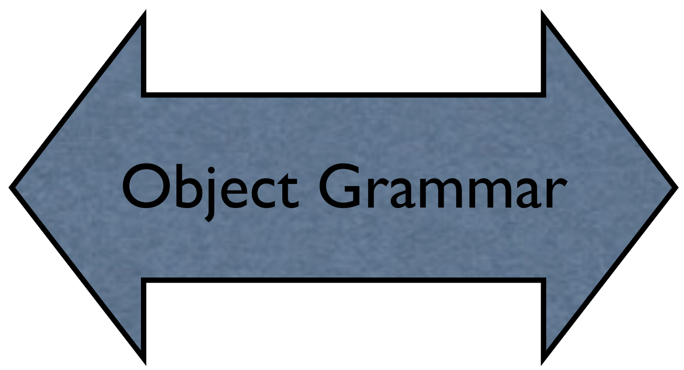

U_{CC}
R_3
C_3
R_1
C
B
E
C_1
R_L
R_2
R_4
C_2
0 V

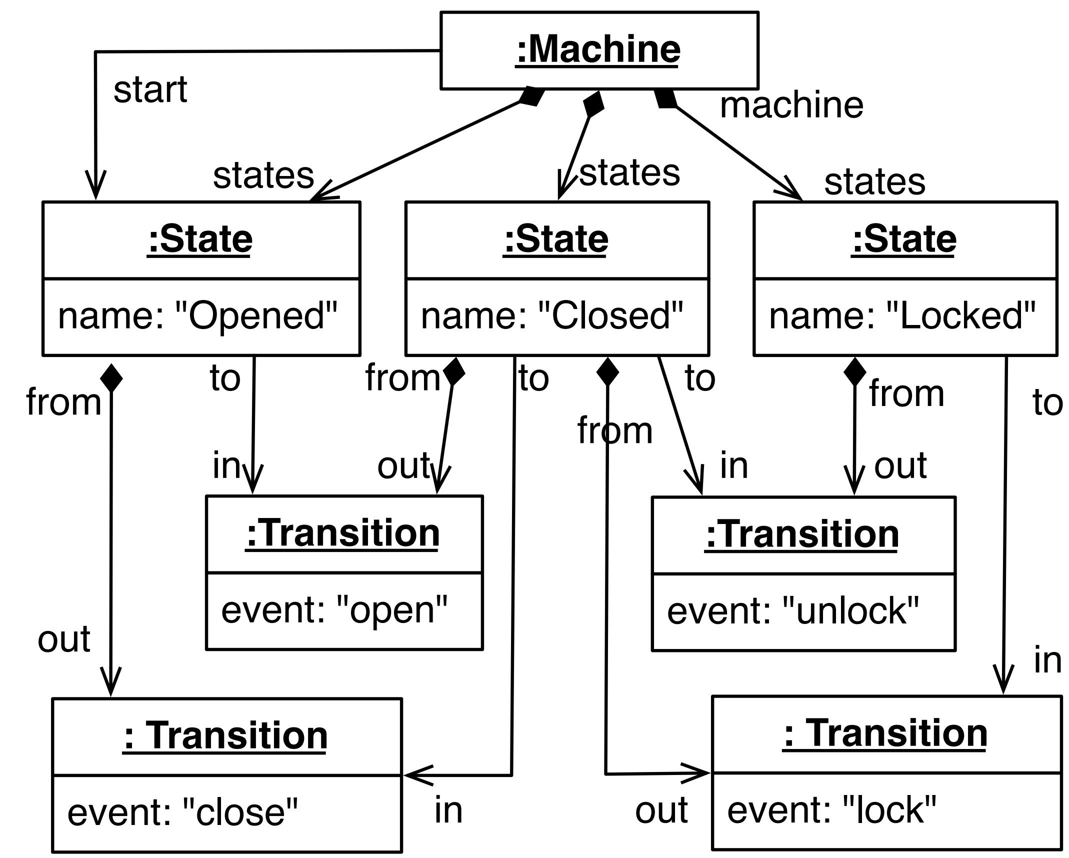

ModelRoot
DesignedFor
DependsOn
Customer
ident
name
address
Orders
Part
partIdent
partName
EndsWith
<<abstract>>
JobStep
ident
name
time
Order
orderNum
orderDate
quantity
shipDate
MachineStep
machineType
programIdent
FinishStep
finishType
paintColor
Shipper
ident
name

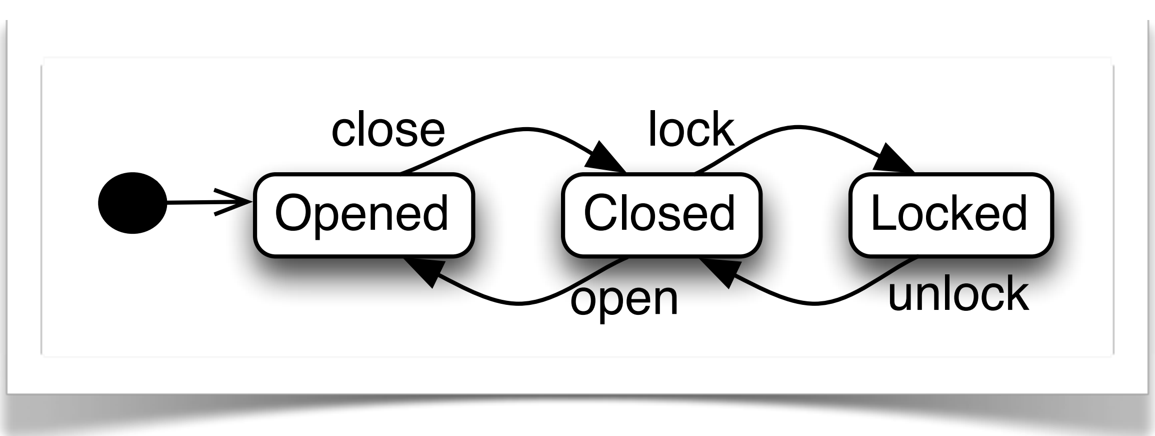

expression
term
+
term
factor
*
factor
constant
variable
(
expression
)
variable
x
y
z
constant
digit
digit
0
1
2
3
4
5
6
7
8
9

Text to objects and back

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```







Object Grammars

- Interleave grammar with *data binding*
  - object construction
  - field assignment
  - predicates
- Bind to *paths* in to create cross references
- Formatting hints to guide pretty printing

Points

```
P ::= [Point] "(" x:int "," y:int ")"
```

Points

Constructor

```
P ::= [Point] "(" x:int "," y:int ")"
```

Points

Constructor

Field binding

```
P ::= [Point] "(" x:int "," y:int ")"
```

Points

Constructor

Field binding

Built-in primitives

```
P ::= [Point] "(" x:int "," y:int ")"
```

Points

Constructor

Field binding

Built-in primitives

```
P ::= [Point] "(" x:int "," y:int ")"
```

The *schema*

```
class Point   x: int   y: int
```

Points

Constructor

Field binding

Built-in primitives

```
P ::= [Point] "(" x:int "," y:int ")"
```

The *schema*

```
class Point   x: int   y: int
```

Points

Constructor

Field binding

Built-in primitives

```
P ::= [Point] "(" x:int "," y:int ")"
```

The *schema*

```
class Point   x: int  y: int
```

Points

Constructor

Field binding

Built-in primitives

```
P ::= [Point] "(" x:int "," y:int ")"
```

The *schema*

```
class Point   x: int  y: int
```

Expressions

```
Exp ::= [Binary] lhs:Exp op:"+" rhs:Exp
     |  [Binary] lhs:Exp op:"*" rhs:Exp
     |  [Const] value:int
     |  "(" Exp ")"
```

```
class Exp
class Binary < Exp
  op: str
  lhs: Exp
  rhs: Exp
class Const < Exp
  value: int
```

Expressions

Both + and * become *Binary* objects

```
Exp ::= [Binary] lhs:Exp op:"+" rhs:Exp
     |  [Binary] lhs:Exp op:"*" rhs:Exp
     |  [Const] value:int
     |  "(" Exp ")"
```

```
class Exp
class Binary < Exp
  op: str
  lhs: Exp
  rhs: Exp
class Const < Exp
  value: int
```

Expressions

Both + and * become *Binary* objects

```
Exp ::= [Binary] lhs:Exp op:"+" rhs:Exp
     |  [Binary] lhs:Exp op:"*" rhs:Exp
     |  [Const] value:int
     |  "(" Exp ")"
```

Parentheses don't introduce objects

```
class Exp
class Binary < Exp
  op: str
  lhs: Exp
  rhs: Exp
class Const < Exp
  value: int
```

Expressions

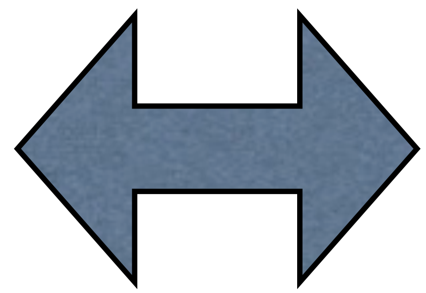
Refactored grammar for disambiguation

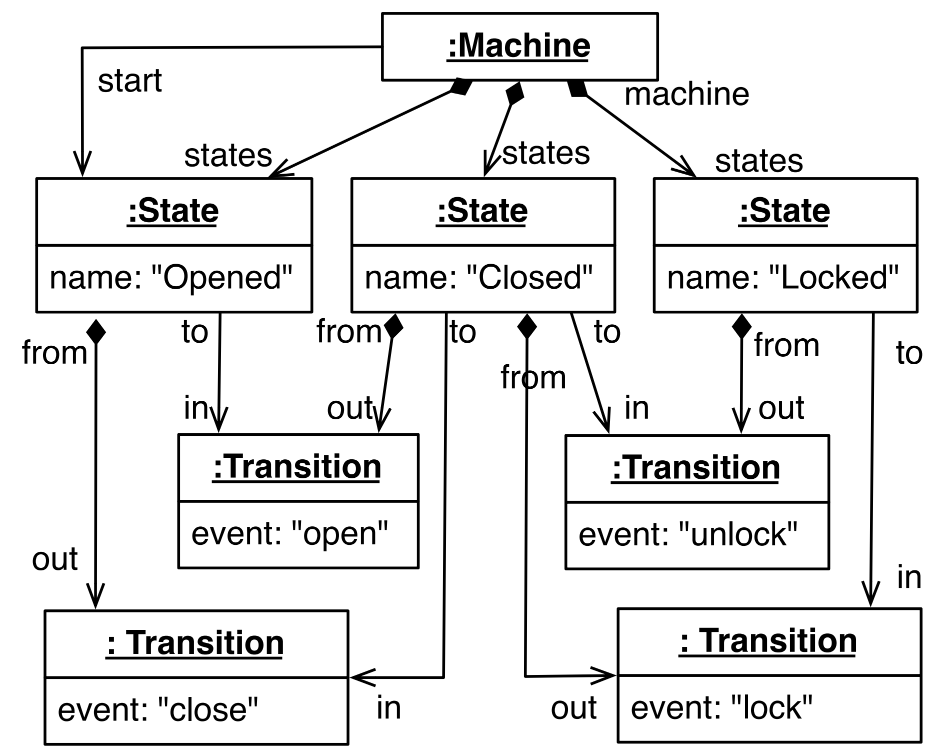
```
Term ::= [Binary] lhs:Term op:"+" rhs:Fact
      |  Fact
Fact ::= [Binary] lhs:Fact op:"*" rhs:Prim
      |  Prim
Prim ::= [Const] value:int
      |  "(" Term ")"
```

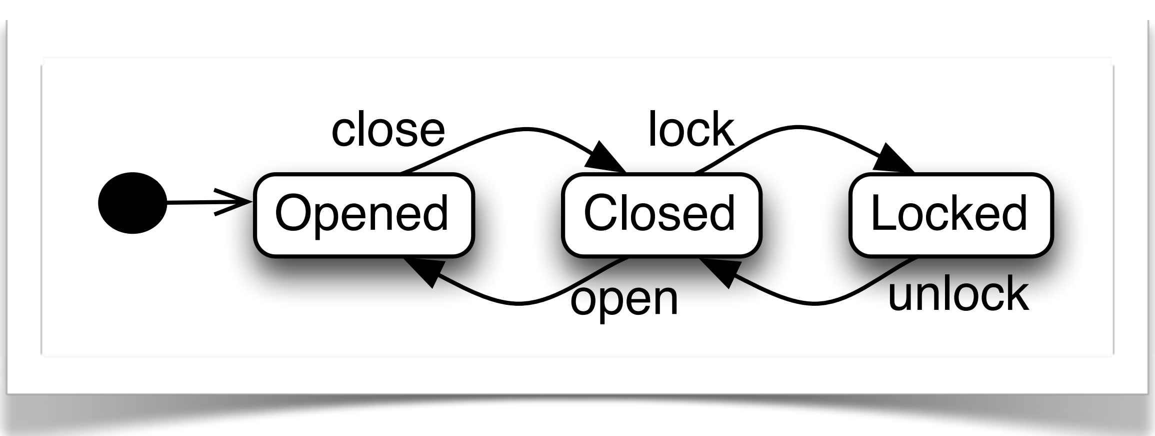
```
class Exp
class Binary < Exp
  op: str
  lhs: Exp
  rhs: Exp
class Const < Exp
  value: int
```

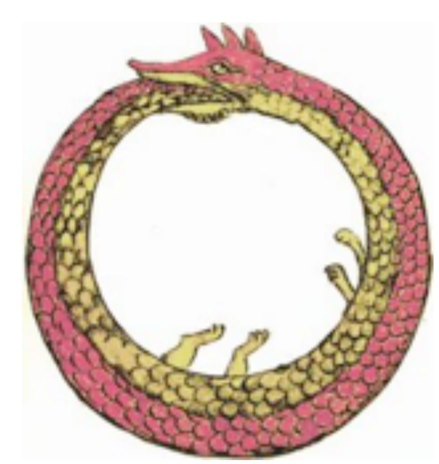
State machines

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```









The object grammar

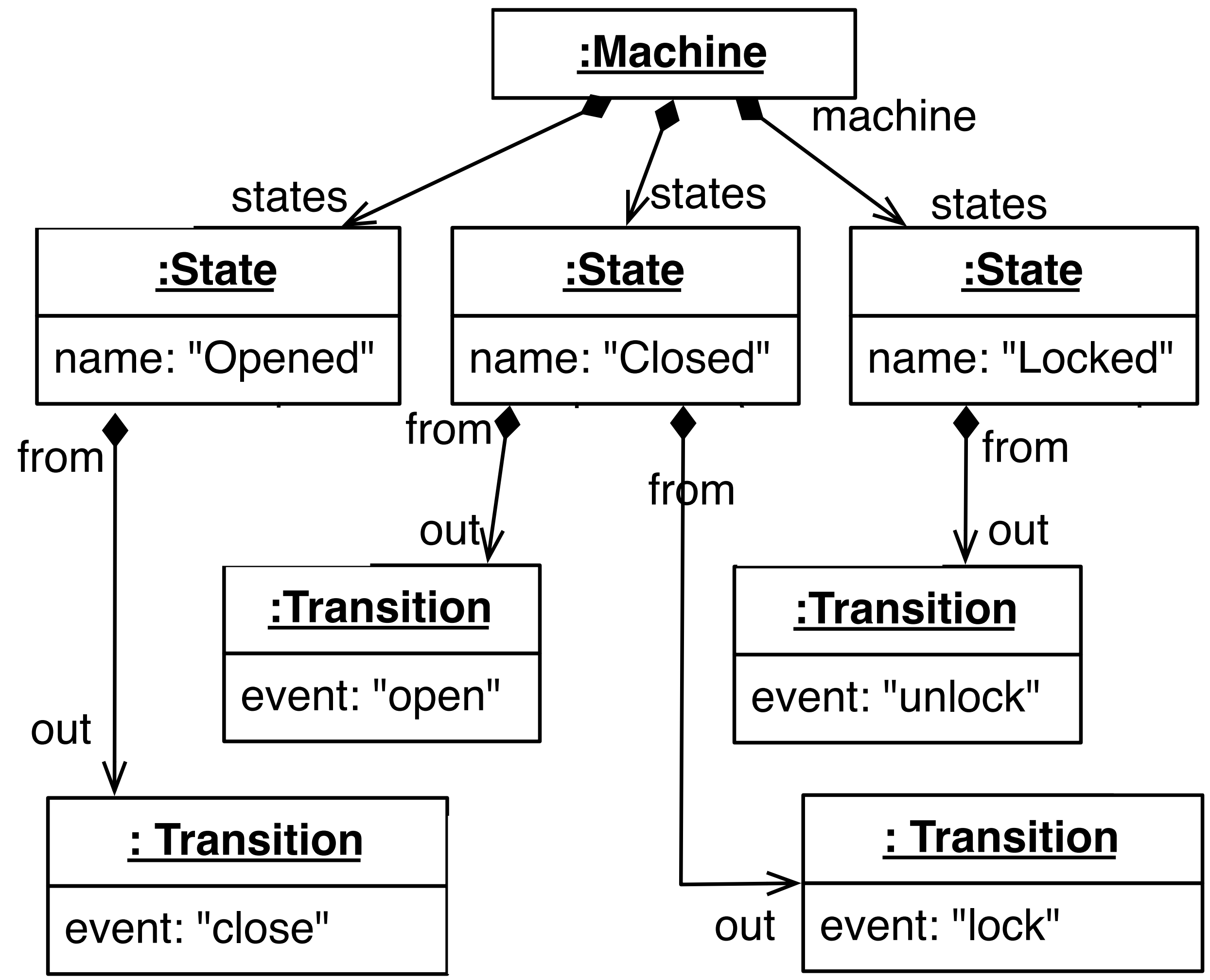
```
M ::= [Machine] "start"  start:</states[it]> states:S*

S ::= [State] "state" name:sym out:T*

T ::= [Transition] "on" event:sym "go" to:</states[it]>
```

