Inferring Compact Models of Communication Protocol Entities

Therese Bohlin
Uppsala University, Uppsala, Sweden

Bengt Jonsson
Uppsala University, Uppsala, Sweden

Siavash Soleimanifard
KTH, Stockholm, Sweden

ISoLA 2010

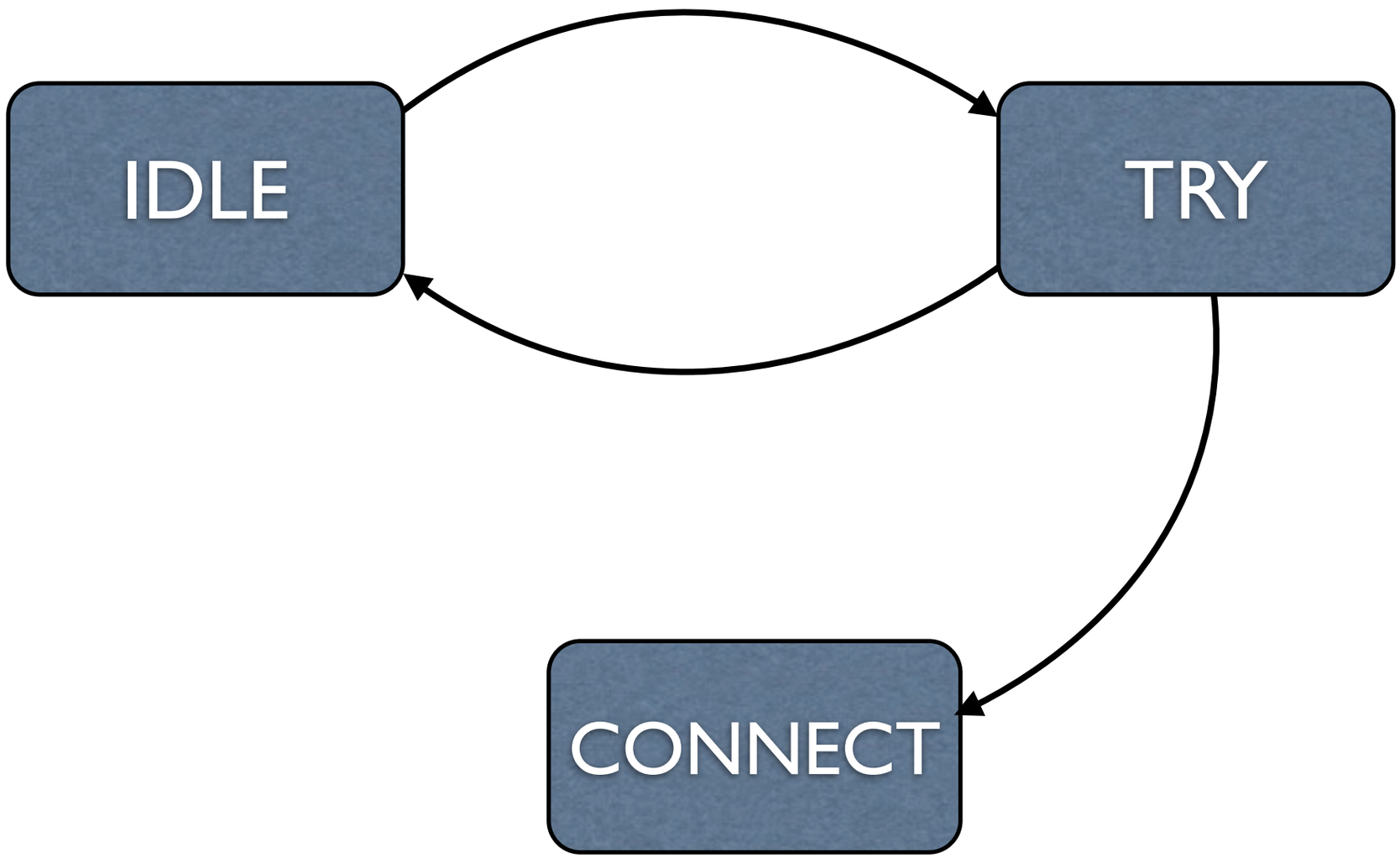
Introduction

- Modeling is important
  - verification
  - test case generation
  - ...
- Sometimes we have to consider a system as a “black-box”

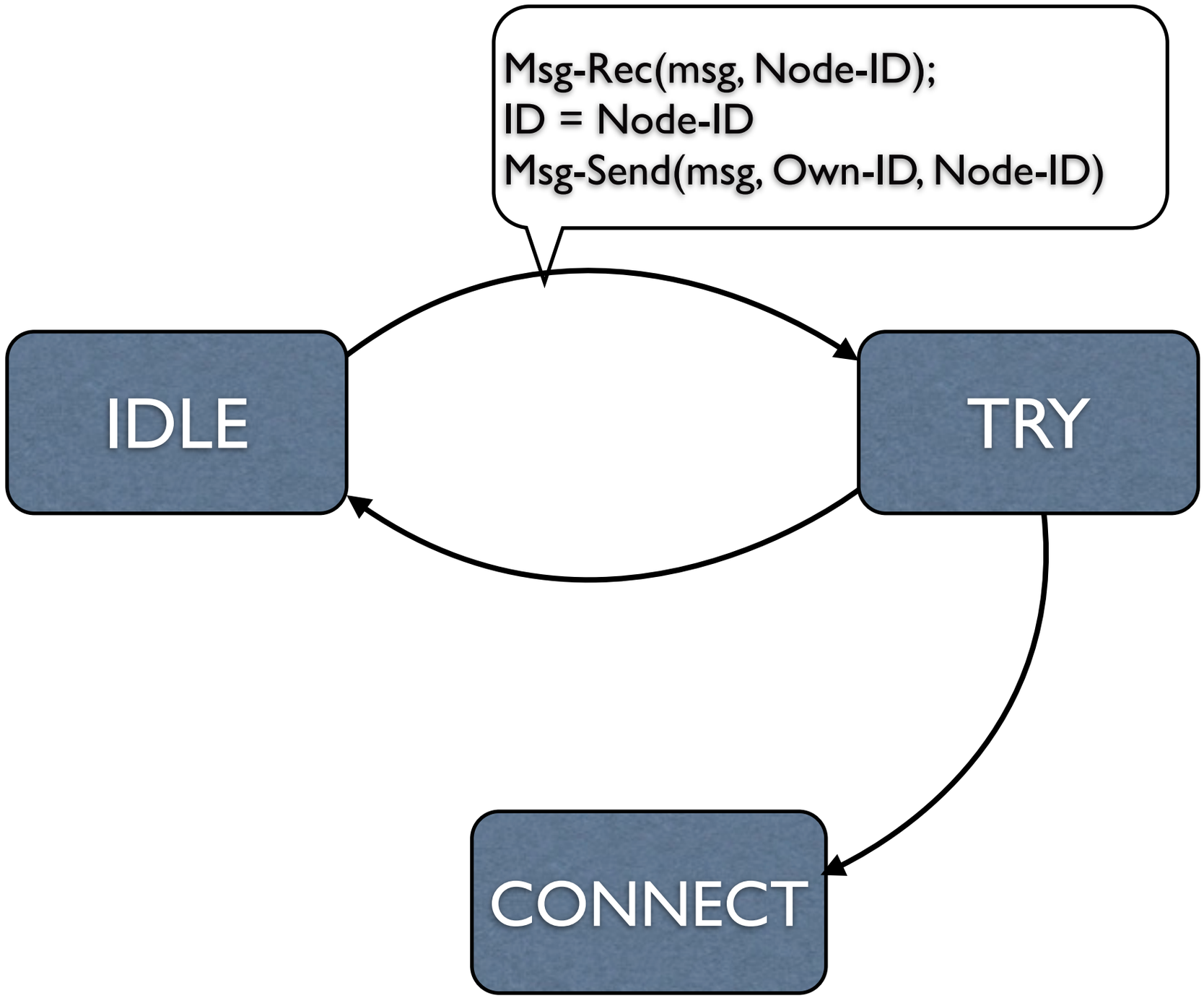
Introduction

- We specifically consider **communication protocols**
  - Define rules from data format and transmission
  - Consist of entities that interact via message passing
  - Messages: type + parameters