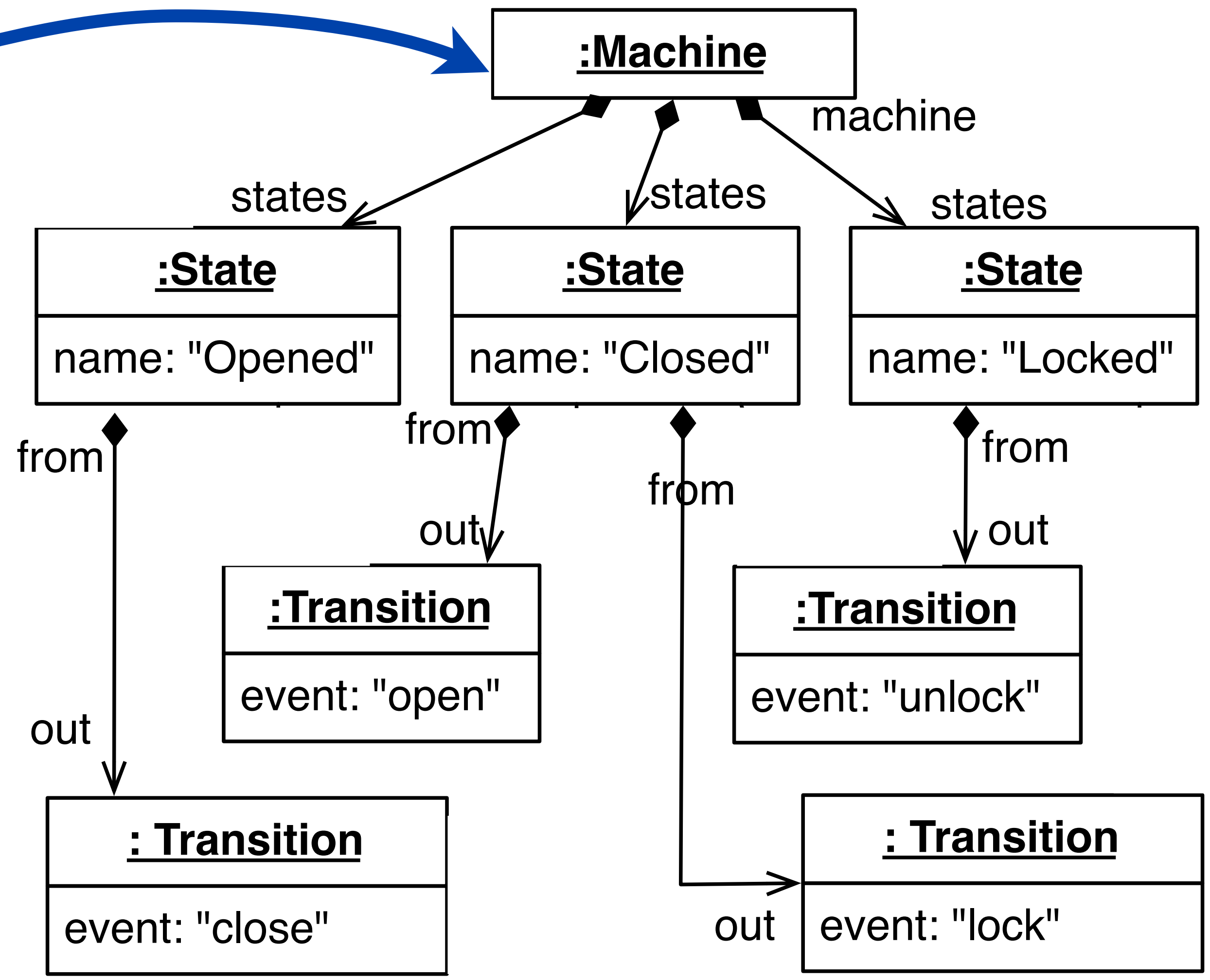
Creating the *spine*

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```



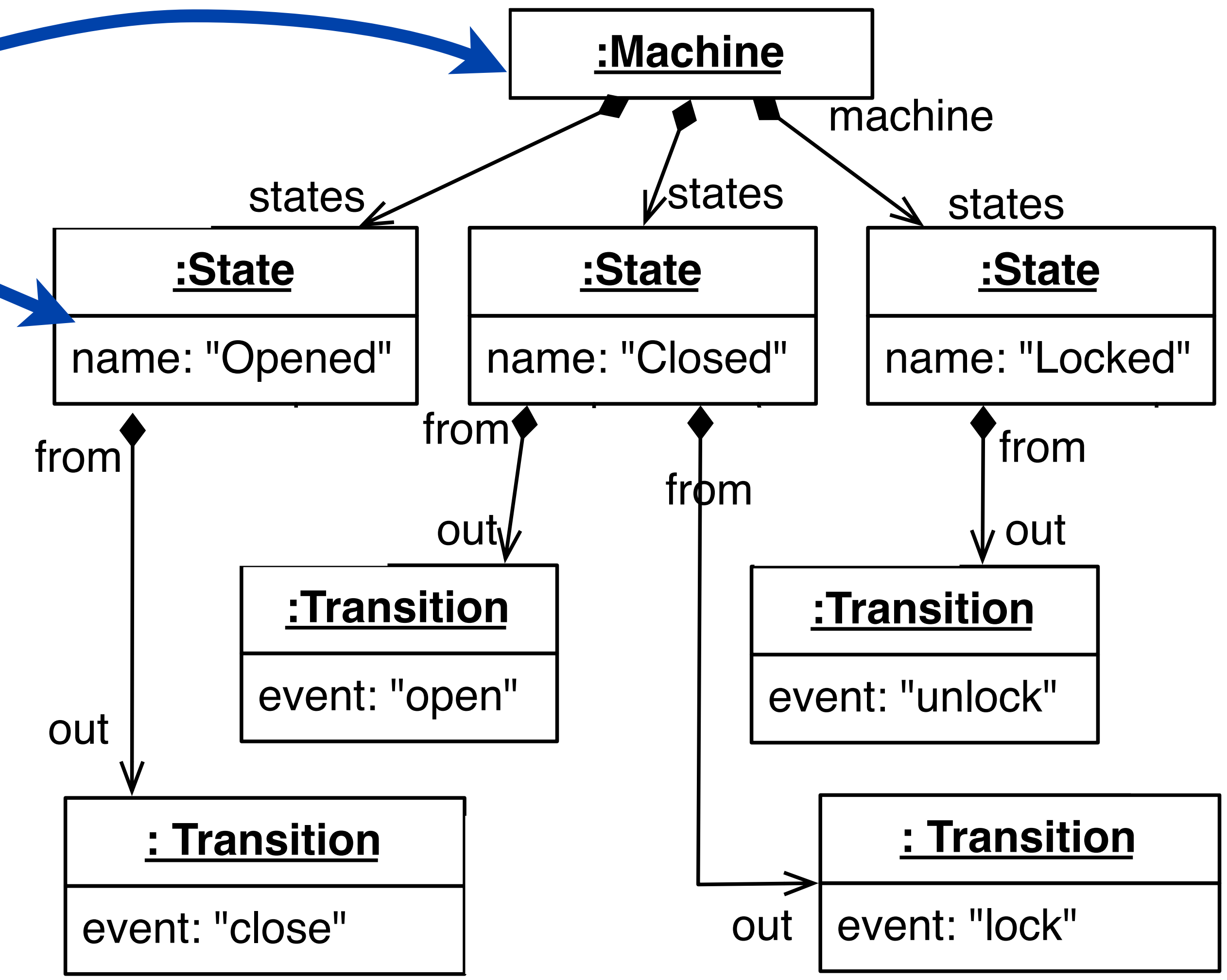
Creating the *spine*

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```



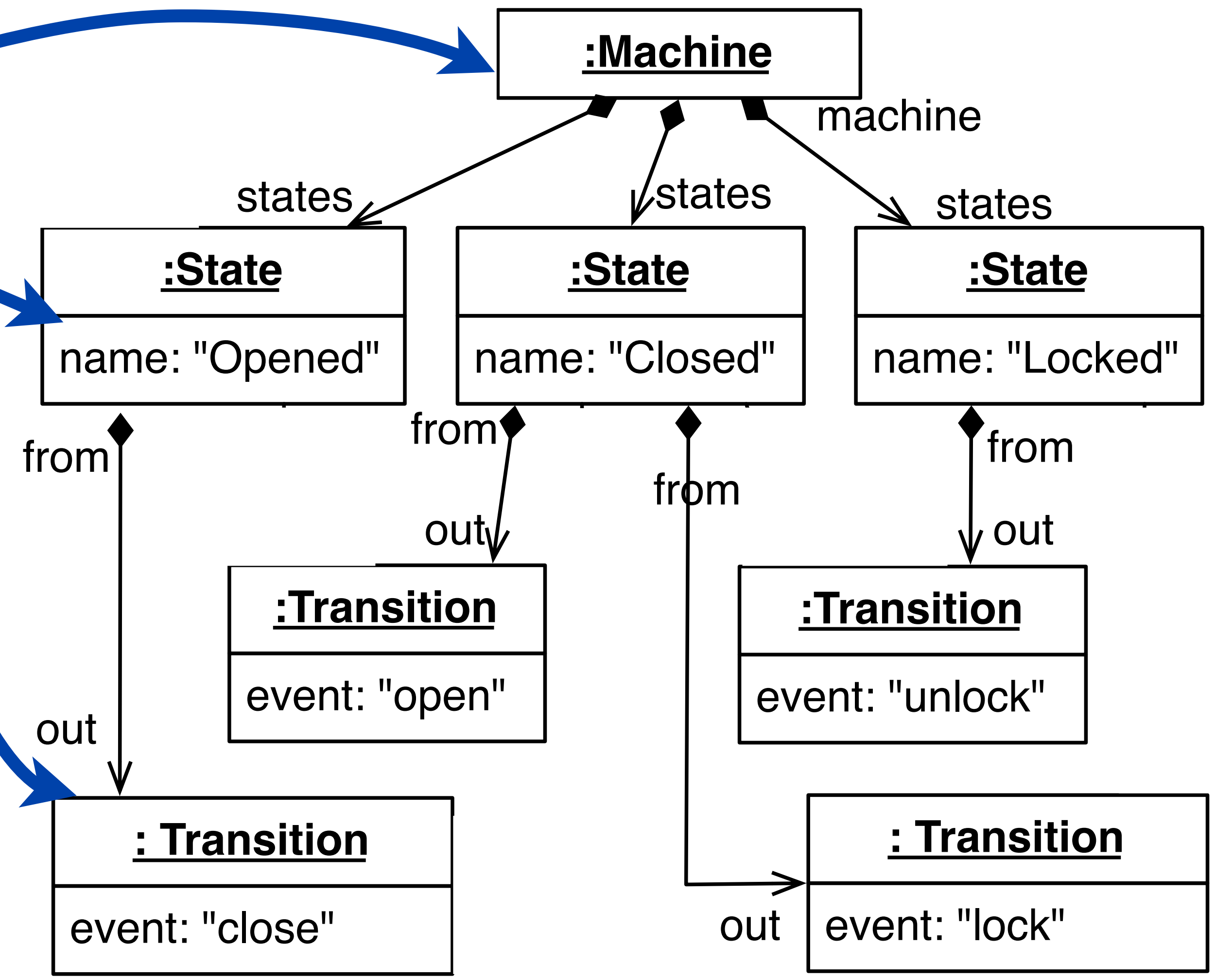
Creating the *spine*

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```



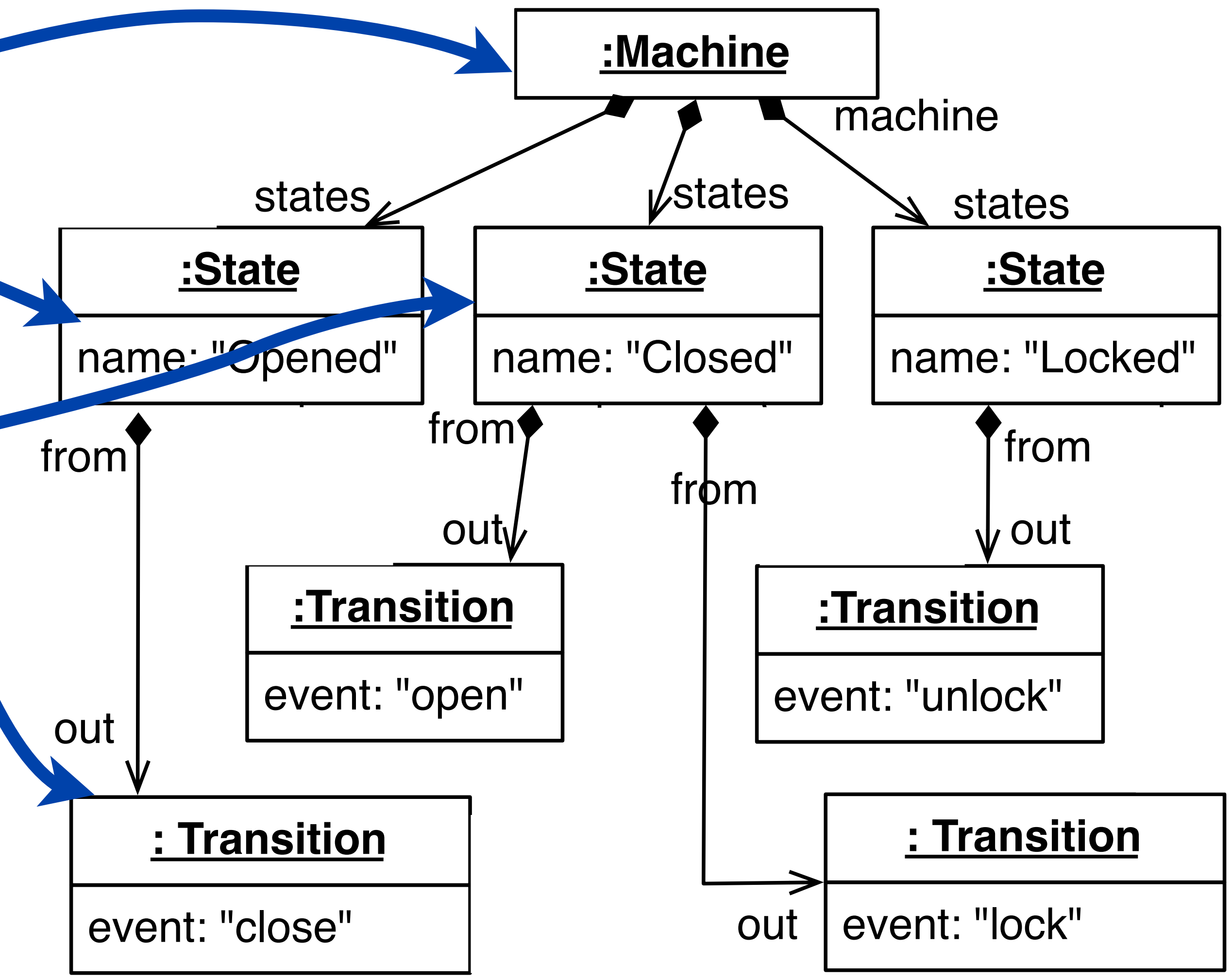
Creating the *spine*

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

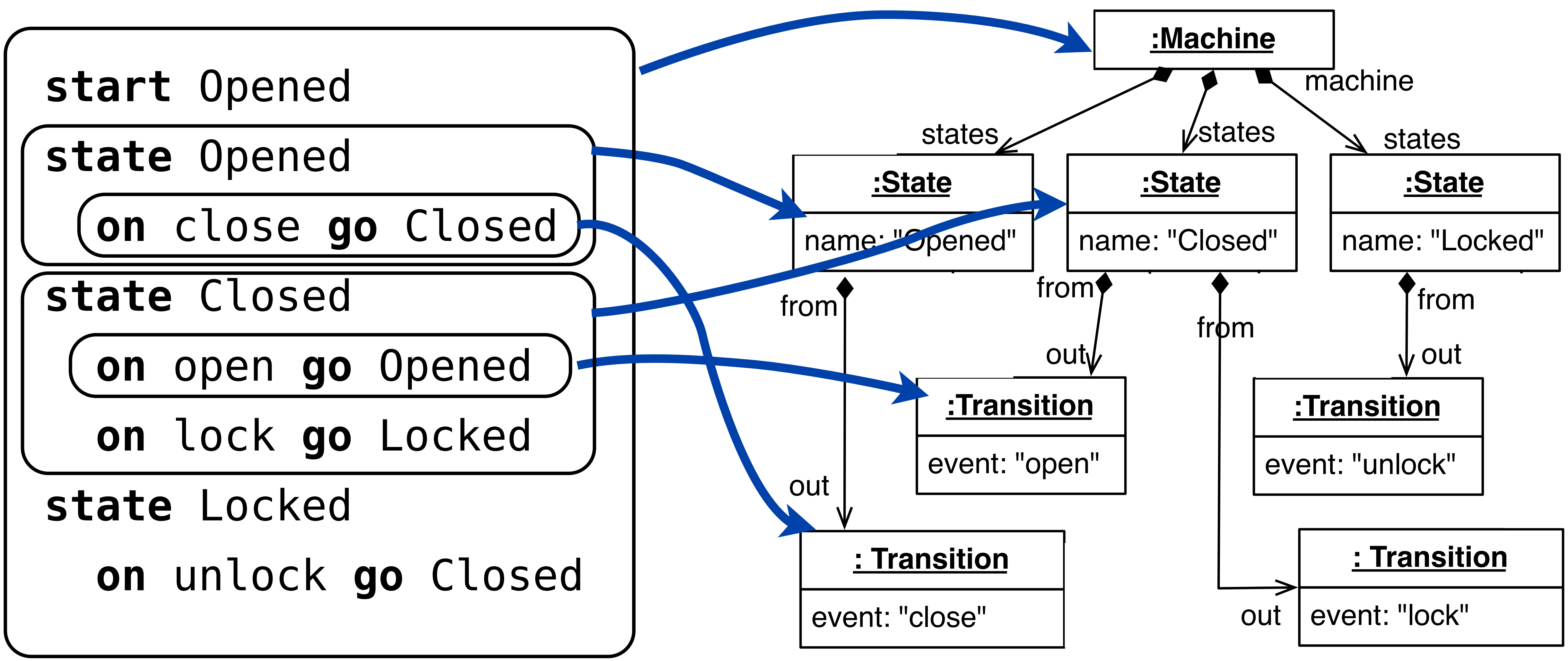


Creating the *spine*

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

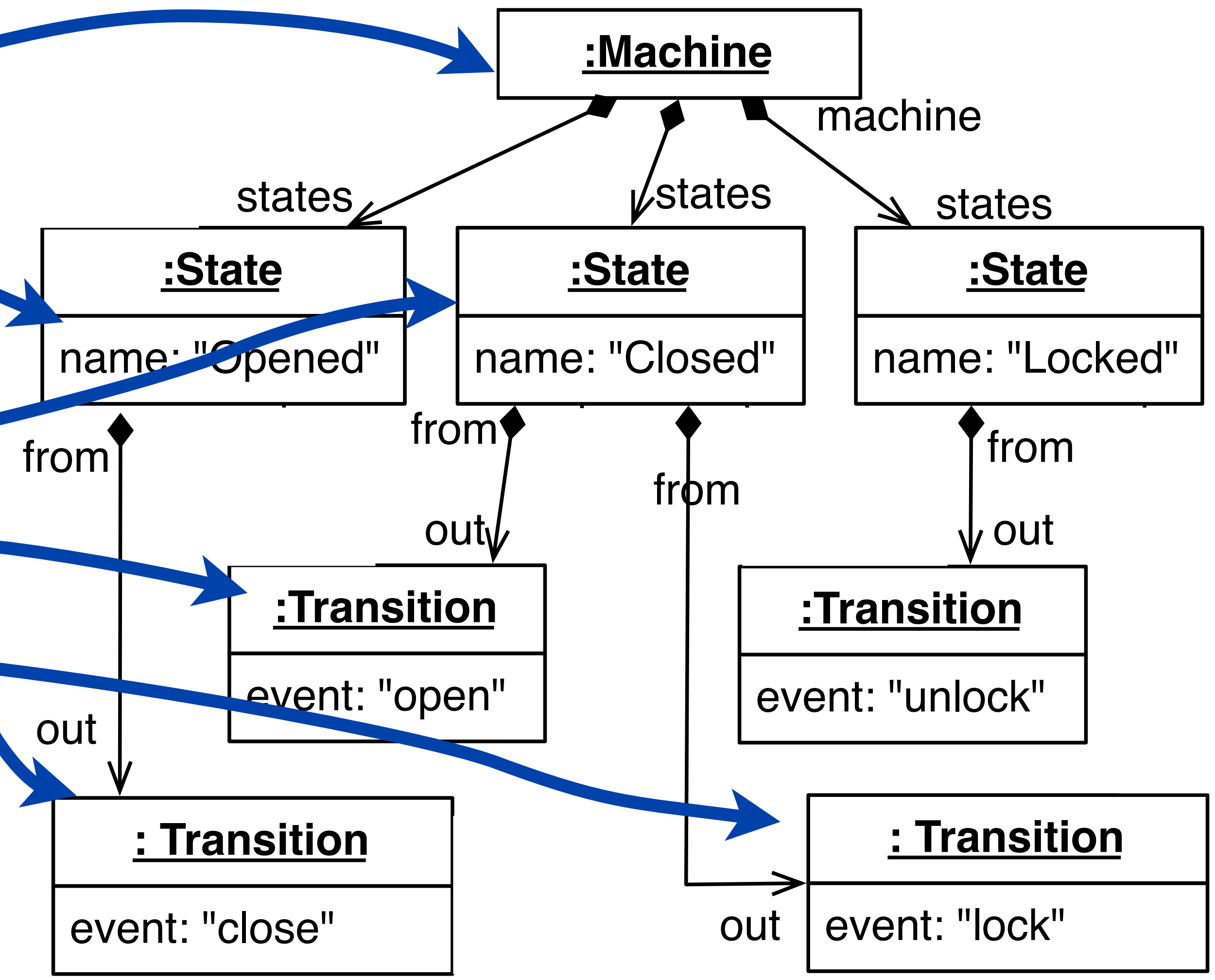


Creating the *spine*

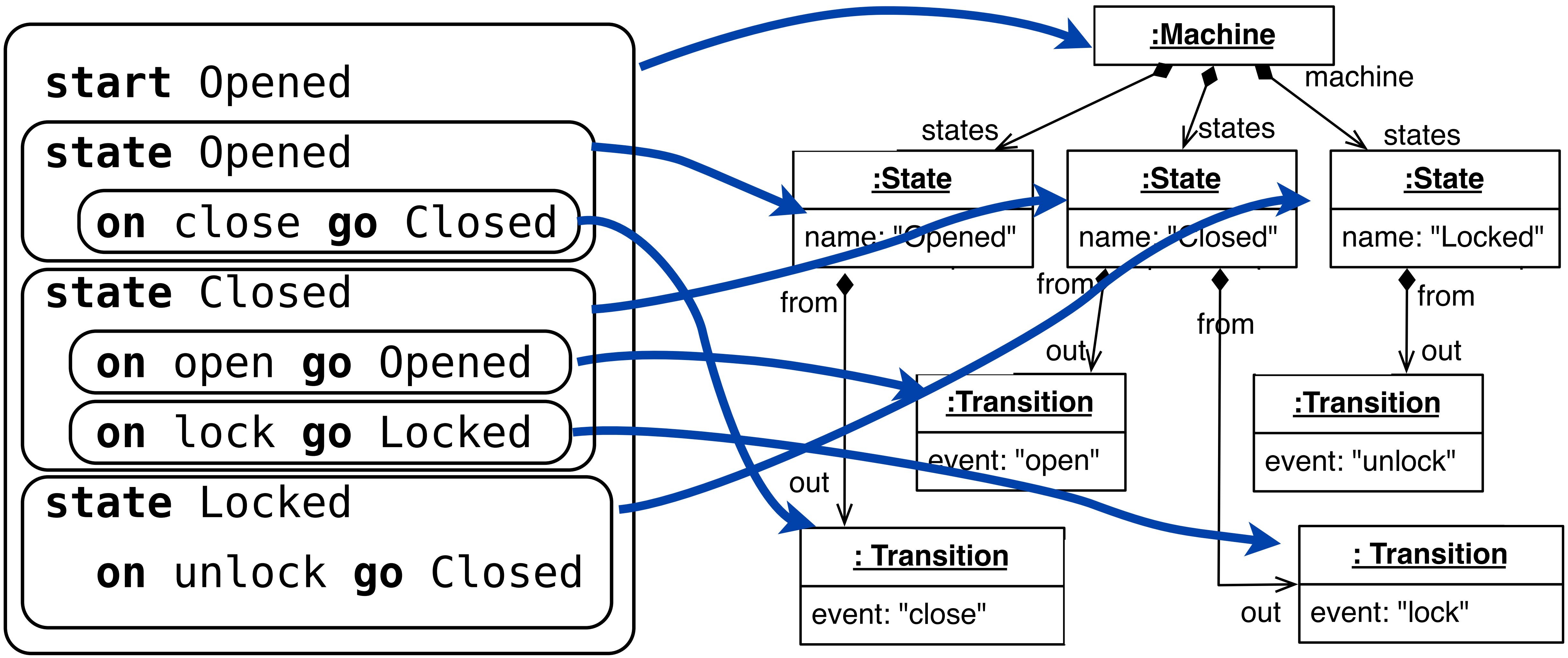


Creating the *spine*

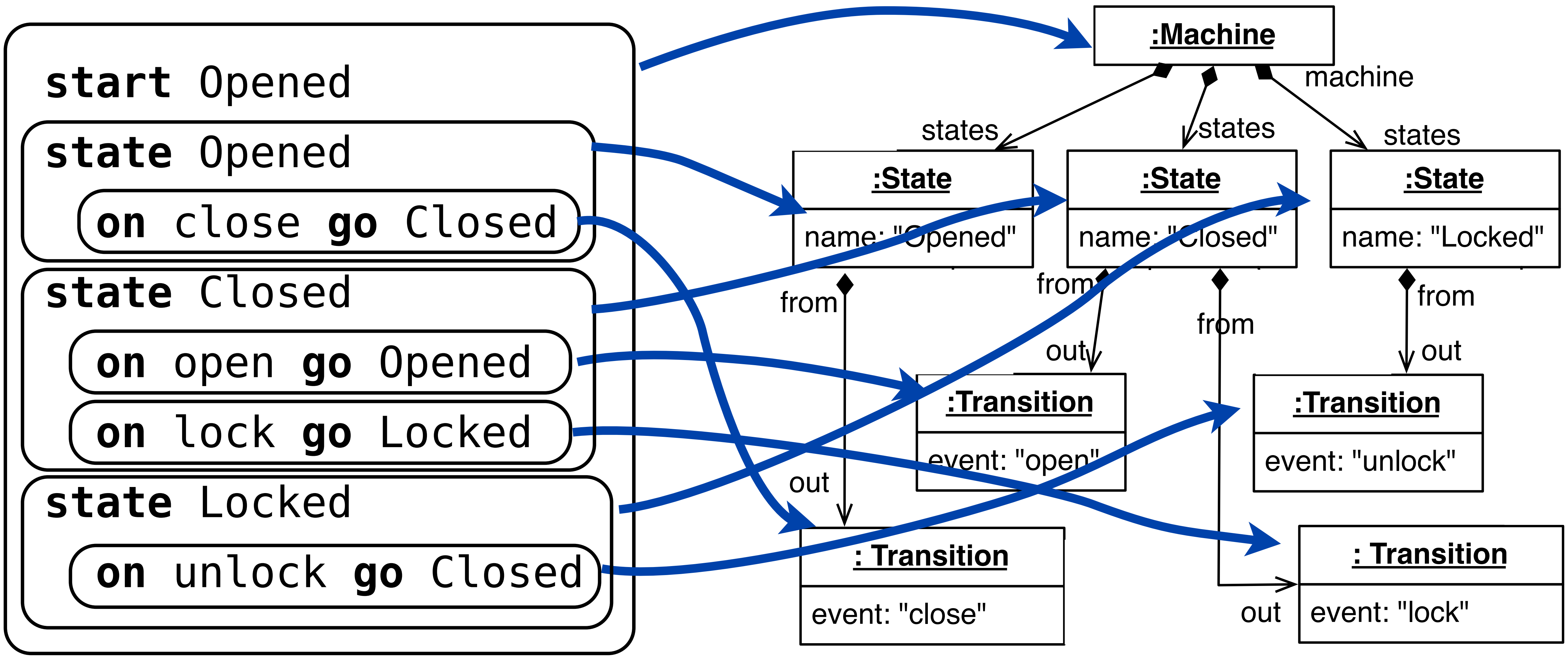