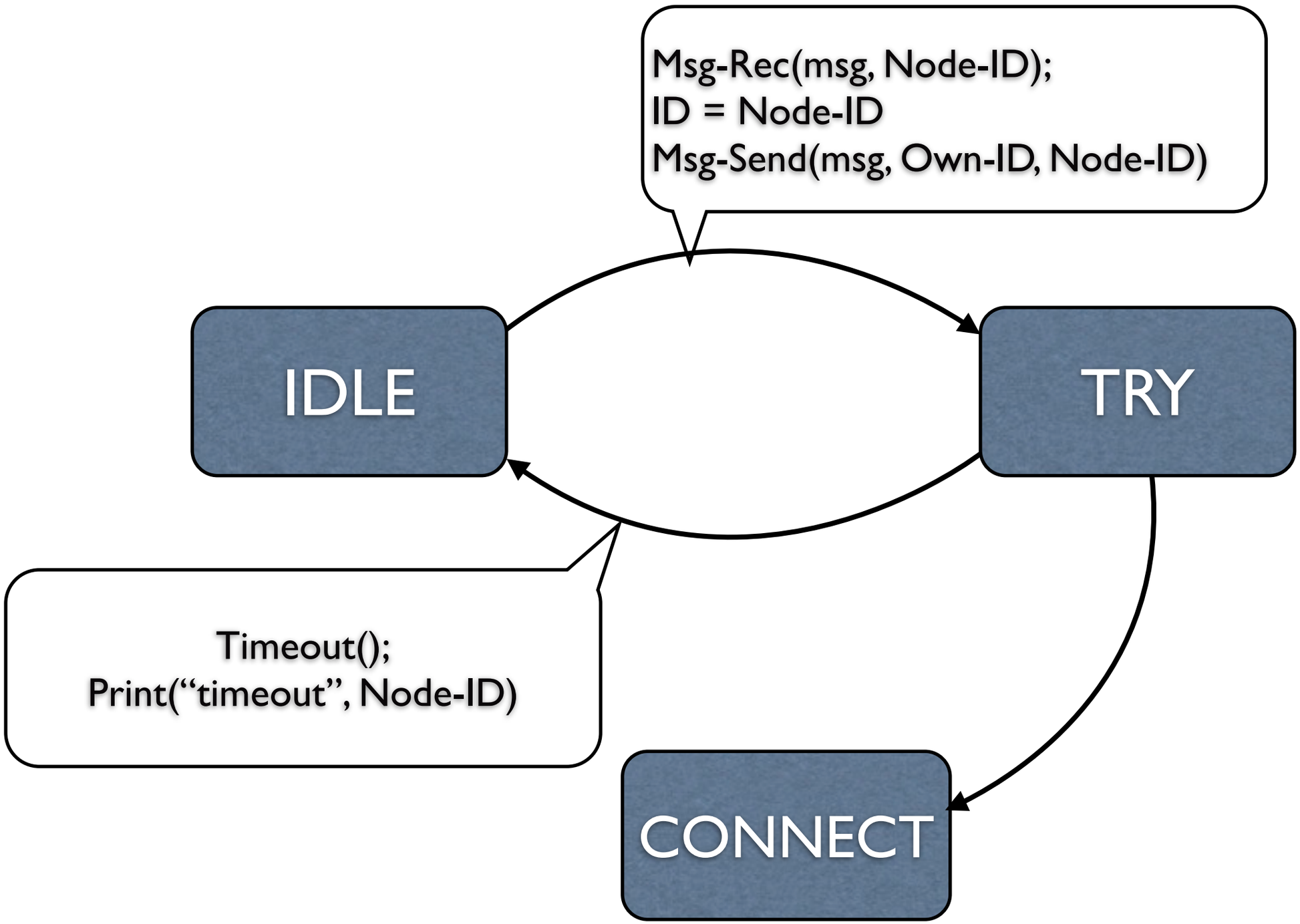
Protocol Example



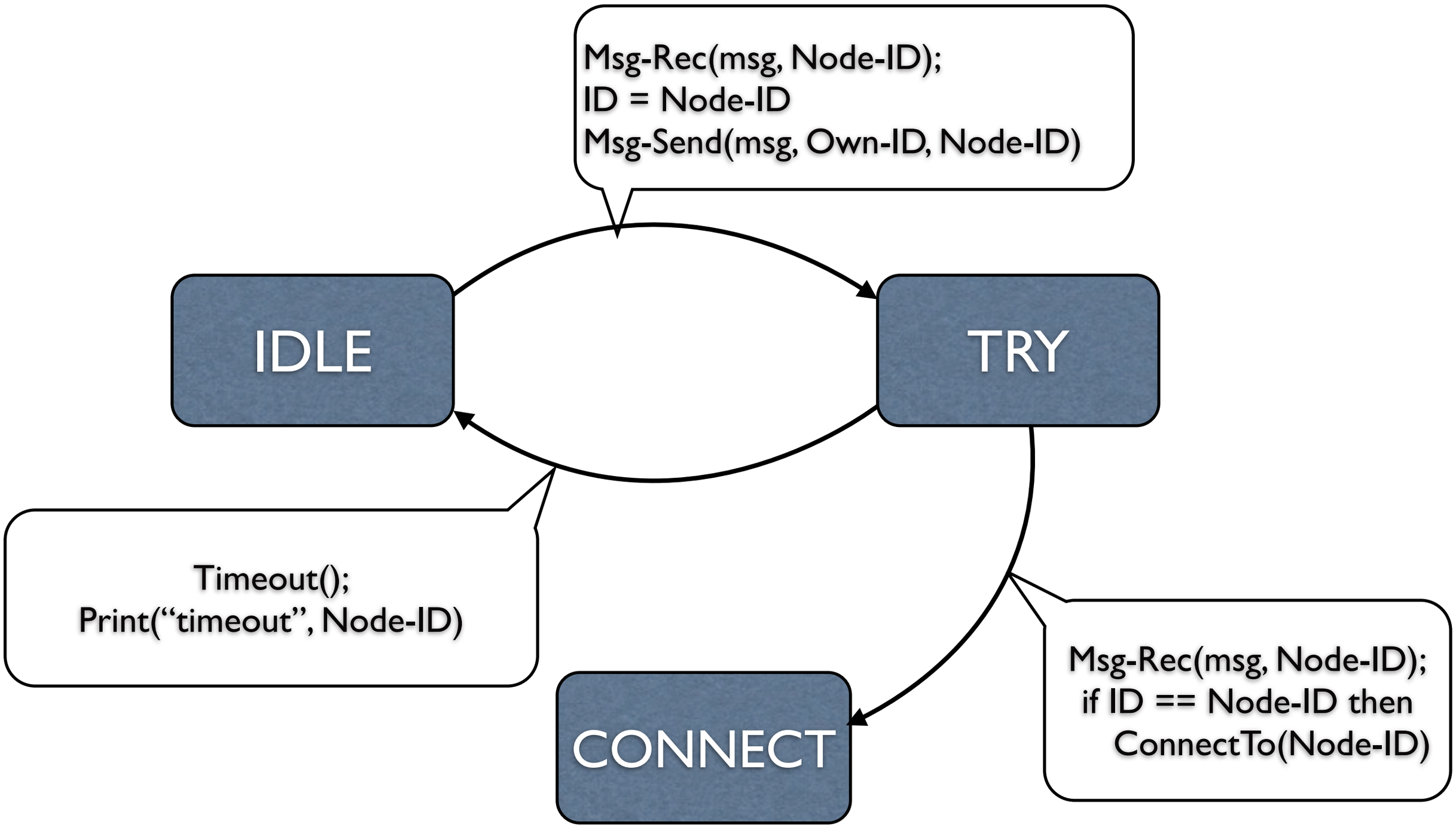
Protocol Example



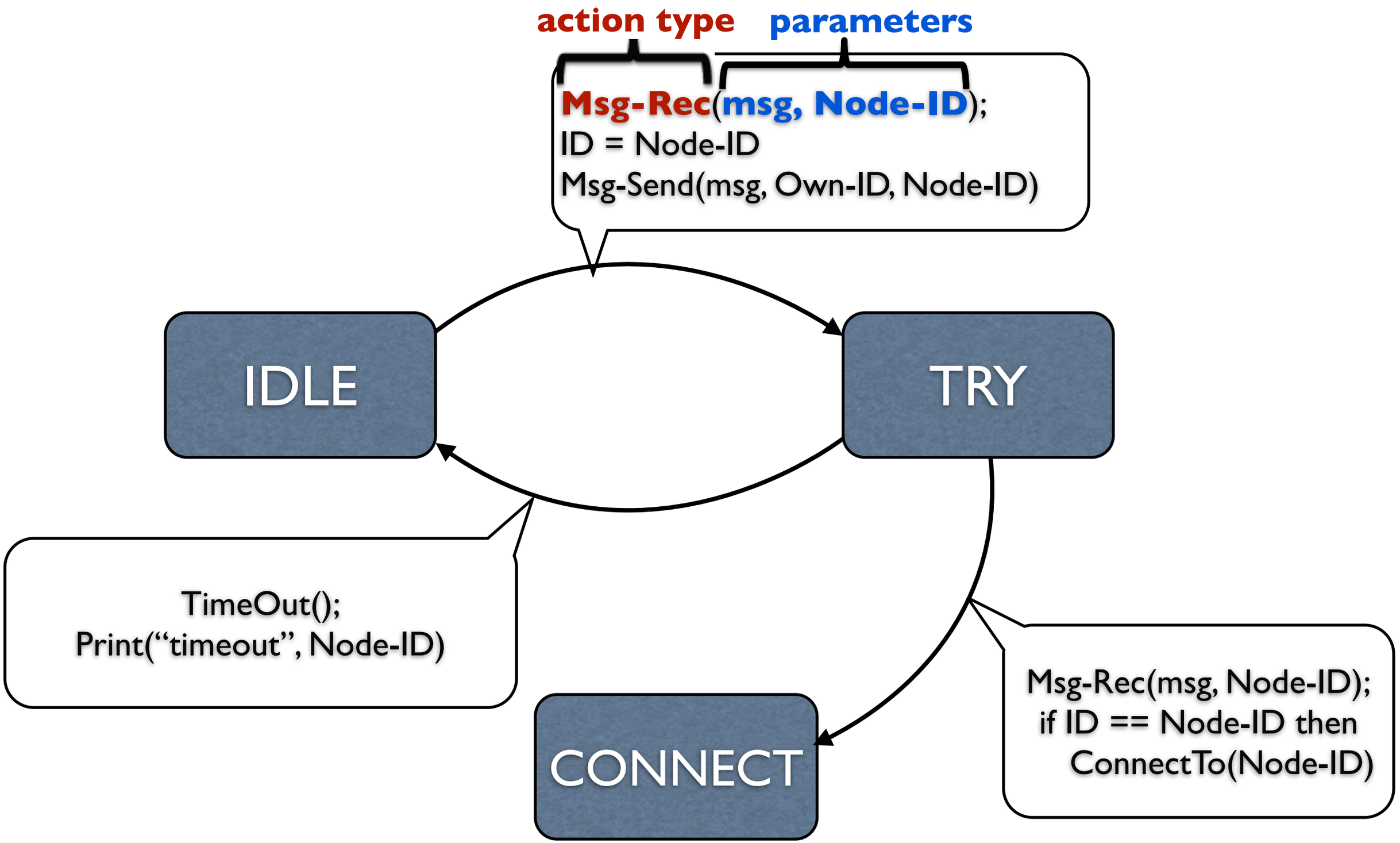
Protocol Example



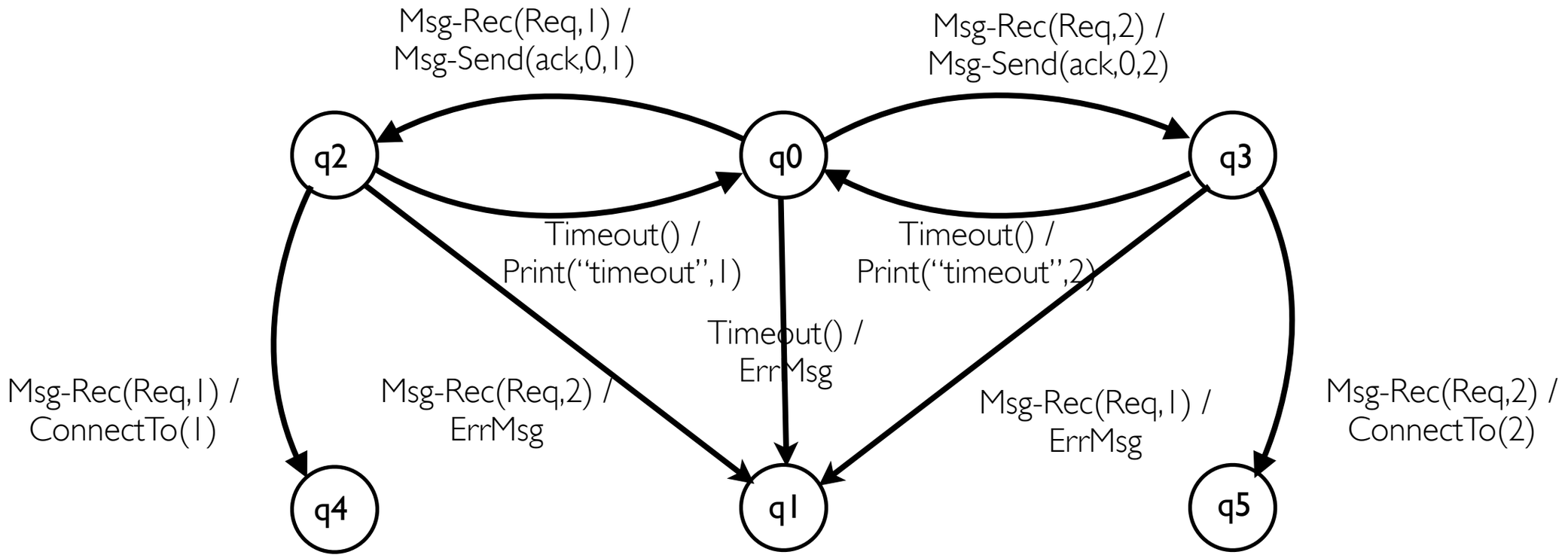
Protocol Example



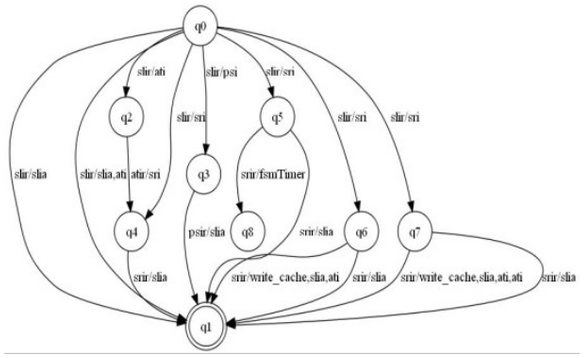
Protocol Example

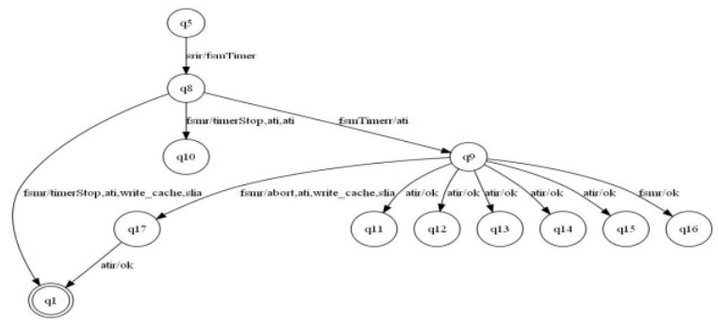


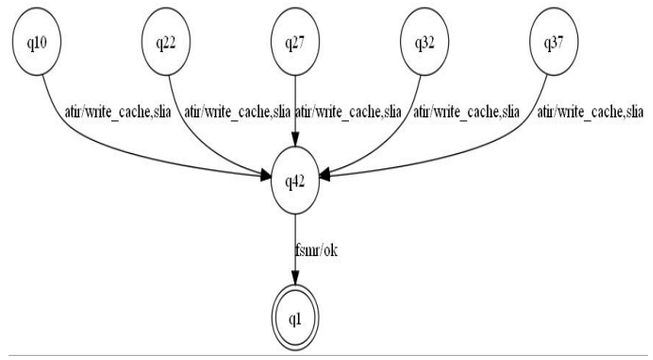
Existing Methods Infer Flat Models

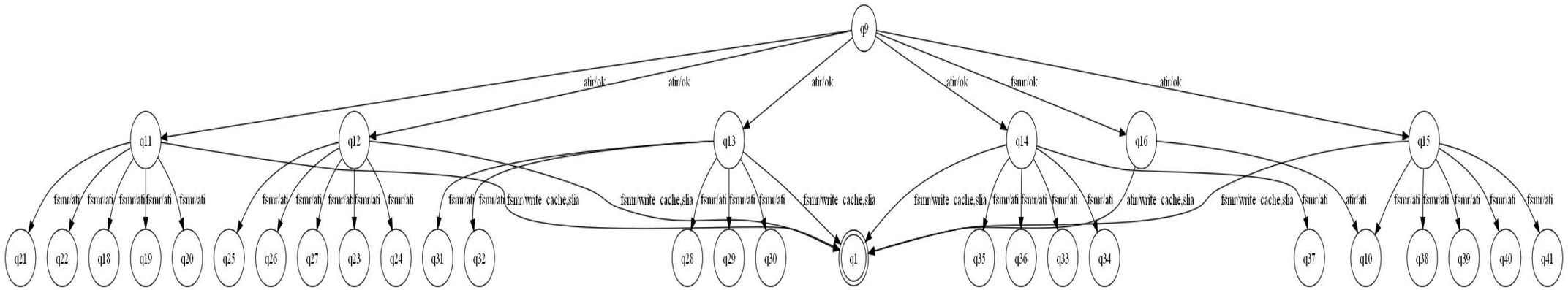