```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```



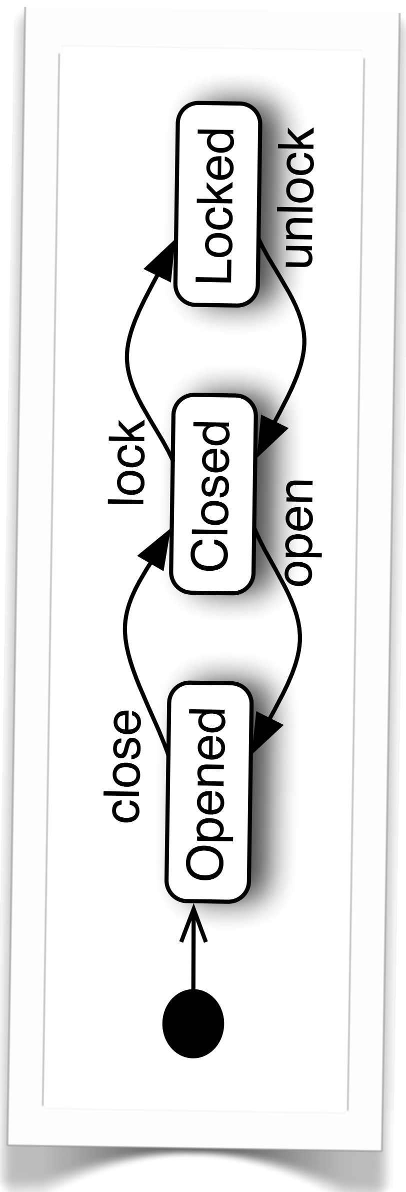
Creating the *spine*



Creating the *spine*

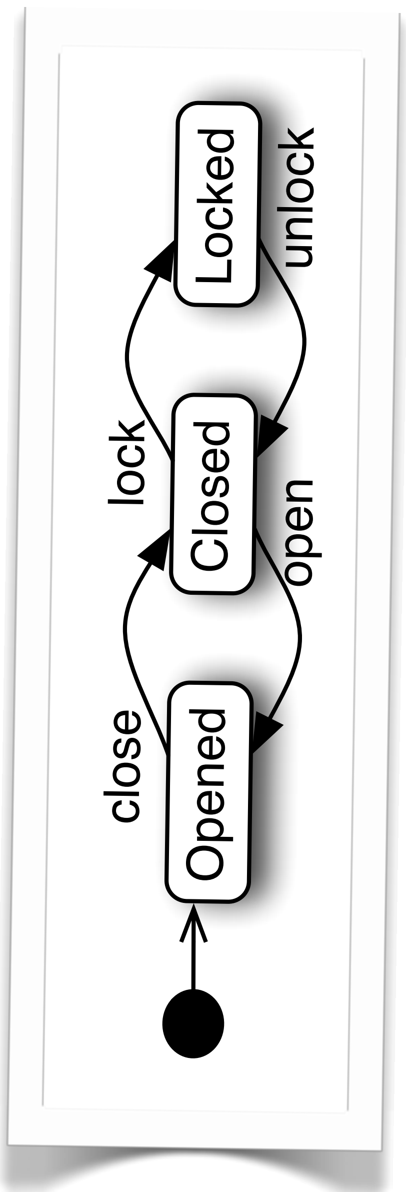


Cross links



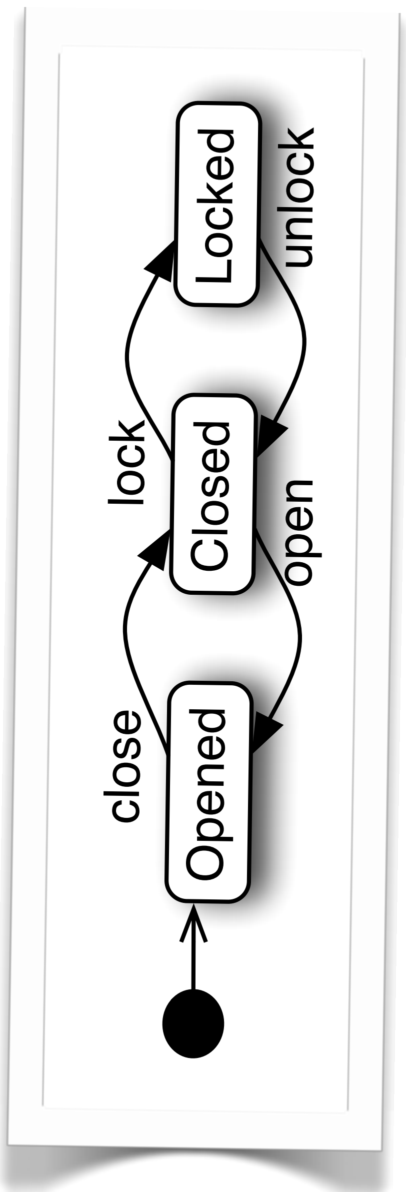

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

Cross links



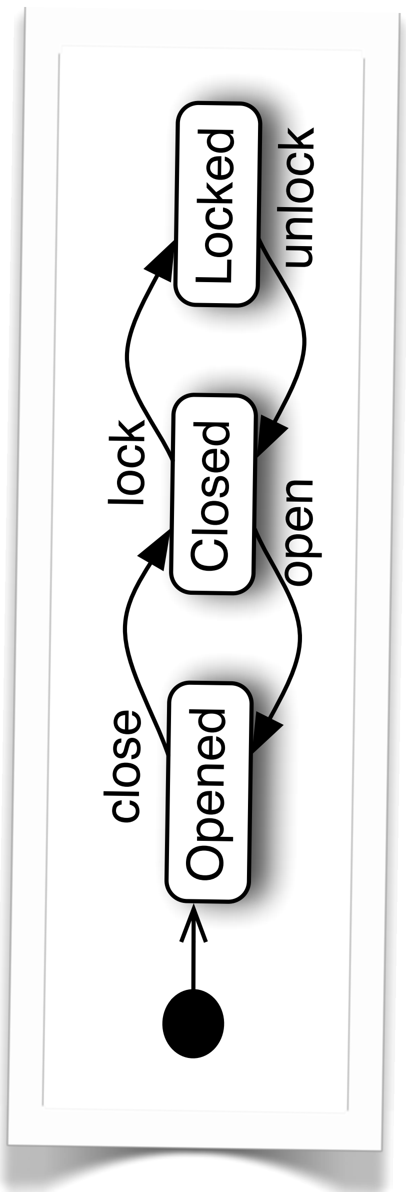

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

Cross links



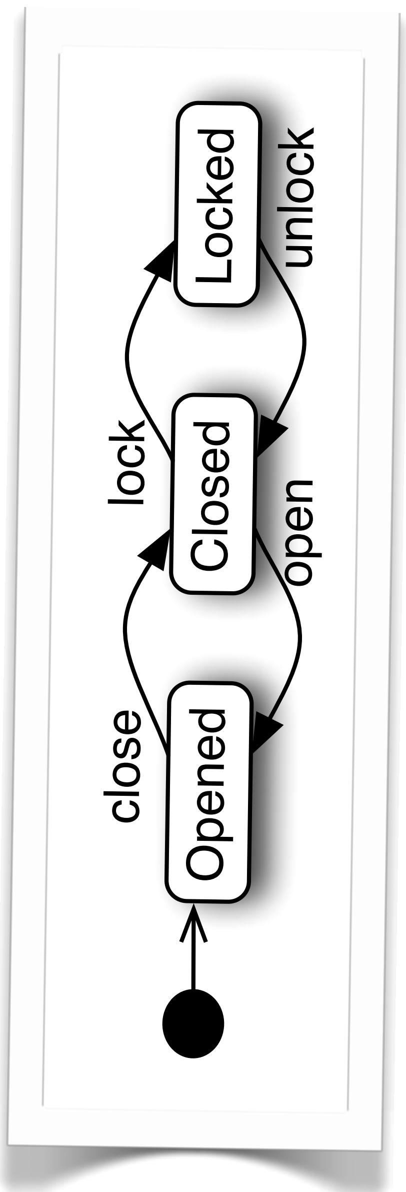

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

Cross links



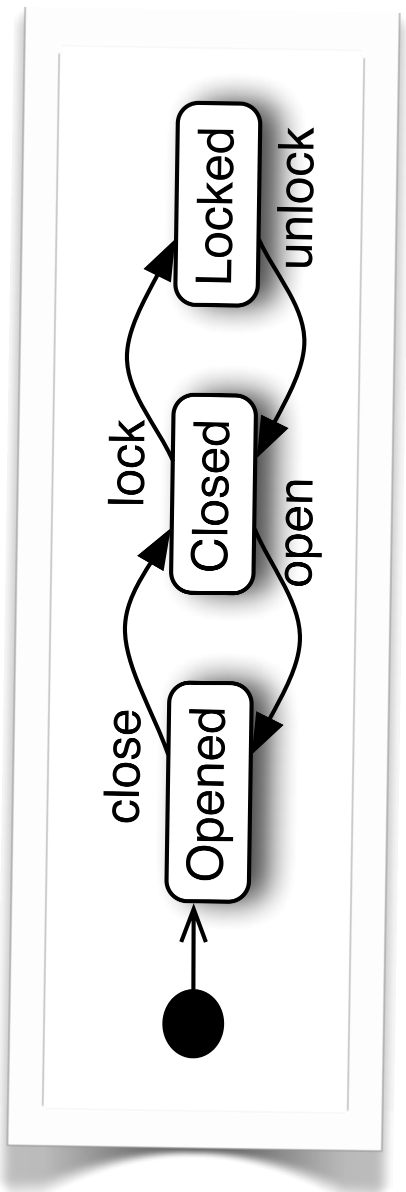

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

Cross links




```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

Cross links




```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

Object path to find the start state with name *it*