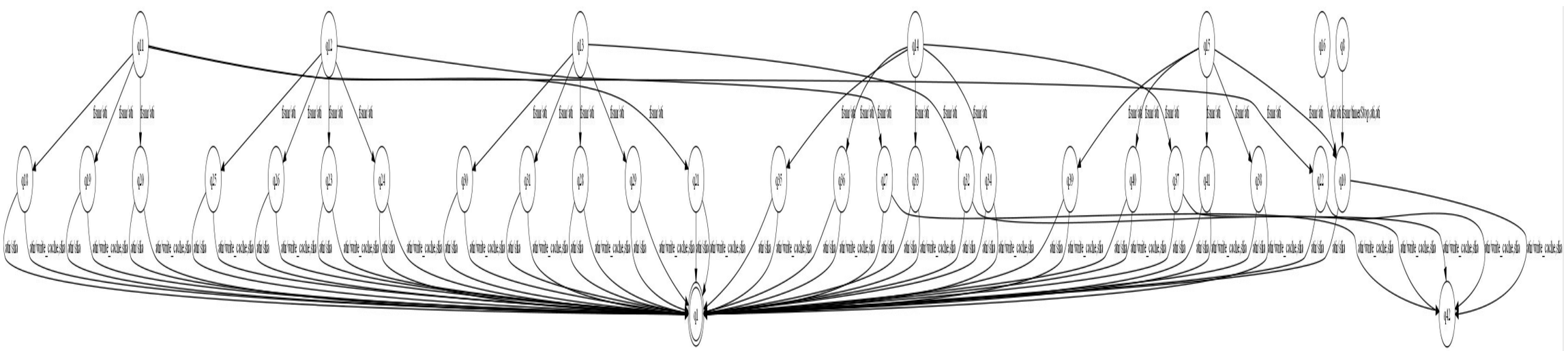
Inferred Model of a System









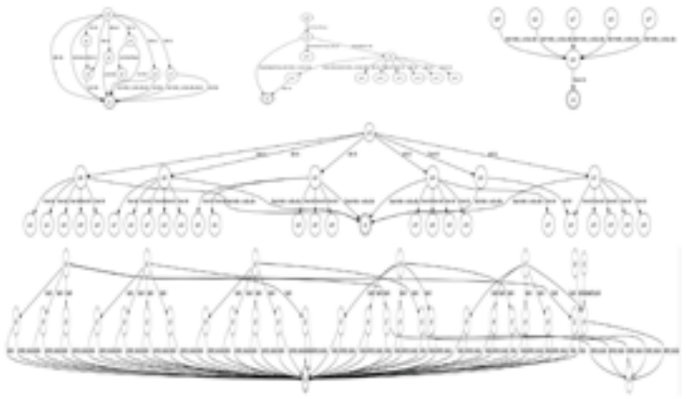


Our Contribution

Learning an understandable model of communication protocol entities by:

(I) regular inference

for modeling



(II) symbolic

representation

Outline of the talk

- (I)regular inference
- (II)symbolic representation of Mealy machine
- Evaluation and Experiments
- Conclusion and future works

Regular Inference

Is used for

- verification and test generation, e.g, [J. Cobleigh, D. Giannakopoulou, and C. Pasareanu. Learning assumptions for compositional verification, 2003]
- model checking without source code [A. Groce, D. Peled, and M. Yannakakis, Adaptive model checking, 2002]
- used for inferring Mealy machine model of a system [A. Hagerer, H. Hungar, O. Niese, and B. Steffen, Model generation by moderated regular extrapolation, 2002]
- ...

Regular Inference

Is used for

- verification and test generation, e.g, [J. Cobleigh, D. Giannakopoulou, and C. Pasareanu. Learning assumptions for compositional verification, 2003]
- model checking without source code [A. Groce, D. Peled, and M. Yannakakis, Adaptive model checking, 2002]
- used for inferring Mealy machine model of a system [A. Hagerer, H. Hungar, O. Niese, and B. Steffen, Model generation by moderated regular extrapolation, 2002]
- ...

We use

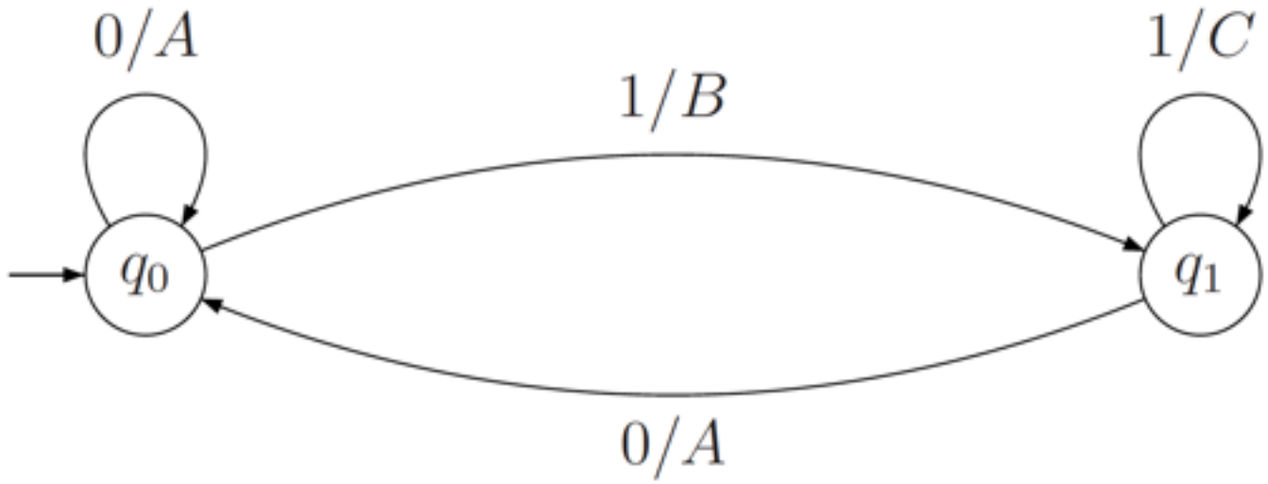
- Angluin's L^* algorithm, Niese 2003
  - Mealy machine models

Mealy Machine Model

- A Mealy machine is $\mathcal{M} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, \lambda \rangle$

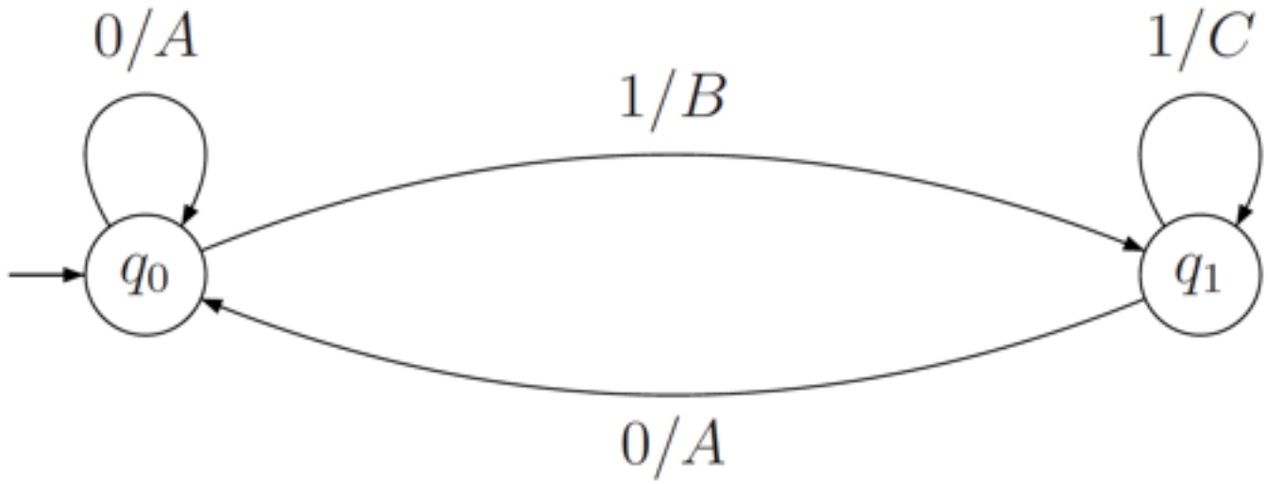
Mealy Machine Model

- A Mealy machine is $\mathcal{M} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, \lambda \rangle$