```
M ::= [Machine] "start" \start:</states[it]> states:S*

S ::= [State] "state" name:sym out:T*

T ::= [Transition] "on" event:sym "go" to:</states[it]>
```

Paths

- Navigate the resulting model along
  - Fields
  - Collections (keyed, positional)
- NB: model may not be finished yet
  - Paths may traverse *cross links* too
  - Iterative fix point

A path

start at the root

navigate into *states*

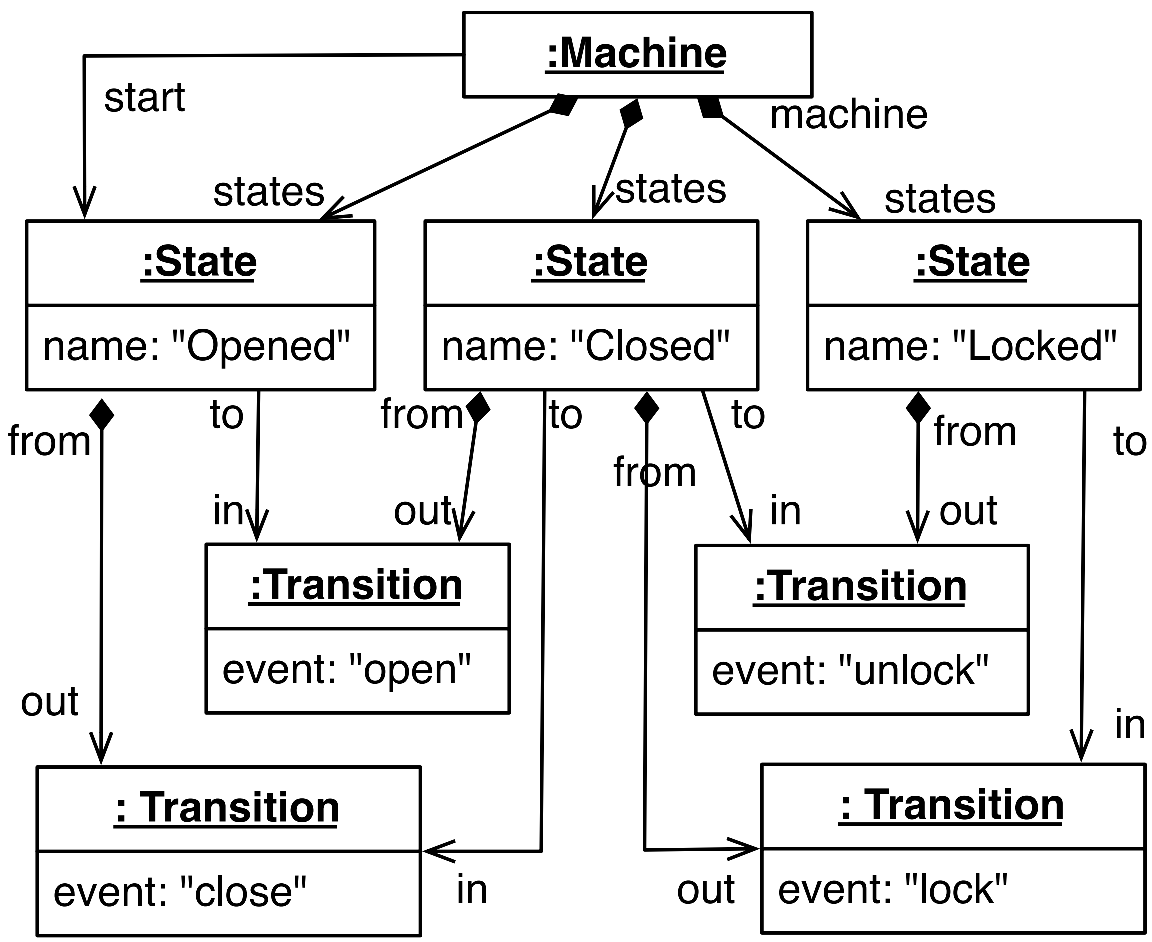
use the parsed identifier as key

```
/states[it]
```

Paths can also start at current object (.) or parent (..)

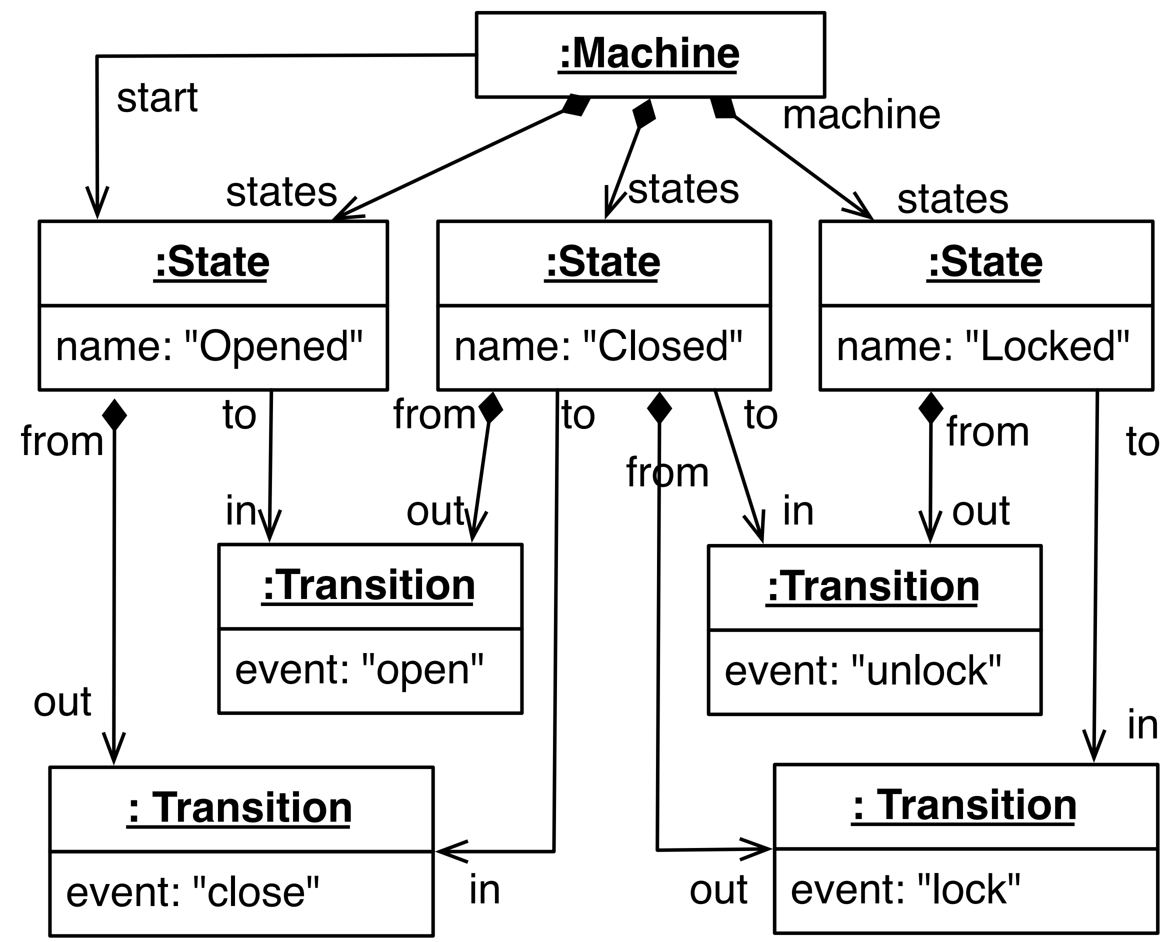
Creating cross links

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```



Creating cross links

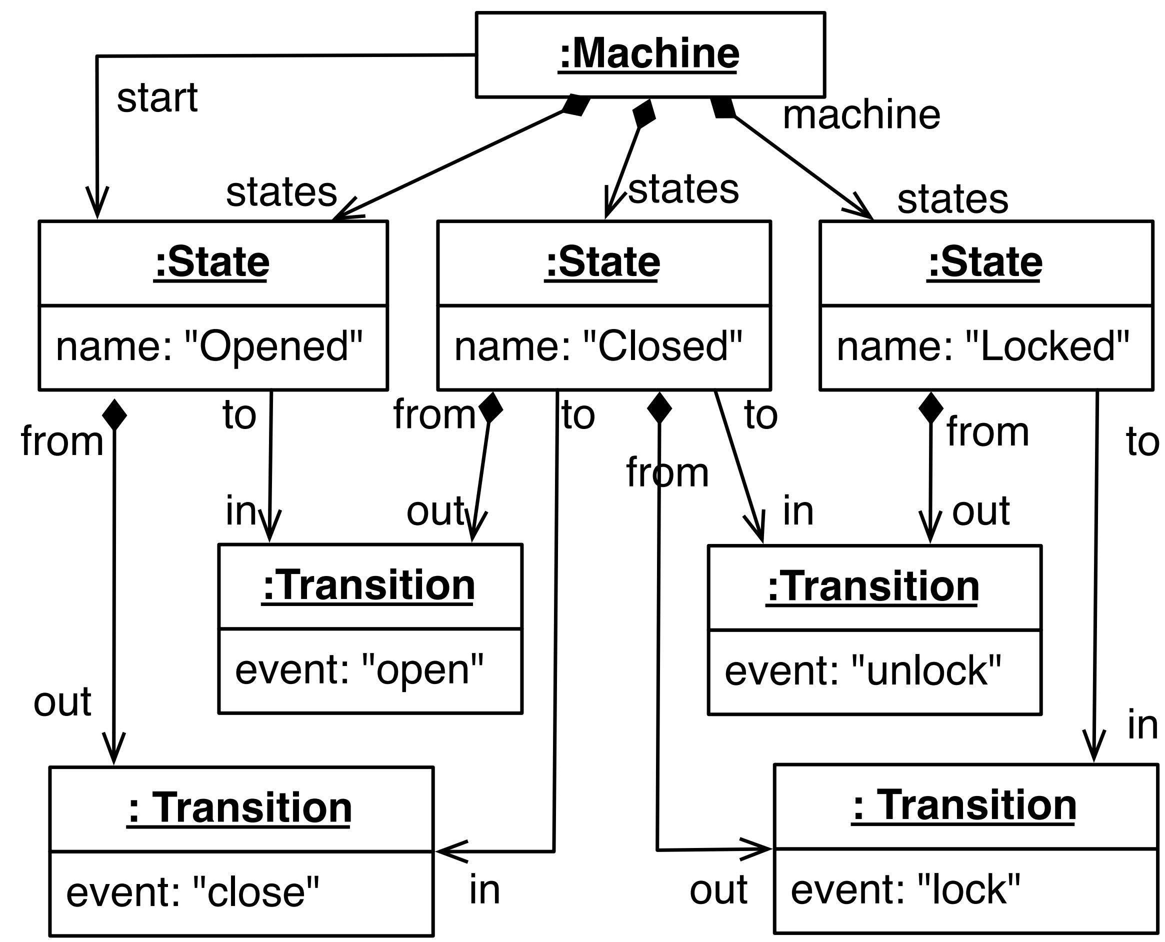
```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```



Creating cross links

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

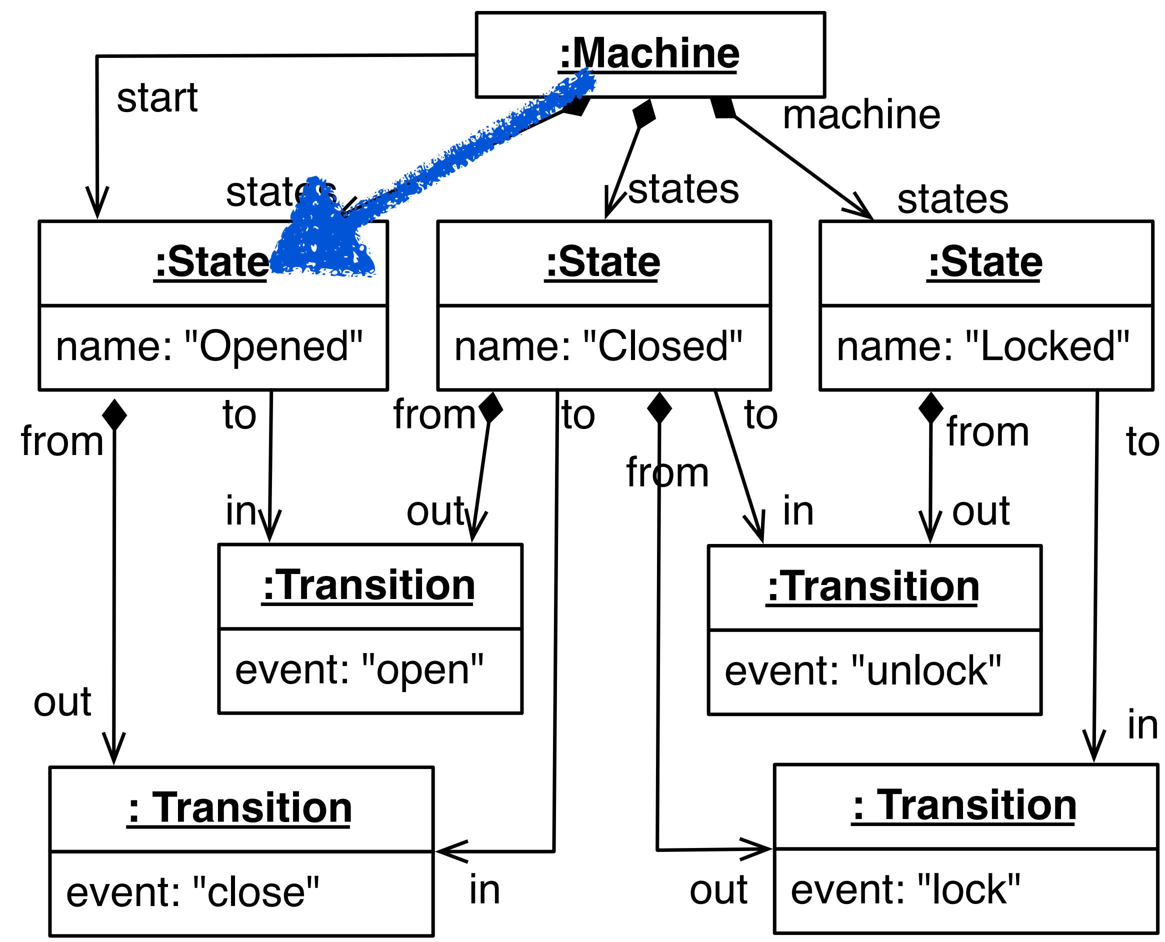
start:</states["Opened"]>



Creating cross links

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

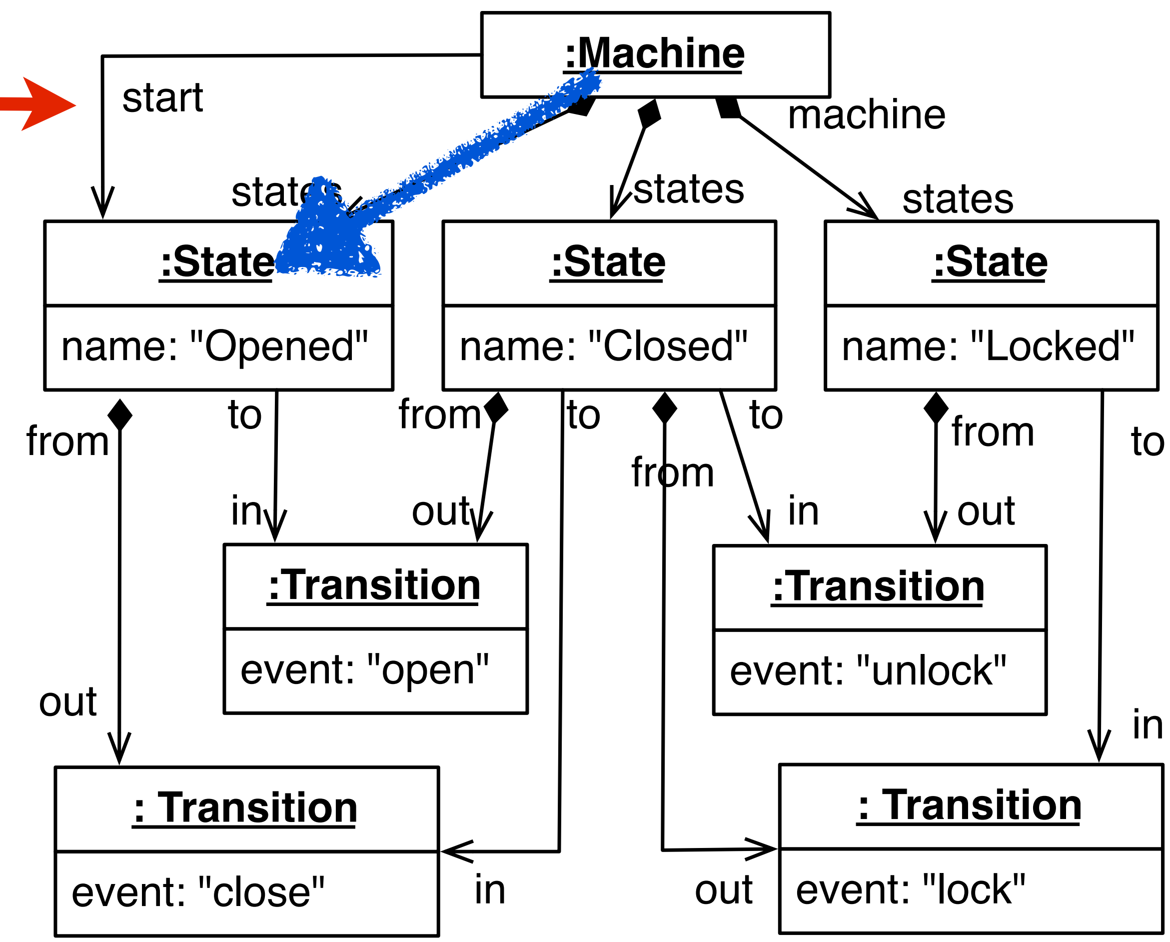
start:</states[“Opened”]>



Creating cross links

```
start Opened
state Opened
   on close go Closed
state Closed
   on open go Opened
   on lock go Locked
state Locked
   on unlock go Closed
```

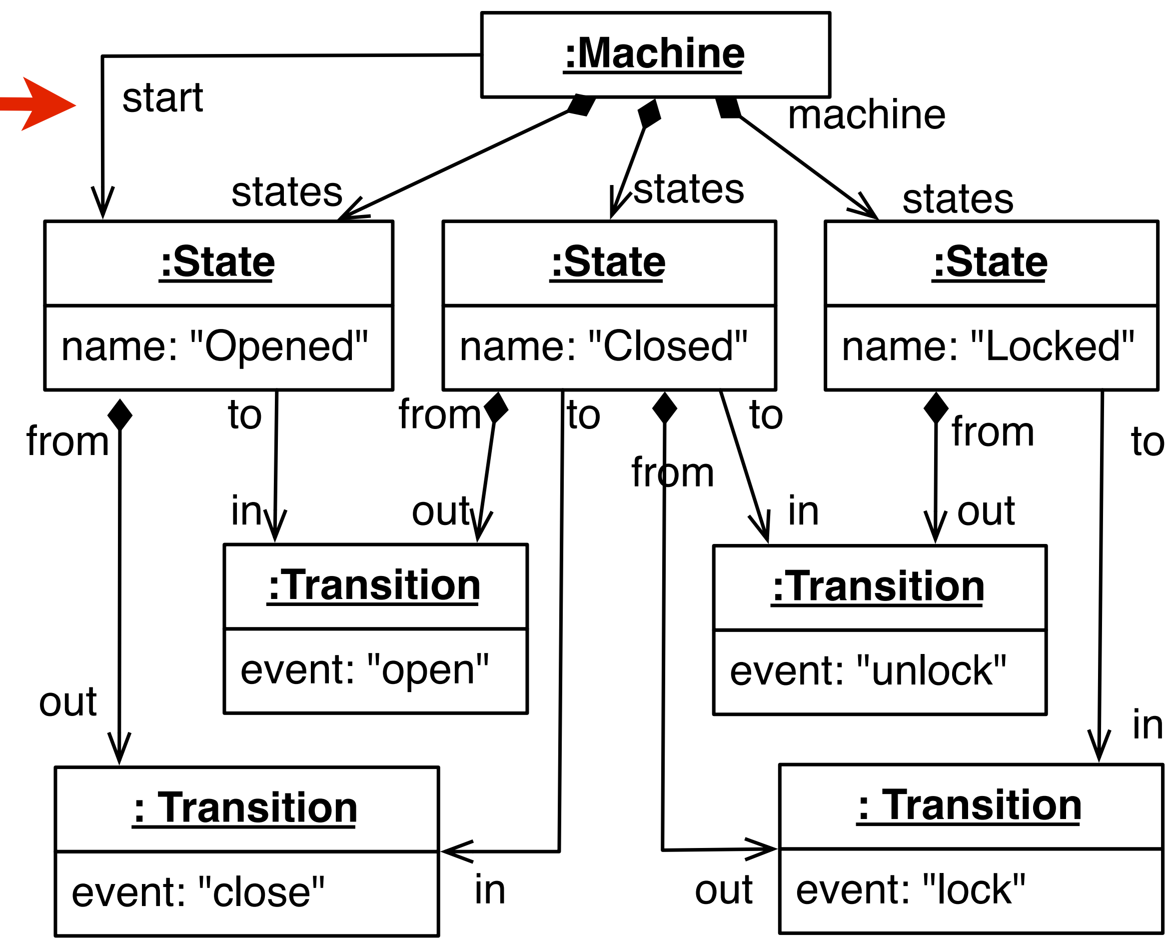
start:</states[“Opened”]>



Creating cross links

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

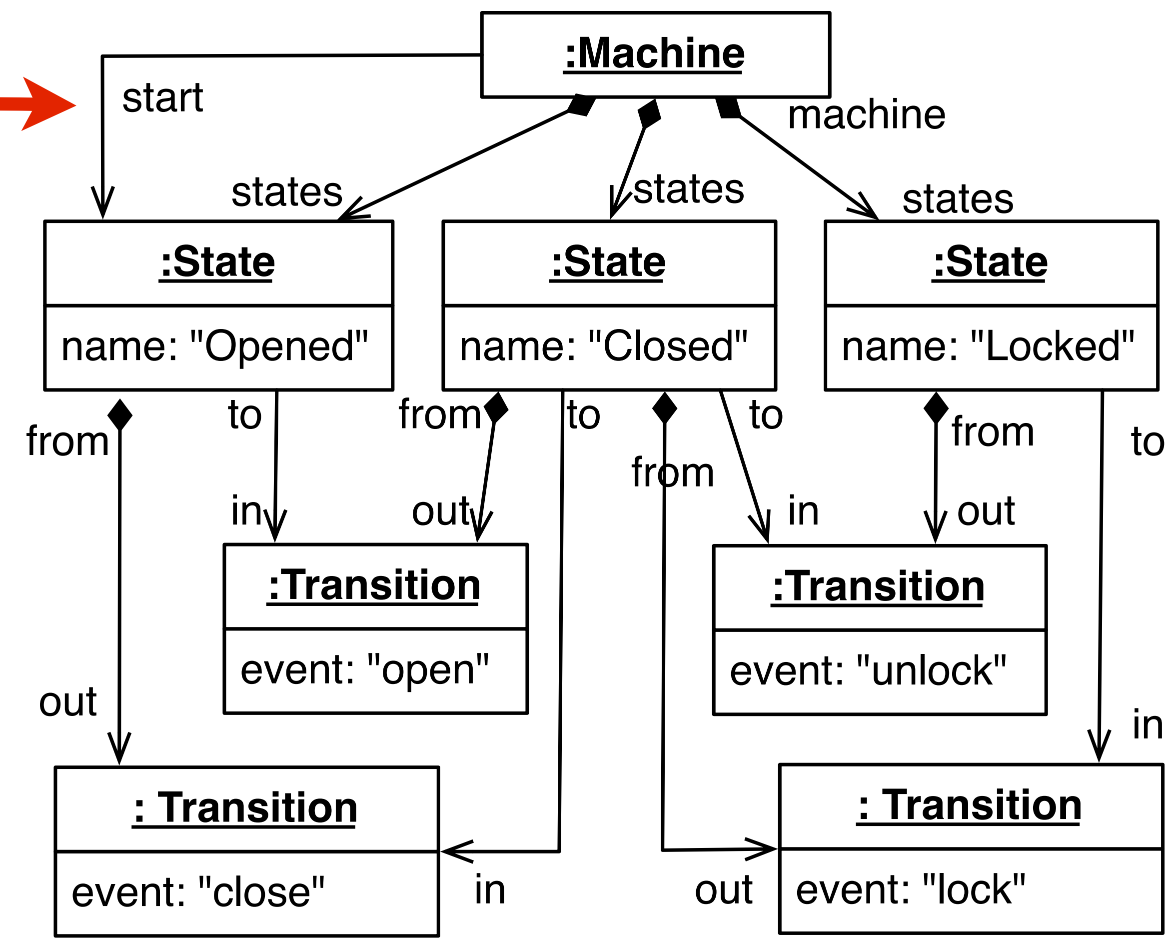
start:</states[“Opened”]>



Creating cross links

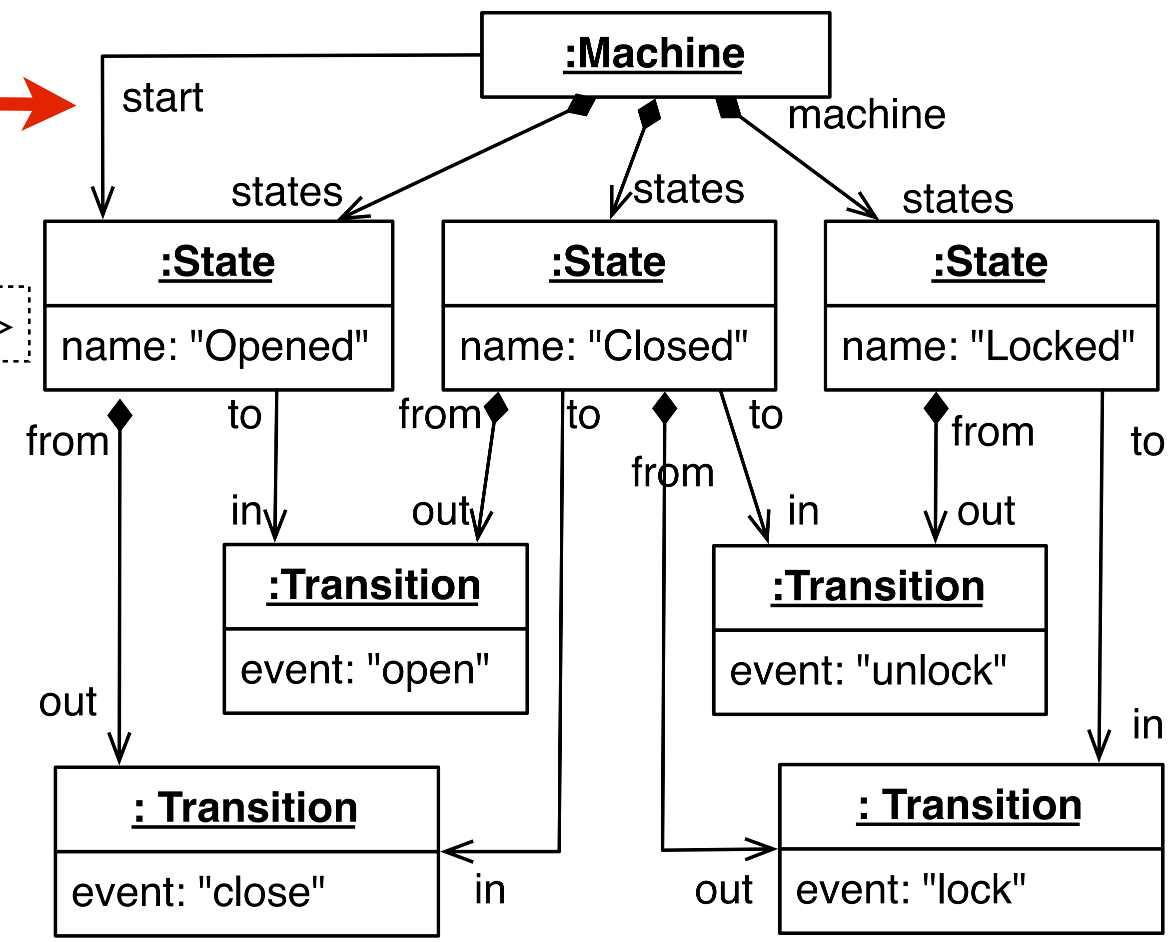
```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

start:</states[“Opened”]>



Creating cross links

```
start Opened      start:</states[“Opened”]>
state Opened
  on close go Closed   to:</states[“Closed”]>
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

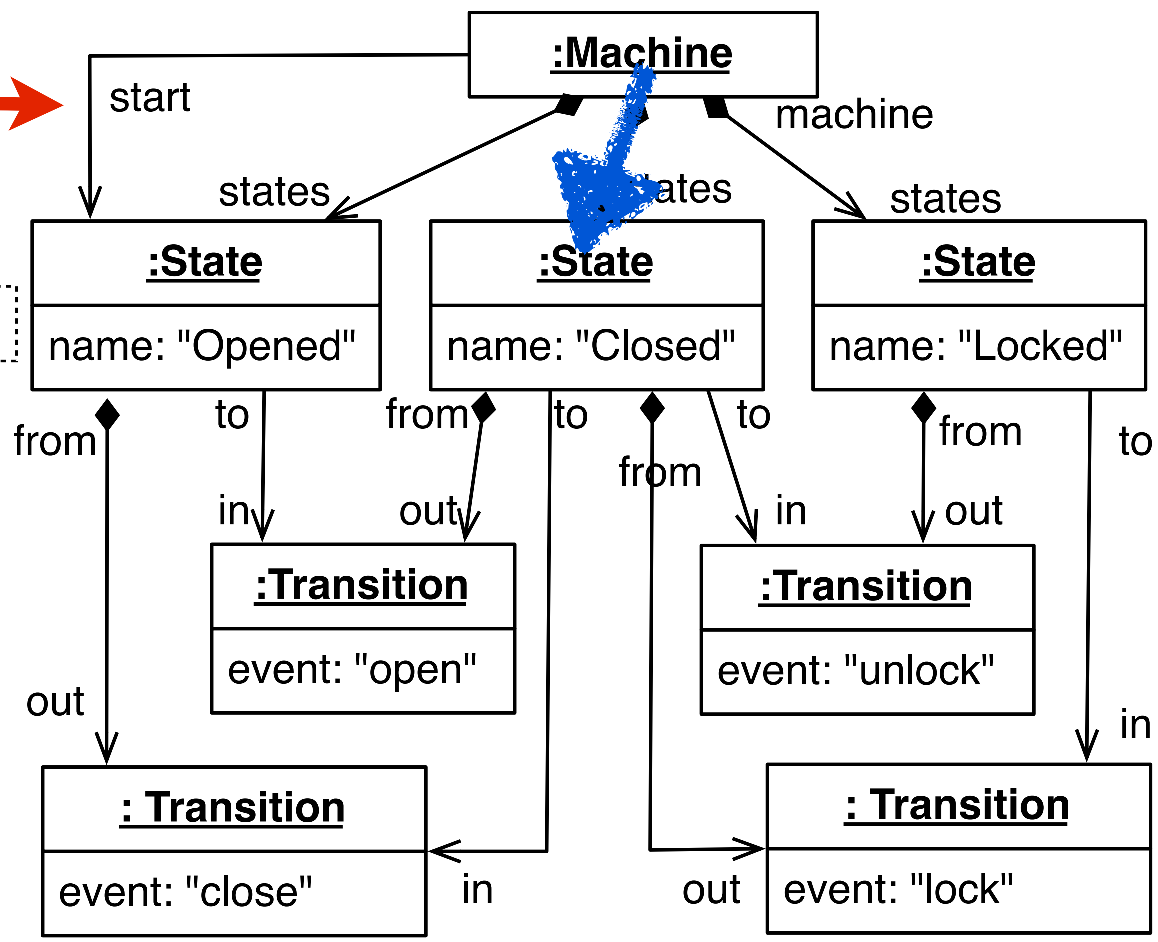


Creating cross links

```
start Opened
state Opened
   on close go Closed
state Closed
   on open go Opened
   on lock go Locked
state Locked
   on unlock go Closed
```

start:</states[“Opened”]>

to:</states[“Closed”]>

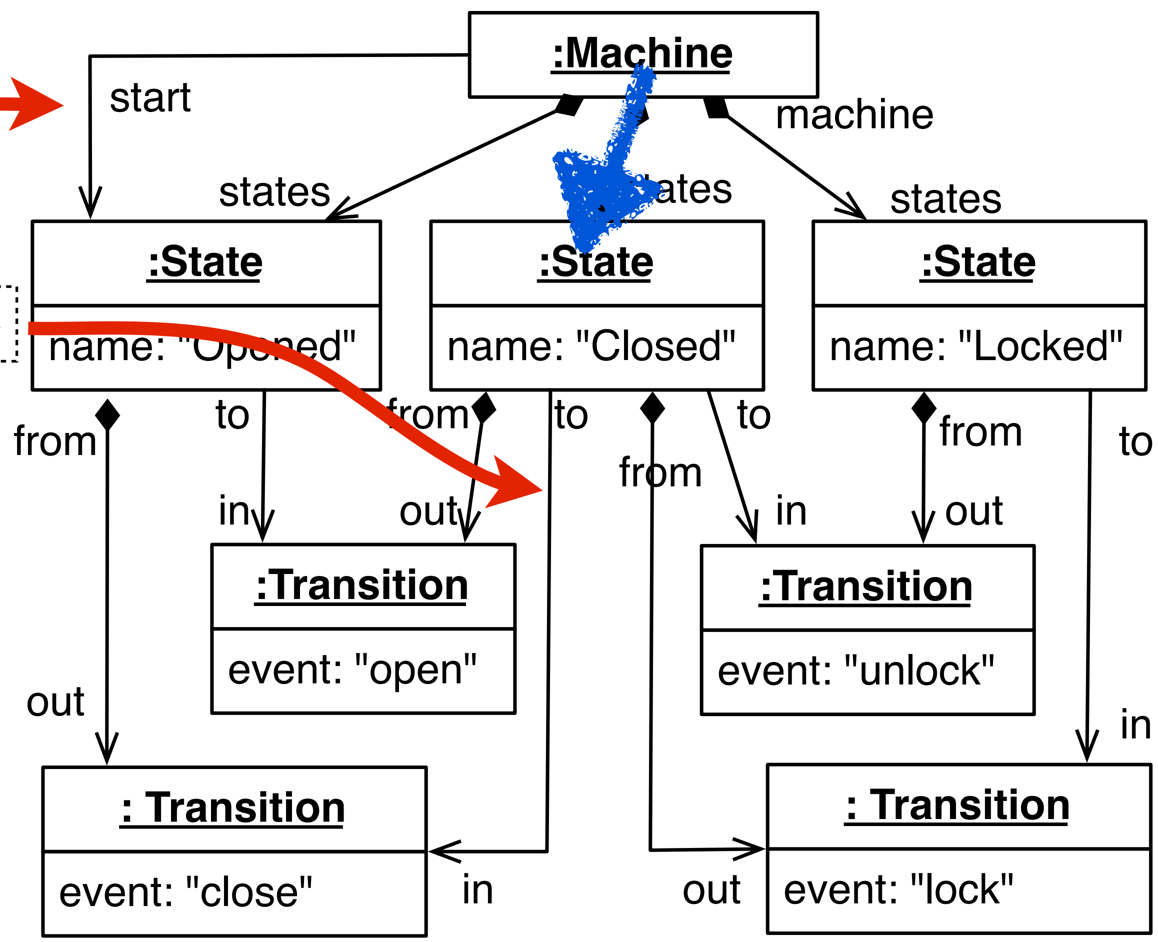


Creating cross links

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

start:</states[“Opened”]>

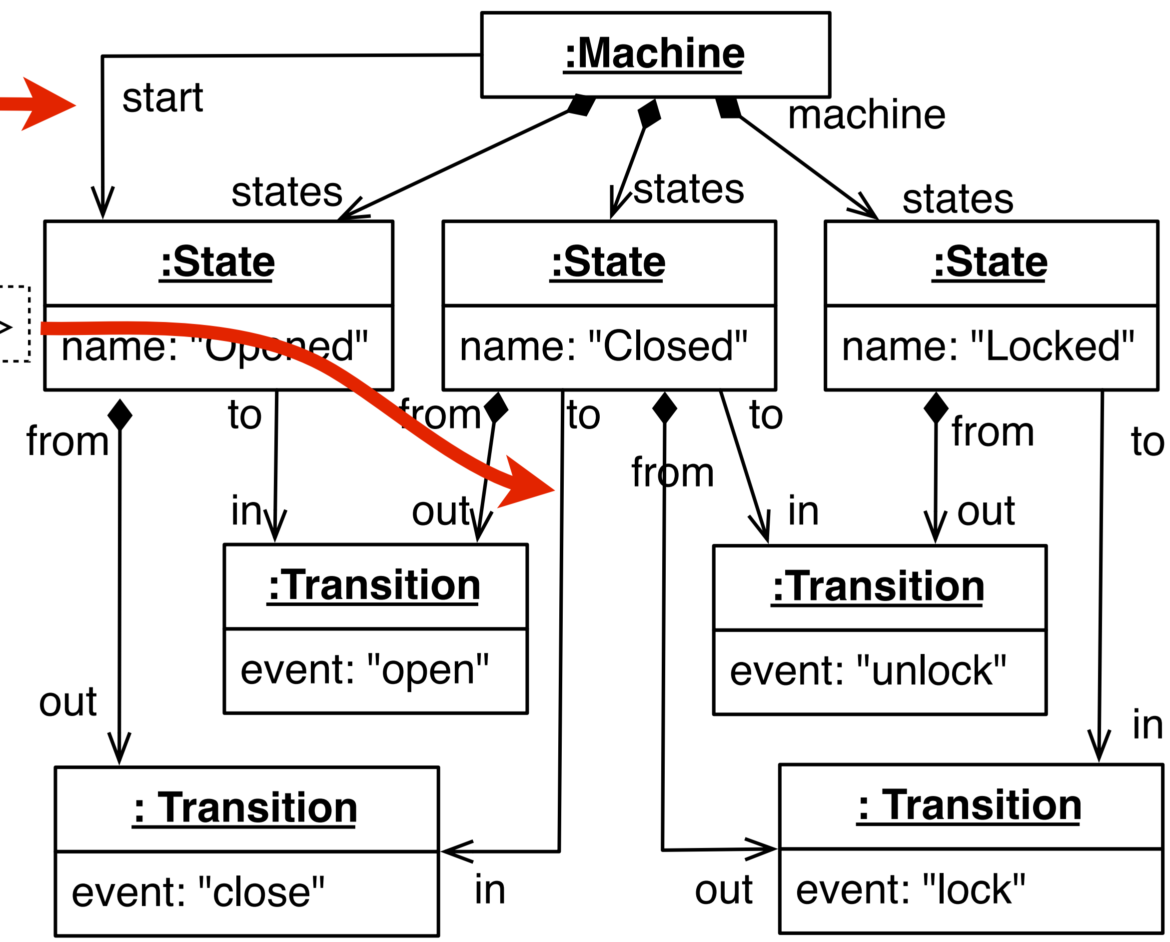
to:</states[“Closed”]>



Creating cross links

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

```
start:</states["Opened"]>
to:</states["Closed"]>
```