Mealy Machine Model

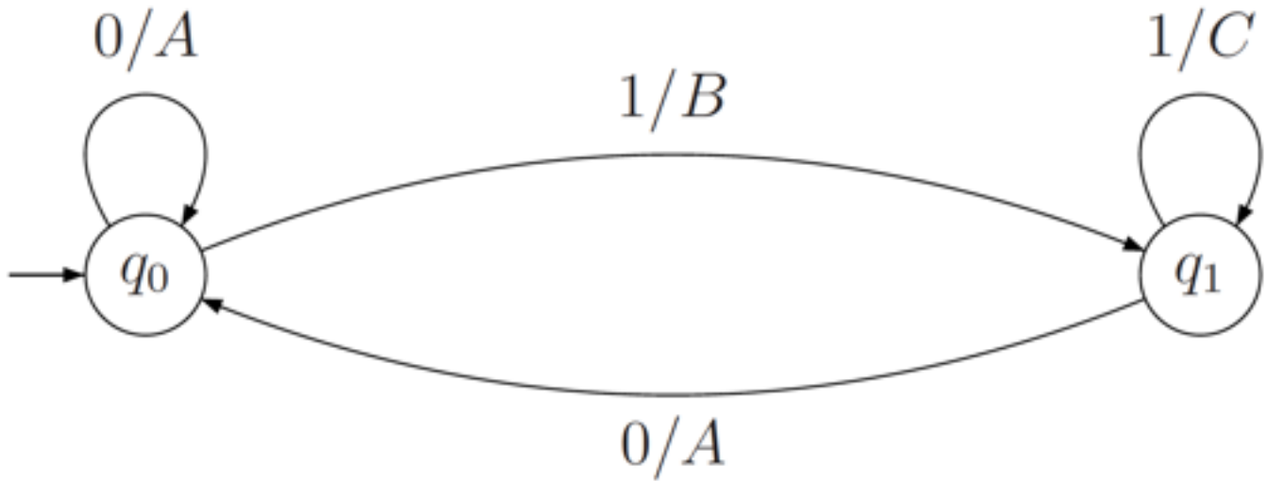
- A Mealy machine is $\mathcal{M} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, \lambda \rangle$



$$\mathcal{M} = \langle \{0, 1\}, \{A, B, C\}, \{q_0, q_1\}, q_0, \delta, \lambda, \rangle$$

Mealy Machine Model

- A Mealy machine is $\mathcal{M} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, \lambda \rangle$

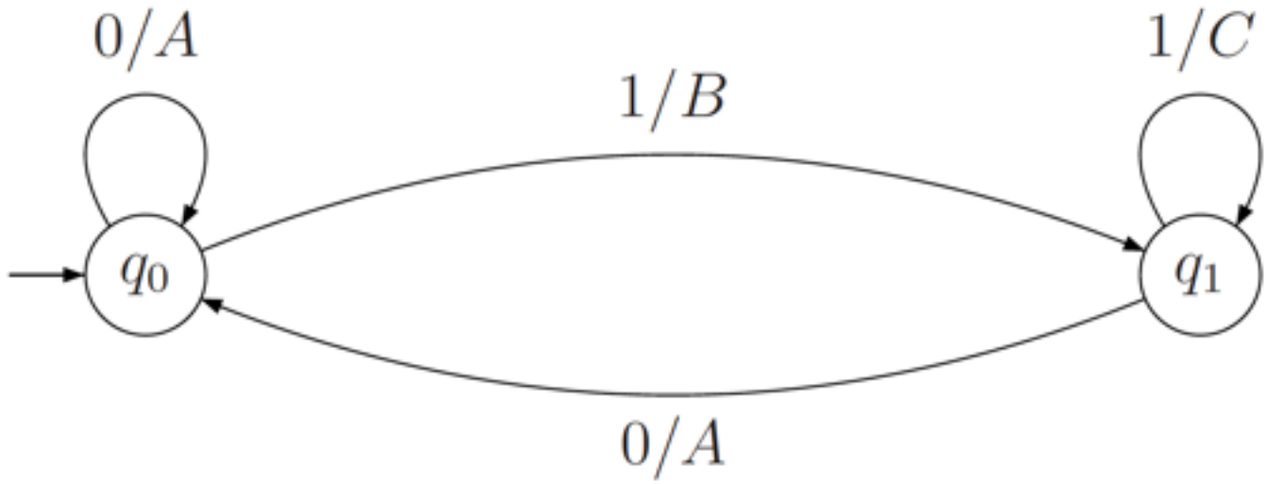


$$\mathcal{M} = \langle \{0,1\}, \{A,B,C\}, \{q_0, q_1\}, q_0, \delta, \lambda, \rangle$$

$$\delta(q_0, 0) = q_0, \quad \delta(q_0, 1) = q_1$$
$$\delta(q_1, 0) = q_0, \quad \delta(q_1, 1) = q_1$$

Mealy Machine Model

- A Mealy machine is $\mathcal{M} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, \lambda \rangle$



$$\mathcal{M} = \langle \{0,1\}, \{A,B,C\}, \{q_0, q_1\}, q_0, \delta, \lambda, \rangle$$

$$\begin{array}{llll} \delta(q_0,0) = q_0, & \delta(q_0,1) = q_1 & \lambda(q_0,0) = A, & \lambda(q_0,1) = B \\ \delta(q_1,0) = q_0, & \delta(q_1,1) = q_1 & \lambda(q_1,0) = A, & \lambda(q_1,1) = C. \end{array}$$