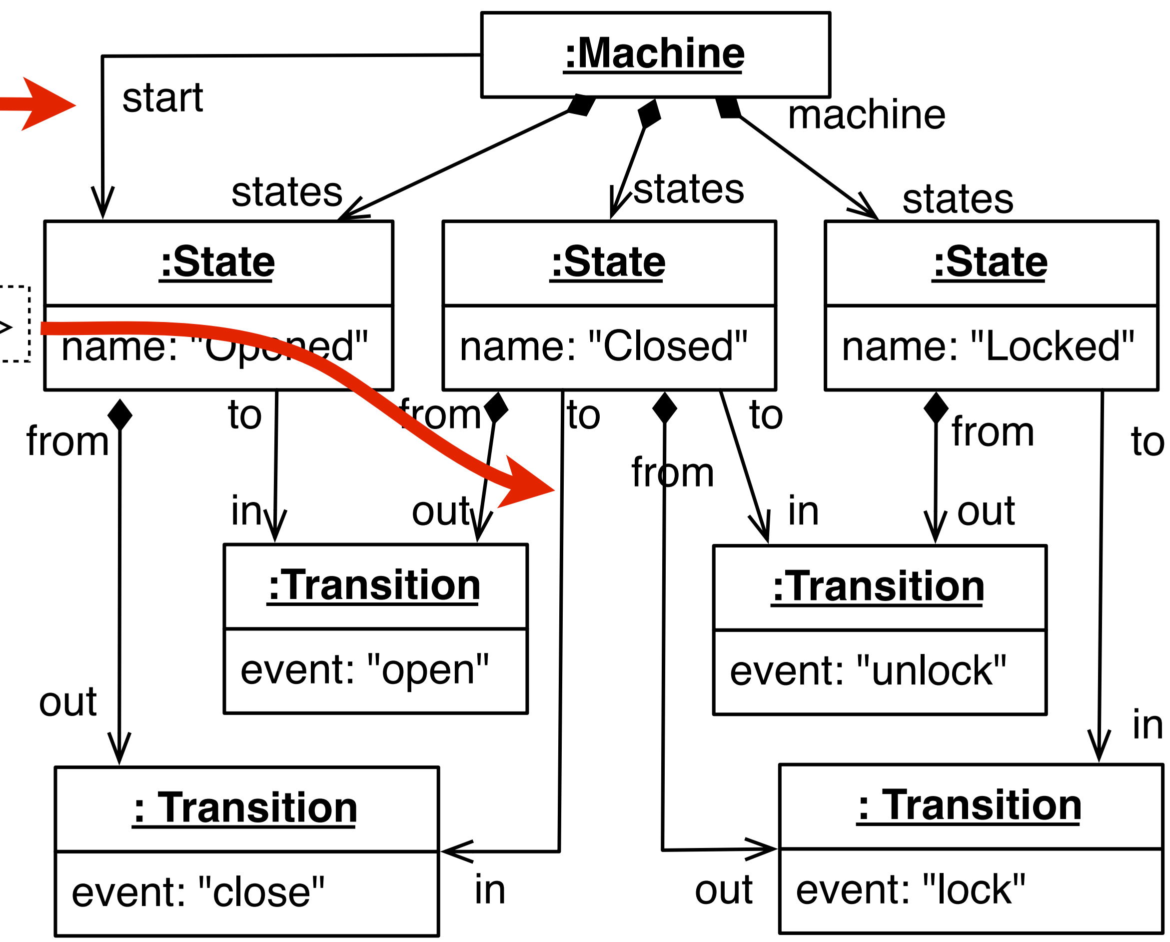


Creating cross links

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

start:</states[“Opened”]>

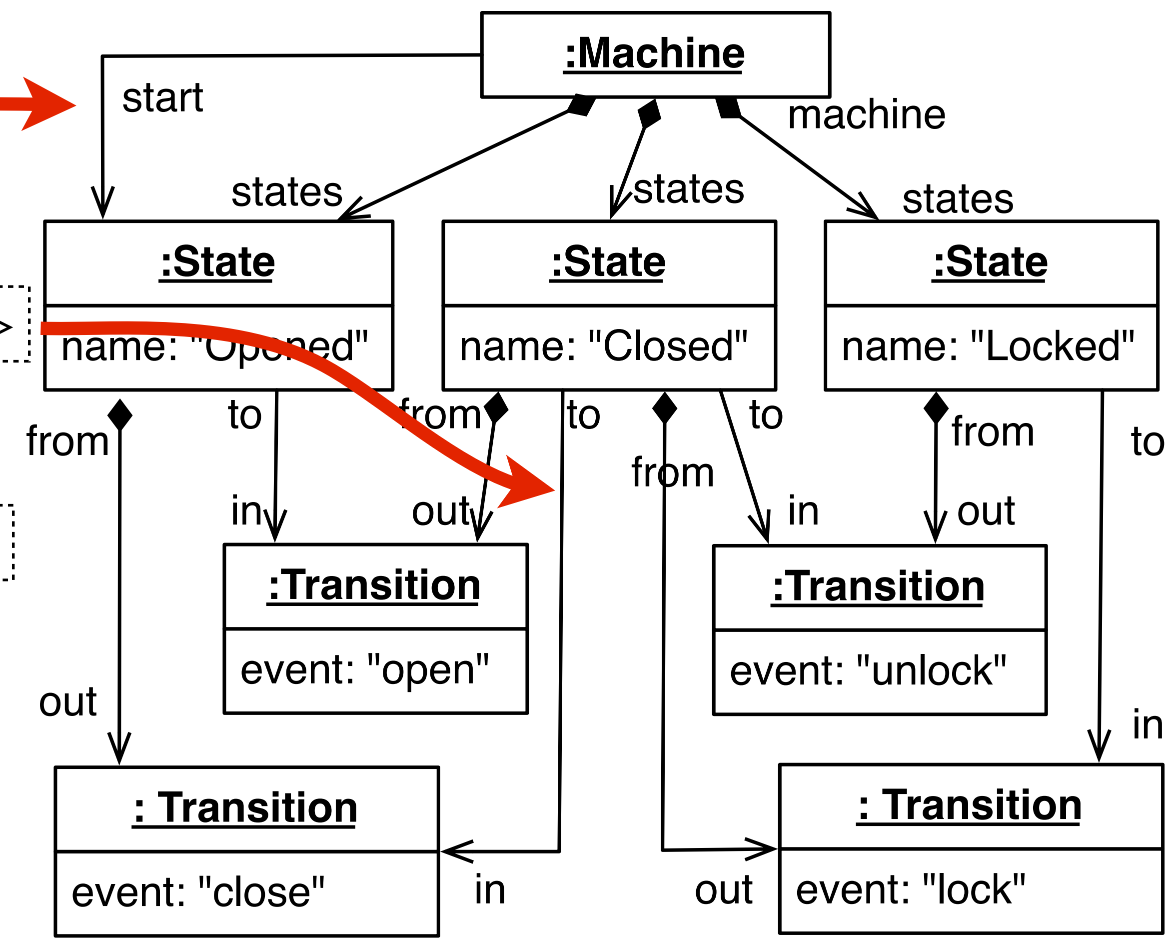
to:</states[“Closed”]>



Creating cross links

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

```
start:</states["Opened"]>
to:</states["Closed"]>
to:</states["Opened"]>
```



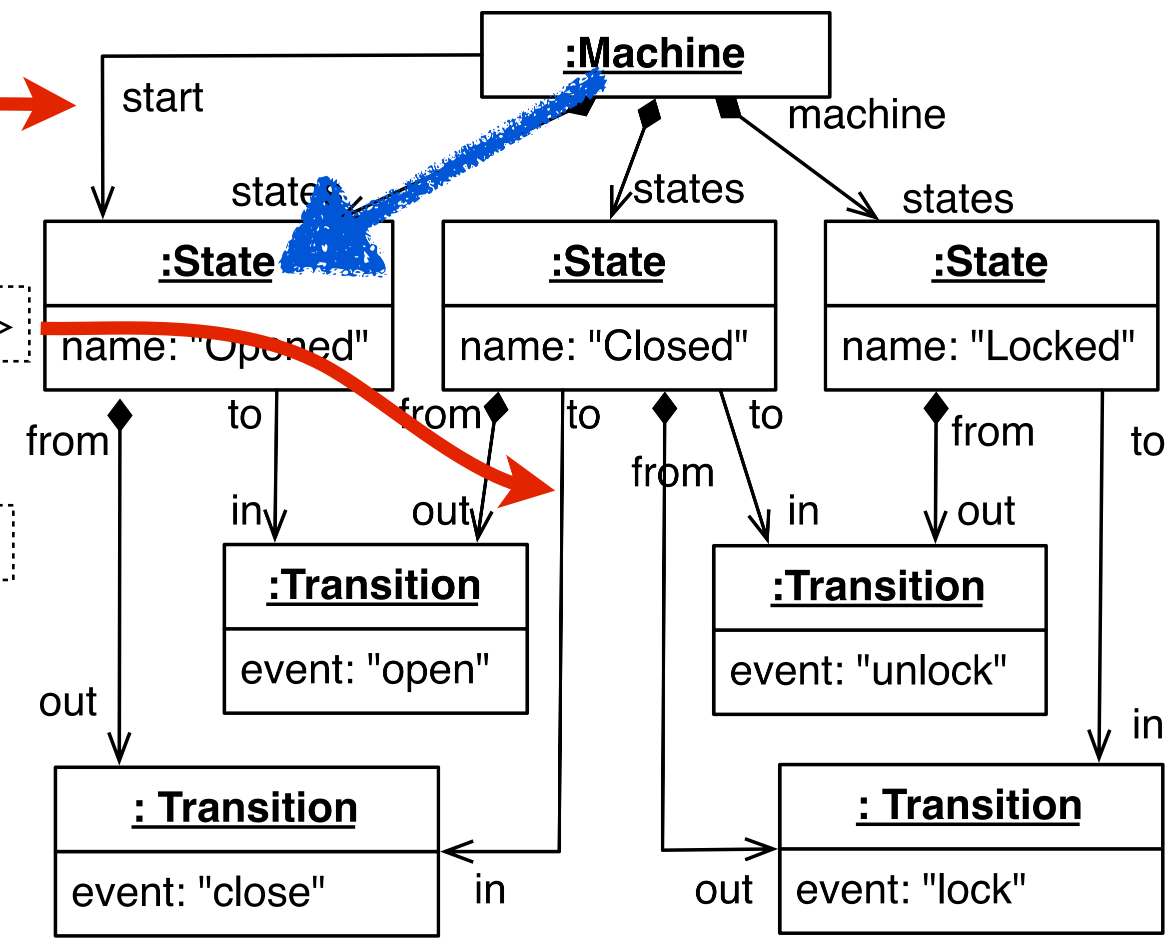
Creating cross links

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

start:</states[“Opened”]>

to:</states[“Closed”]>

to:</states[“Opened”]>



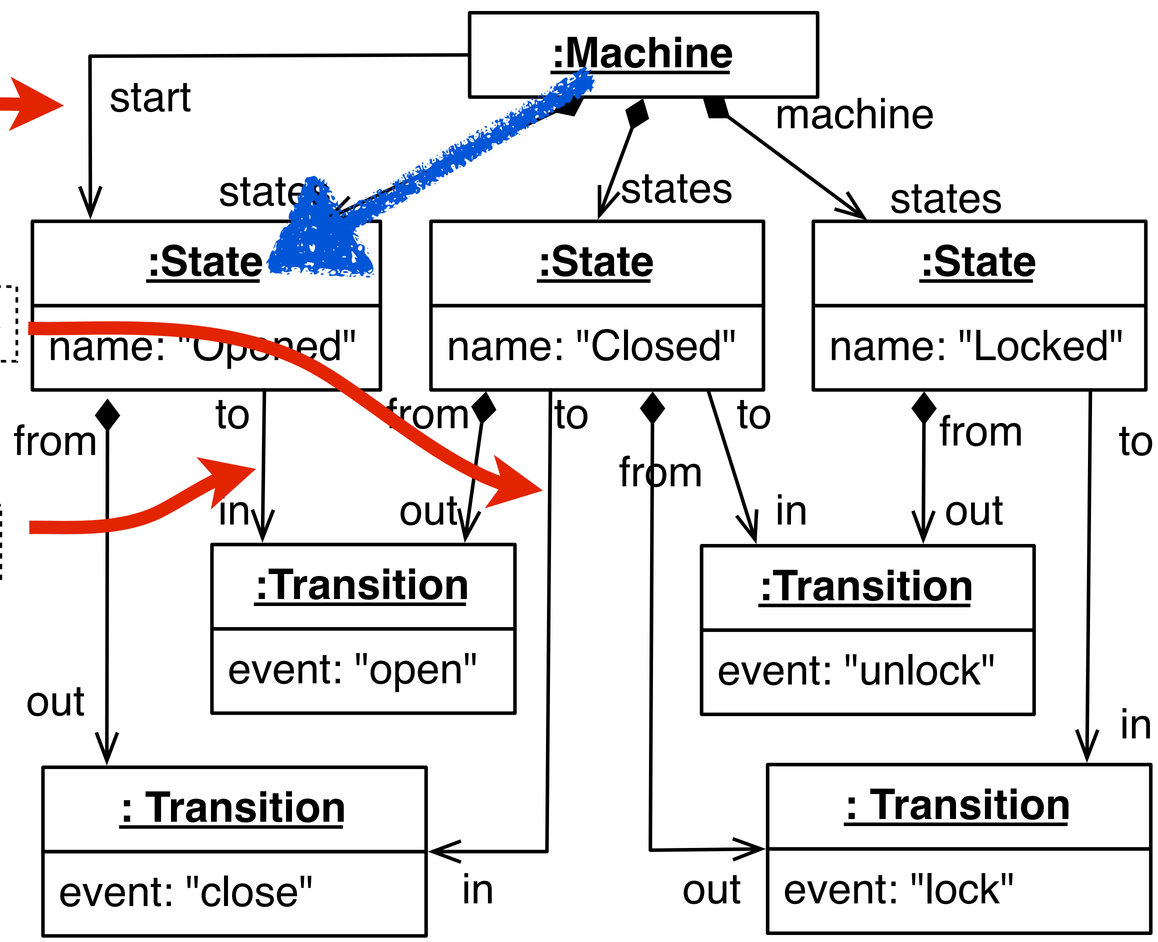
Creating cross links

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

start:</states[“Opened”]>

to:</states[“Closed”]>

to:</states[“Opened”]>



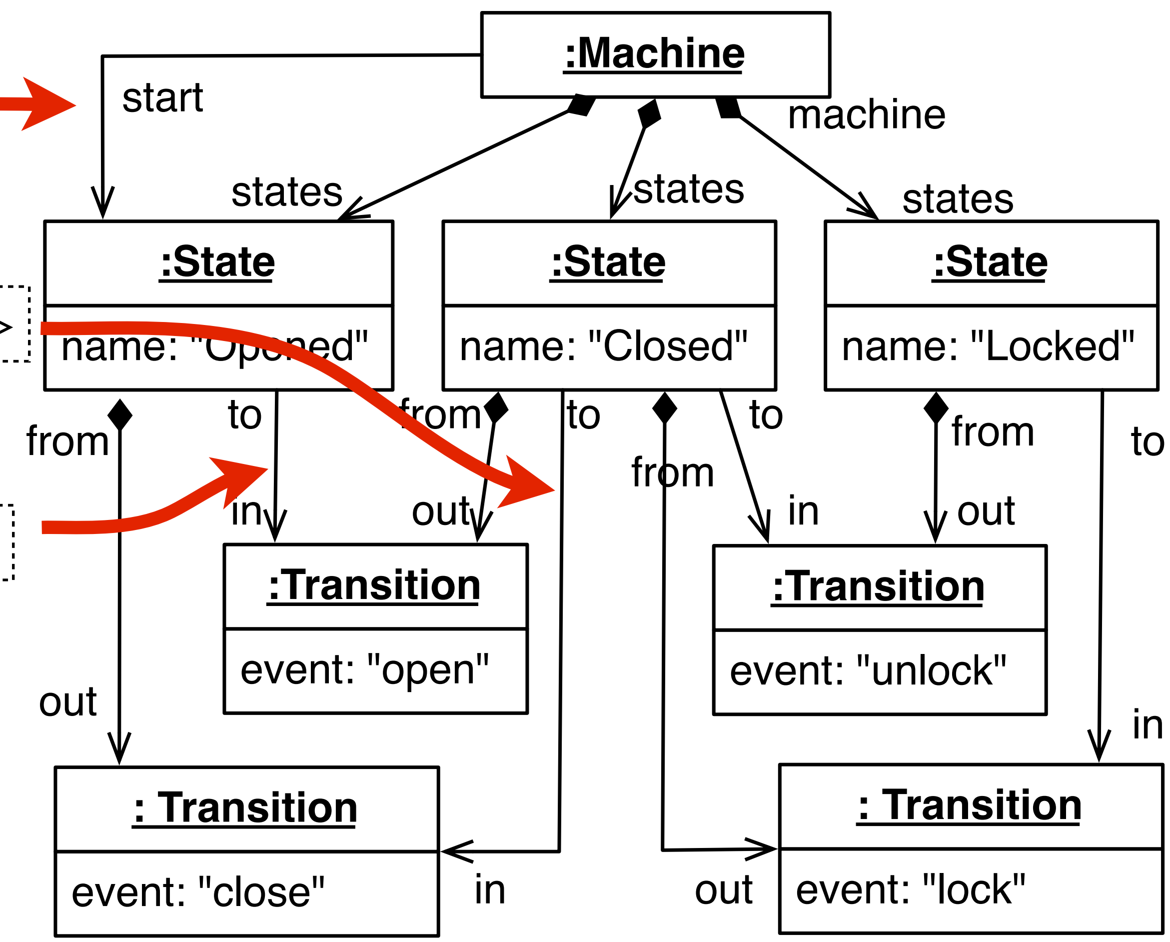
Creating cross links

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

`start:</states[“Opened”]>`

`to:</states[“Closed”]>`

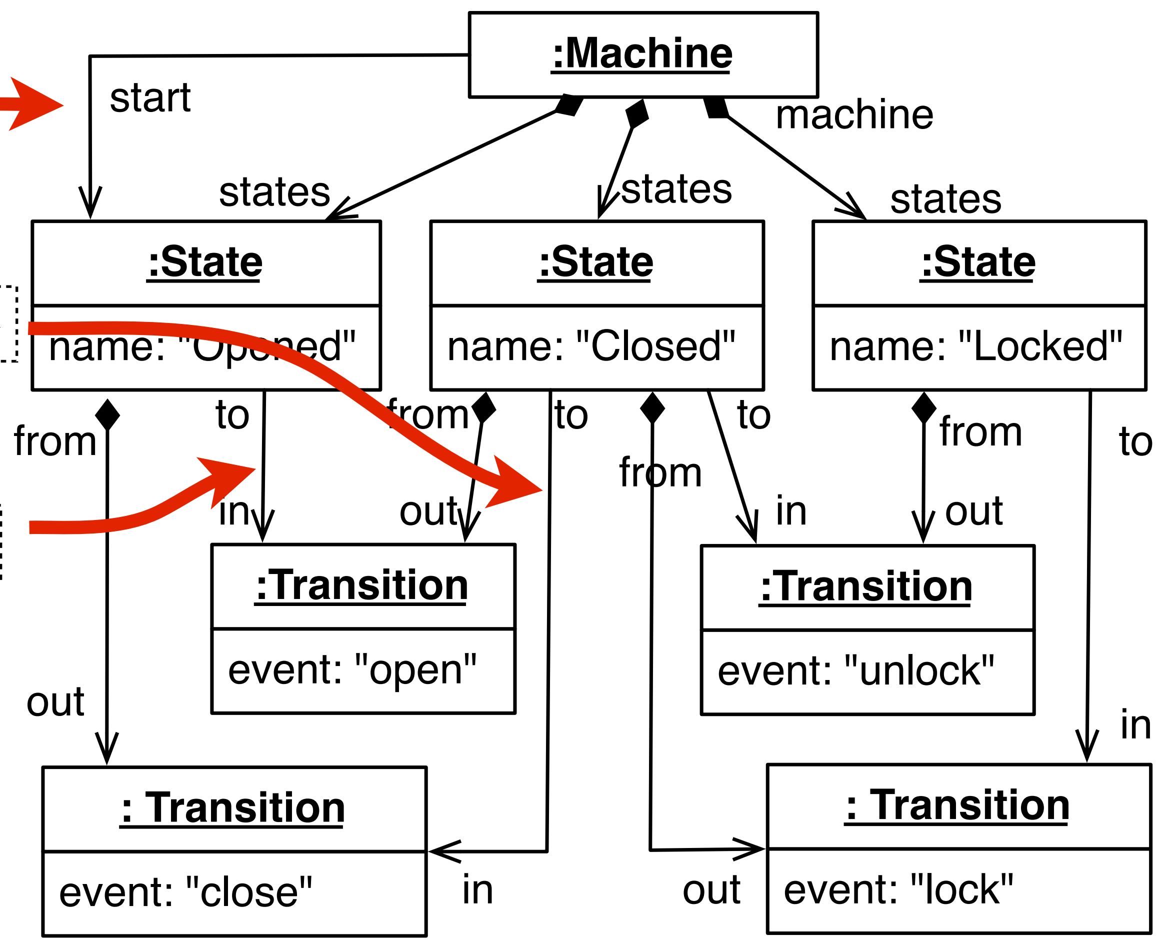
`to:</states[“Opened”]>`



Creating cross links

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

```
start:</states["Opened"]>
to:</states["Closed"]>
to:</states["Opened"]>
```



Creating cross links

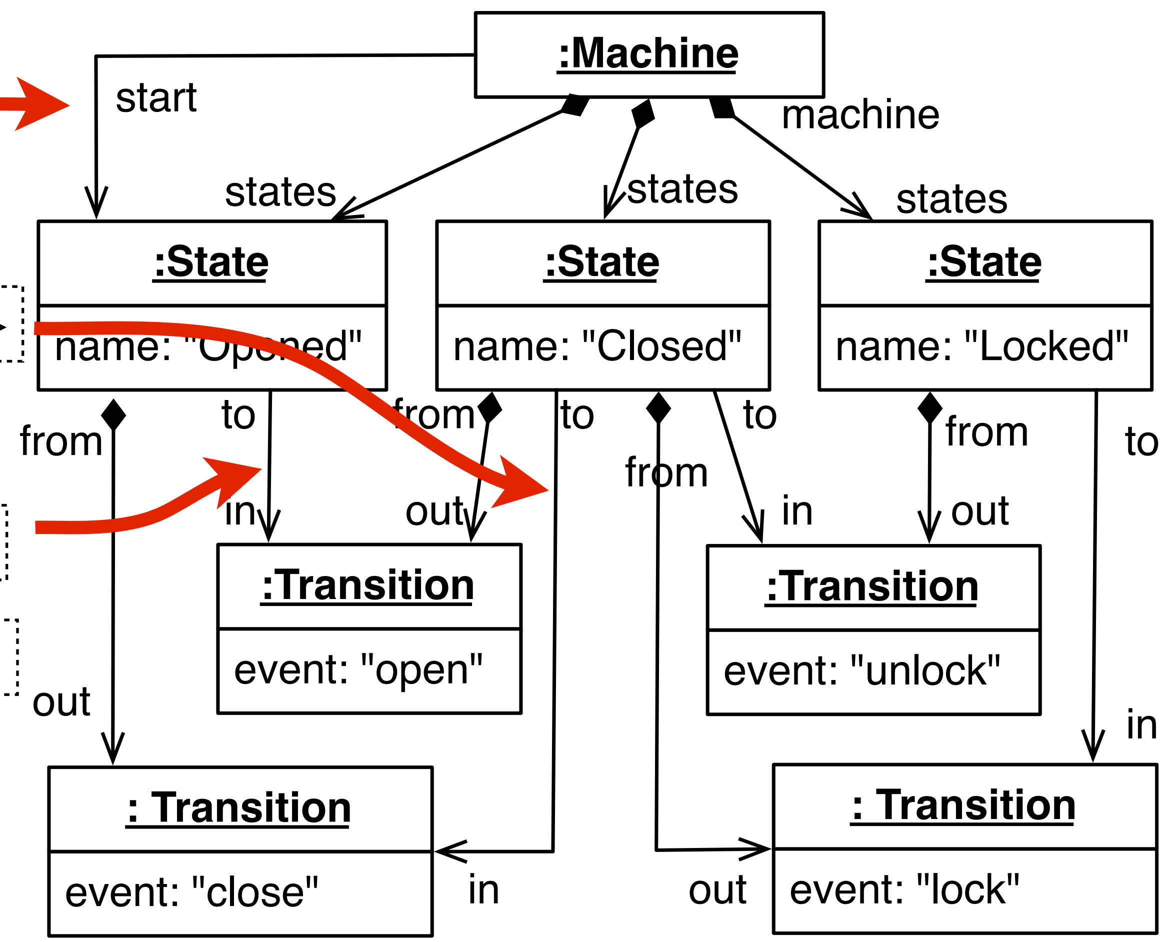
```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

start:</states[“Opened”]>

to:</states[“Closed”]>

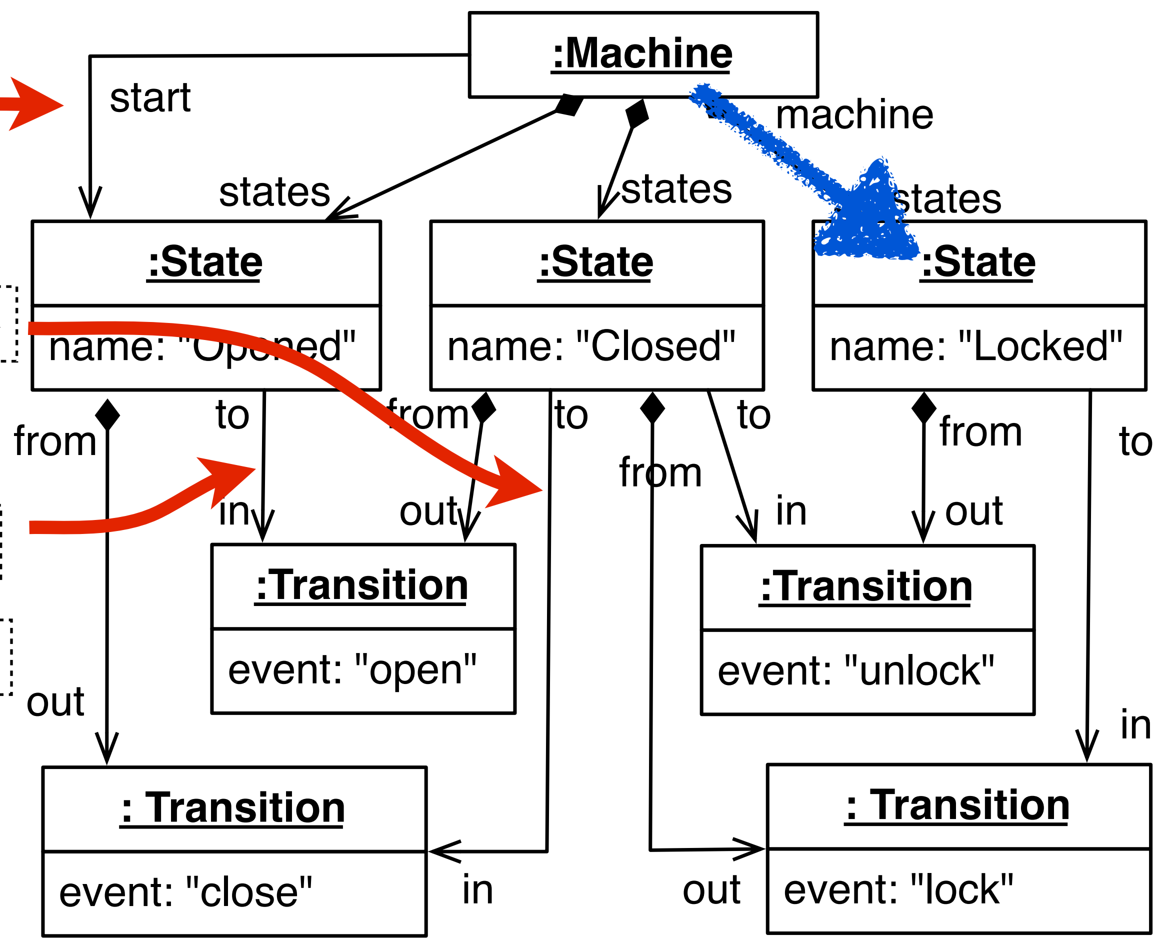
to:</states[“Opened”]>

to:</states[“Locked”]>



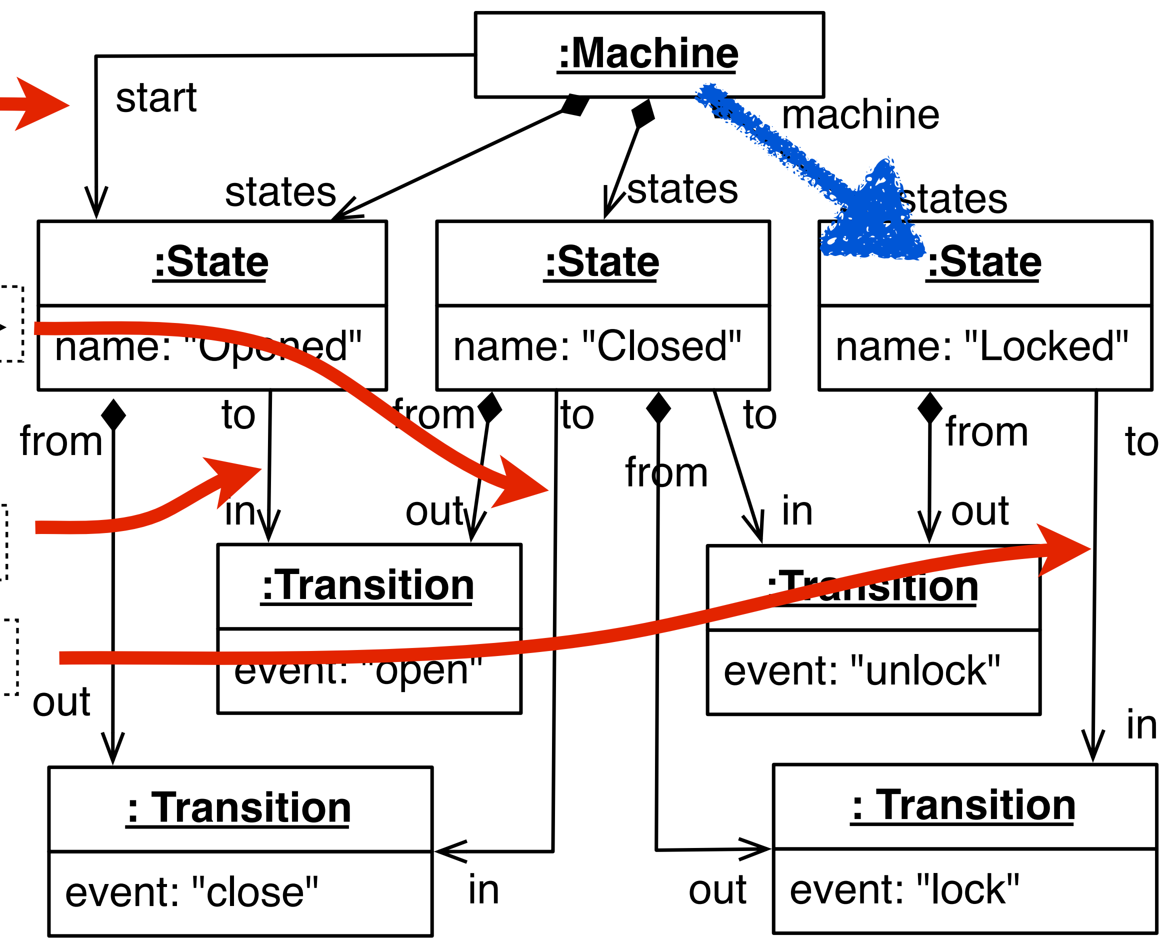
Creating cross links

```
start Opened              start:</states[“Opened”]>
state Opened
  on close go Closed      to:</states[“Closed”]>
state Closed
  on open go Opened       to:</states[“Opened”]>
  on lock go Locked       to:</states[“Locked”]>
state Locked
  on unlock go Closed
```



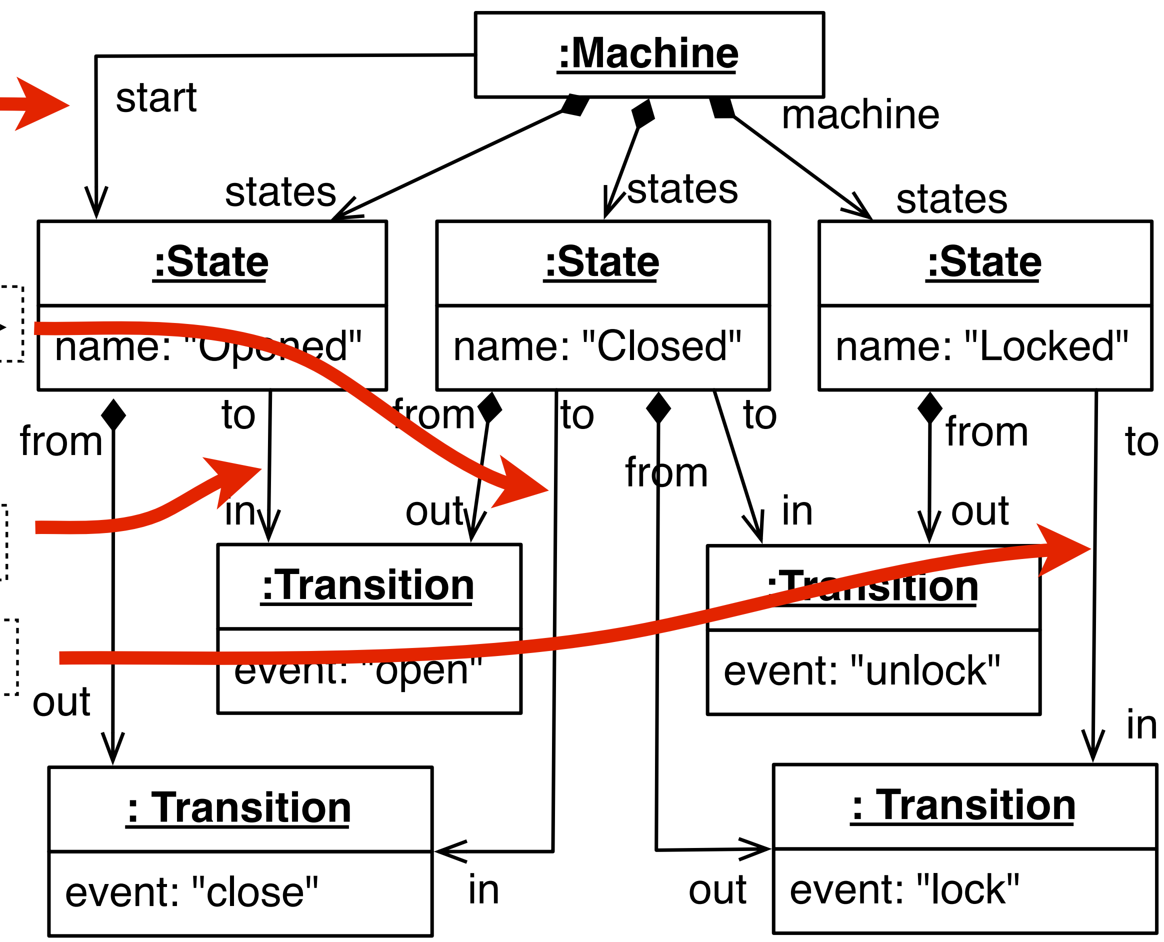
Creating cross links

```
start Opened              start:</states[“Opened”]>
state Opened
  on close go Closed      to:</states[“Closed”]>
state Closed
  on open go Opened       to:</states[“Opened”]>
  on lock go Locked       to:</states[“Locked”]>
state Locked
  on unlock go Closed
```



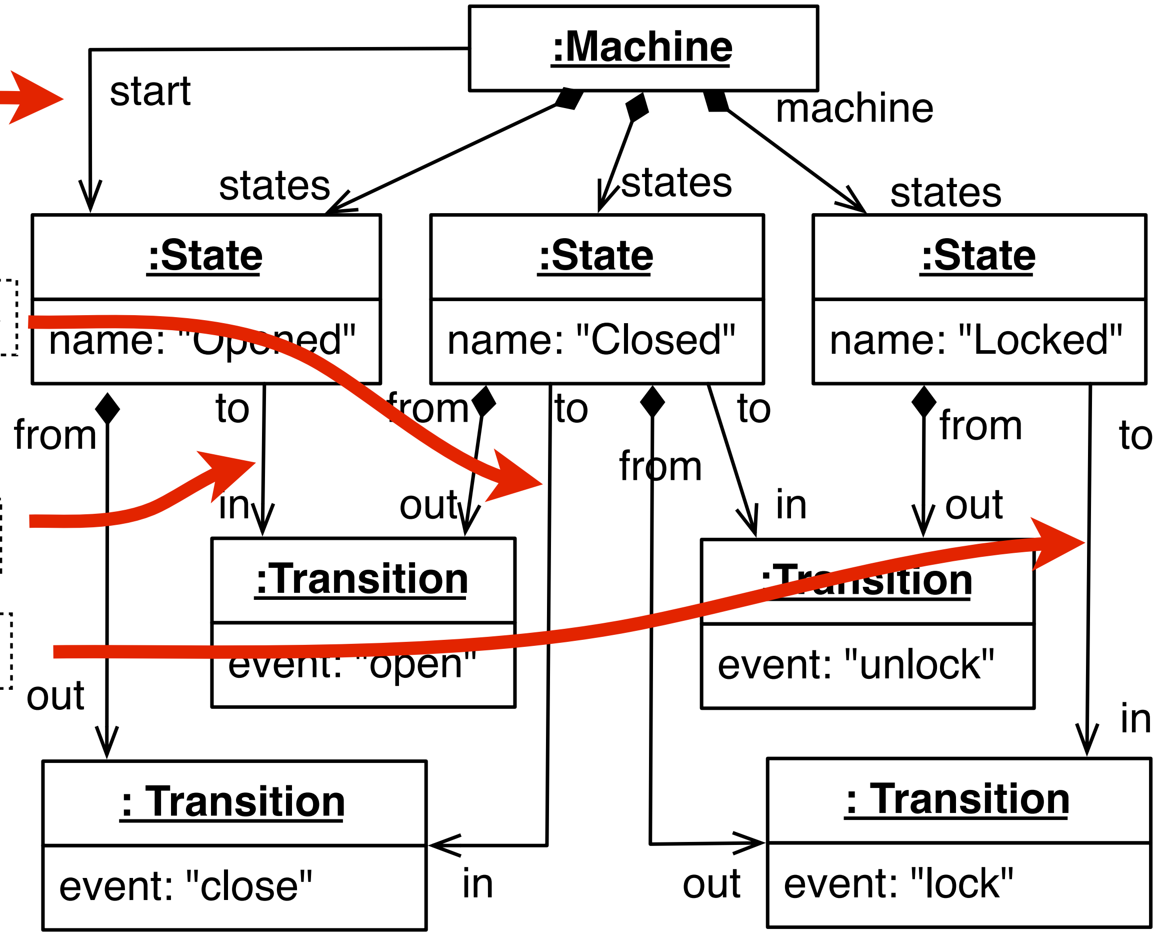
Creating cross links

```
start Opened                 start:</states[“Opened”]>
state Opened
  on close go Closed         to:</states[“Closed”]>
state Closed
  on open go Opened          to:</states[“Opened”]>
  on lock go Locked          to:</states[“Locked”]>
state Locked
  on unlock go Closed
```



Creating cross links

```
start Opened                  start:</states[“Opened”]>
state Opened
  on close go Closed          to:</states[“Closed”]>
state Closed
  on open go Opened           to:</states[“Opened”]>
  on lock go Locked           to:</states[“Locked”]>
state Locked
  on unlock go Closed
```



Creating cross links

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

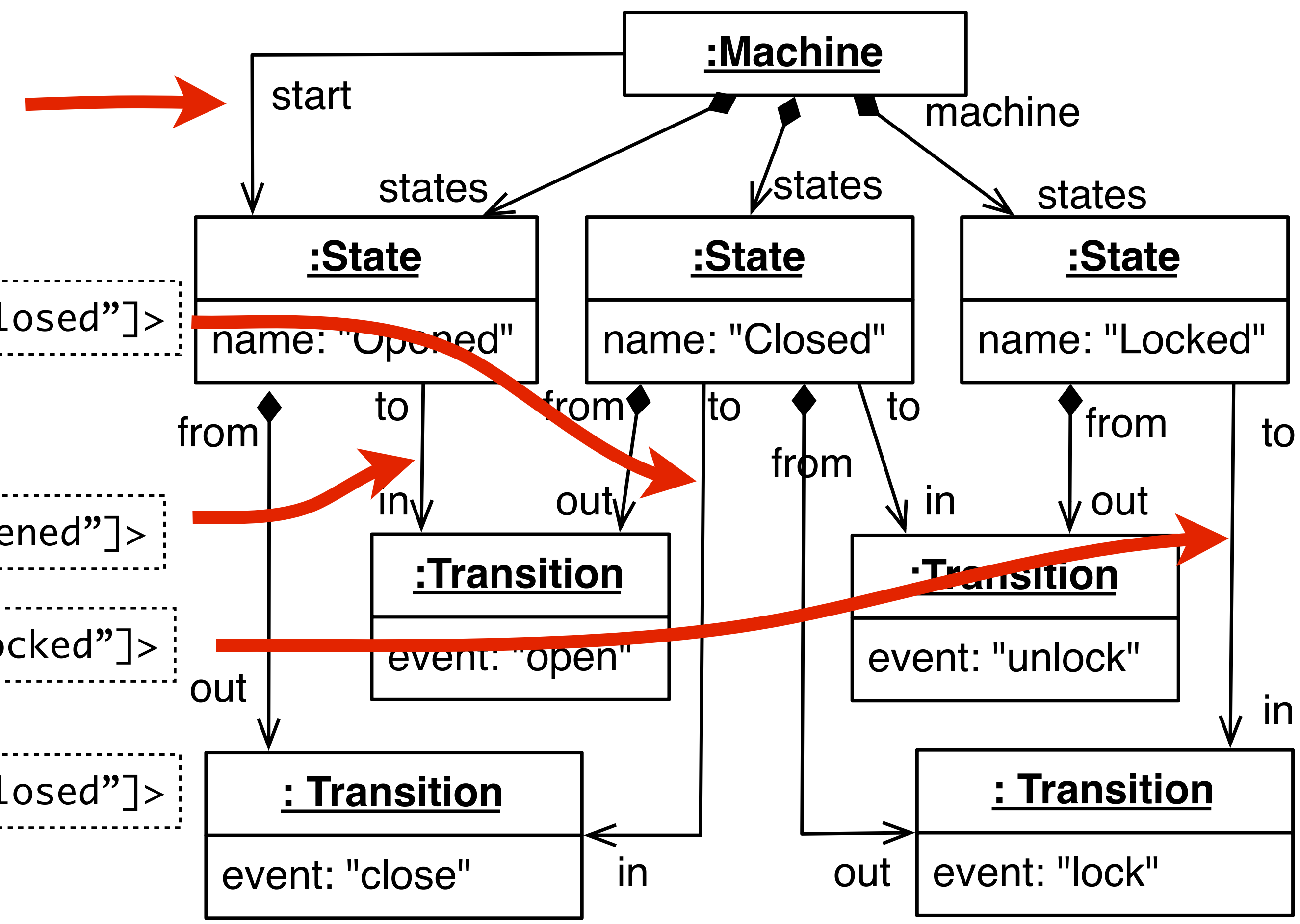
start:</states[“Opened”]>

to:</states[“Closed”]>

to:</states[“Opened”]>

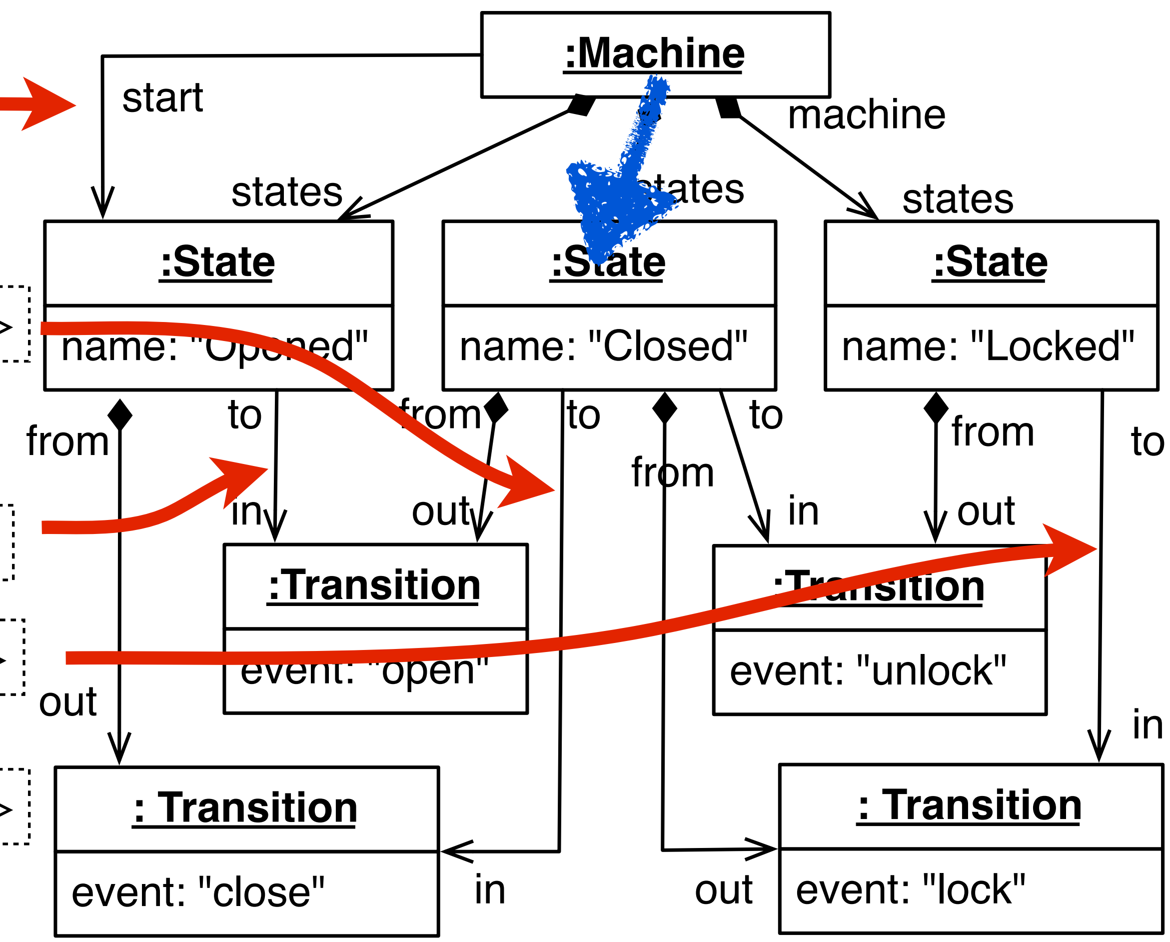
to:</states[“Locked”]>

to:</states[“Closed”]>



Creating cross links

```
start Opened            start:</states["Opened"]>
state Opened
  on close go Closed    to:</states["Closed"]>
state Closed
  on open go Opened     to:</states["Opened"]>
  on lock go Locked     to:</states["Locked"]>
state Locked            to:</states["Closed"]>
  on unlock go Closed
```



Creating cross links

```
start Opened
state Opened
   on close go Closed
state Closed
   on open go Opened
   on lock go Locked
state Locked
   on unlock go Closed
```

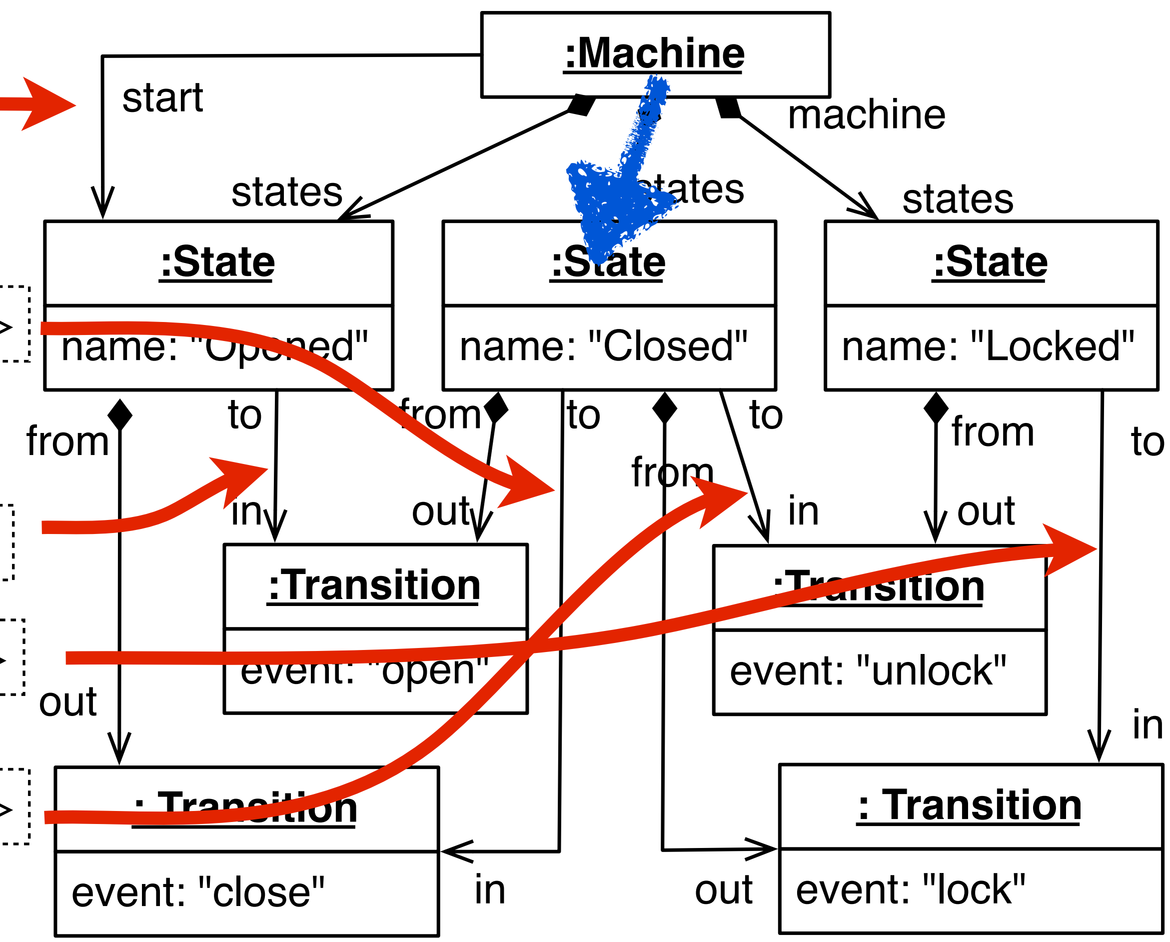
start:</states[“Opened”]>

to:</states[“Closed”]>

to:</states[“Opened”]>

to:</states[“Locked”]>

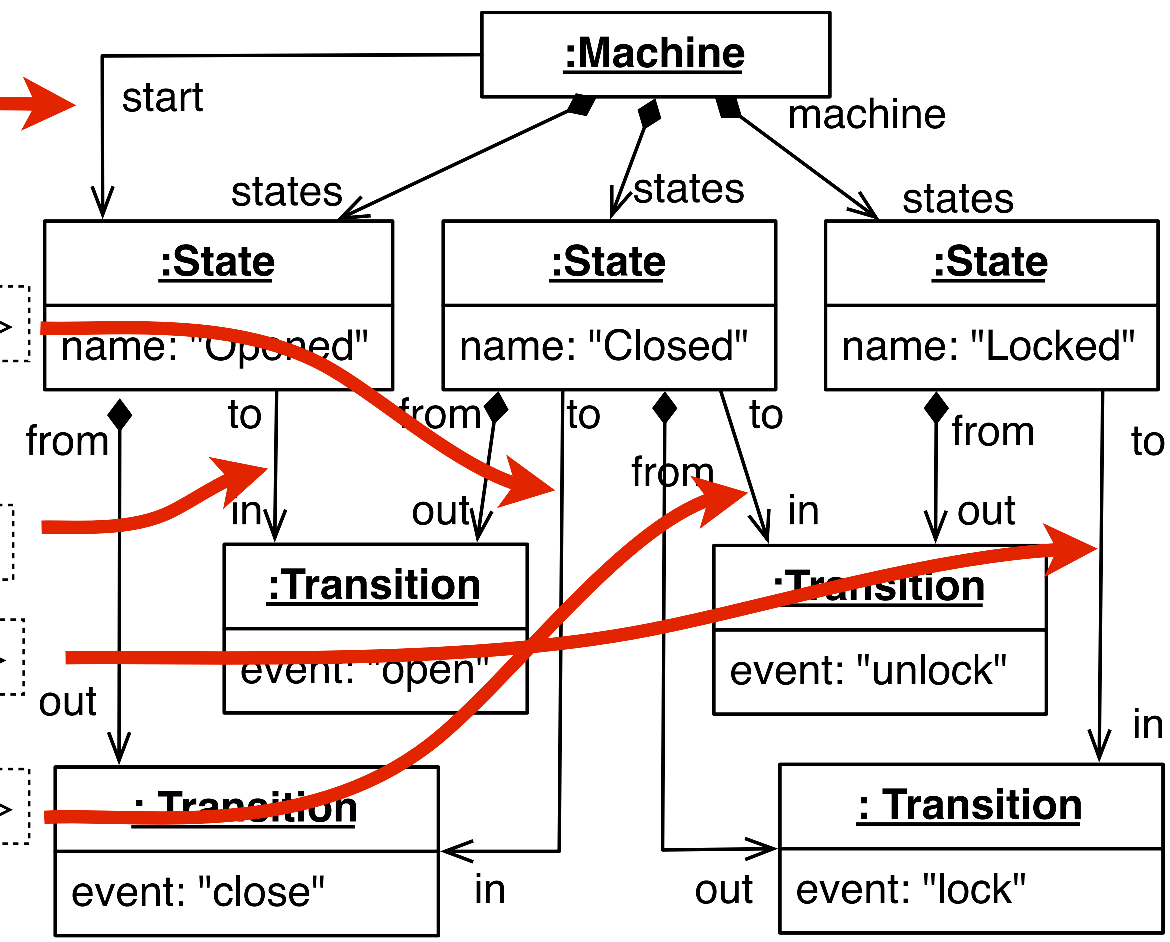
to:</states[“Closed”]>



Creating cross links

```
start Opened
state Opened
  on close go Closed
state Closed
  on open go Opened
  on lock go Locked
state Locked
  on unlock go Closed
```

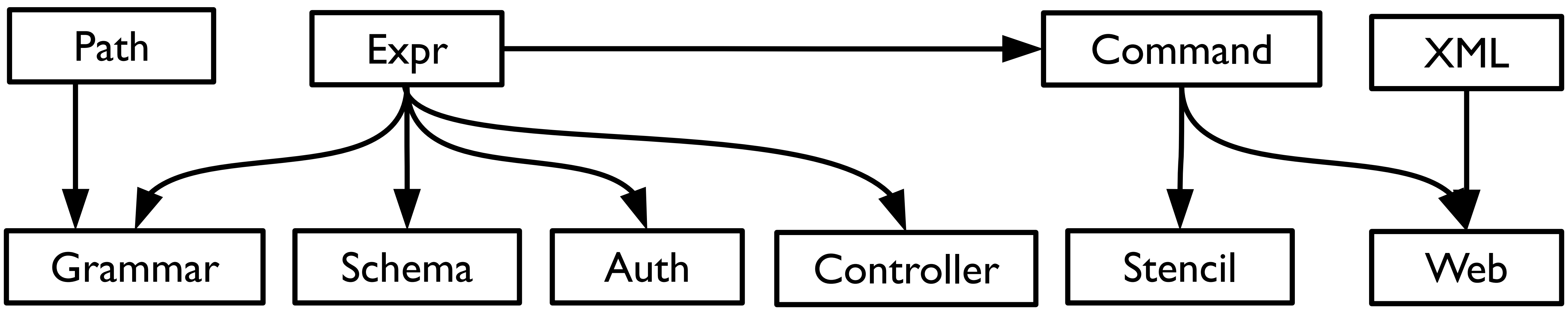
```
start:</states["Opened"]>
to:</states["Closed"]>
to:</states["Opened"]>
to:</states["Locked"]>
to:</states["Closed"]>
```



Assessment

- Bi-directional & compositional
- Flexible:
  - interleaved data binding
  - path-based references & predicates
  - formatting hints
- Self-described

Composition in Ensō



Conclusion

- Object grammars: mapping text to objects and vice versa
- Declarative paths for resolving cross-references
- Flexible, bi-directional and compositional
- Foundation of Ensō

Ensō 円相

Don't Design Your Programs, Program Your Designs

http://www.enso-lang.org/