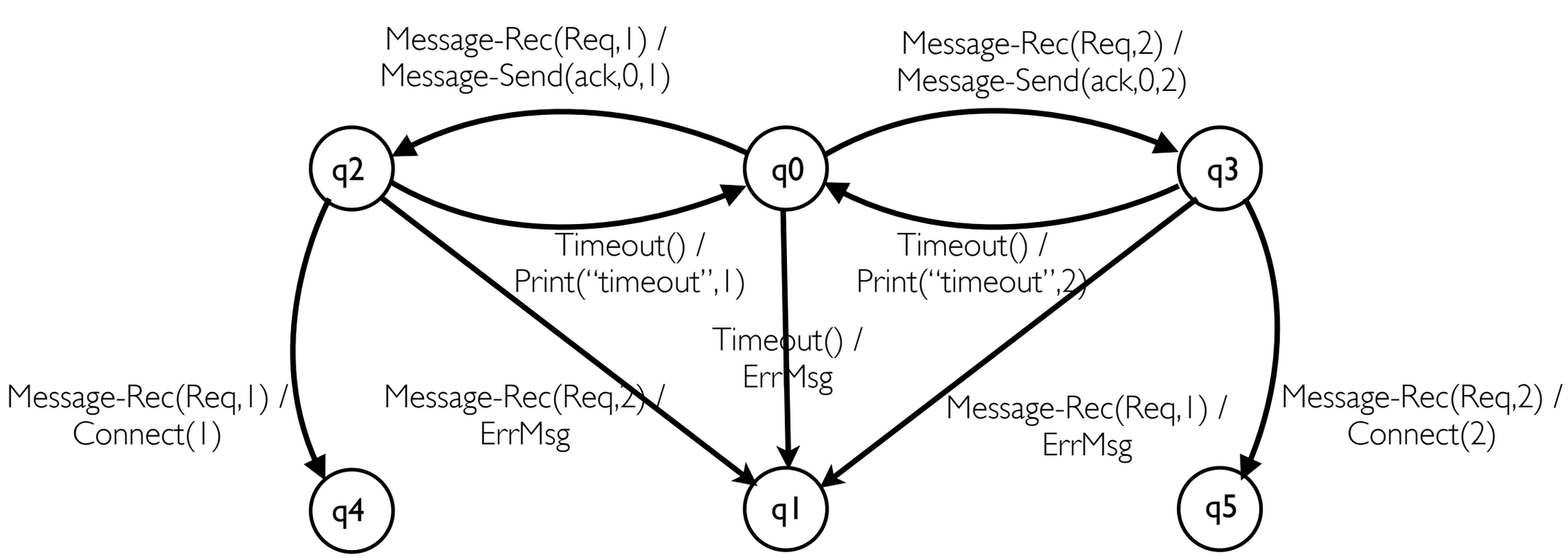
Inferring a Mealy Machine

- An existing algorithm, Niese adaptation of L^*

Inferring a Mealy Machine

- An existing algorithm, Niese adaptation of L^*

The Flat Inferred Mealy Machine Model of the Example



Symbolic Mealy Machine

- States

- Action Expressions

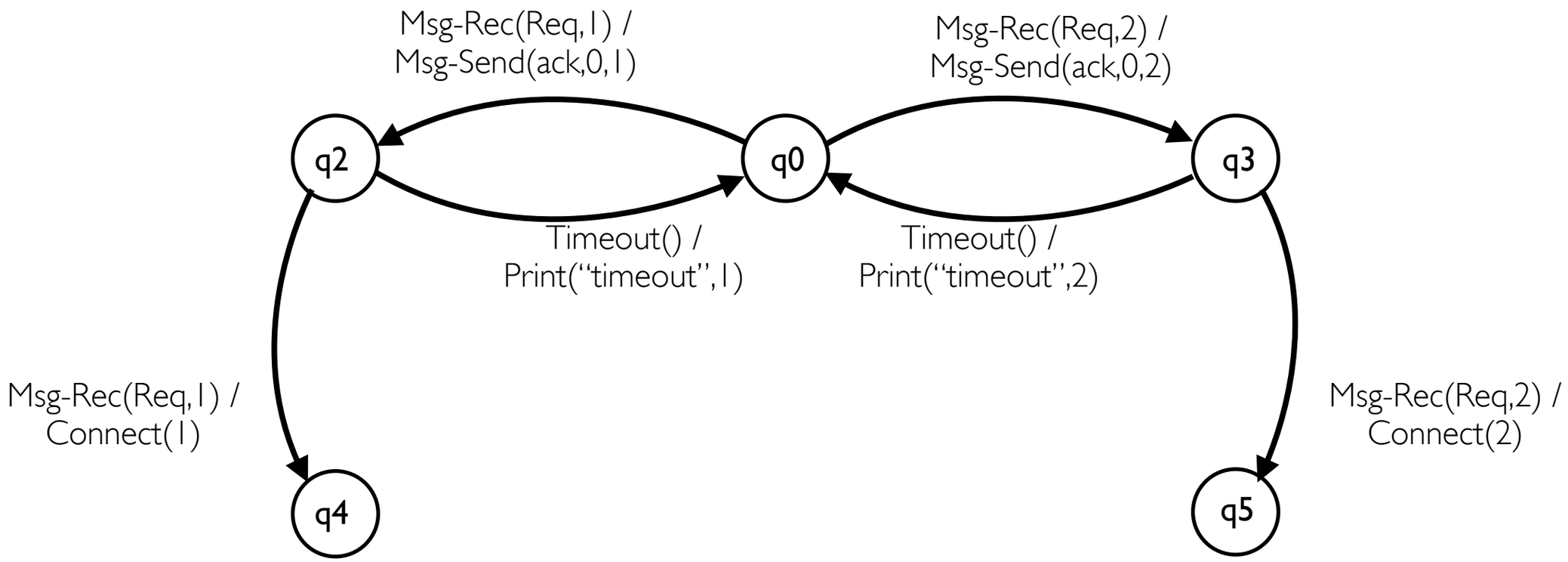
Symbolic Mealy Machine

- States: transformed at
  - control location
  - state variables
- Action Expressions: reaction to input message
  - for each location and each input action type

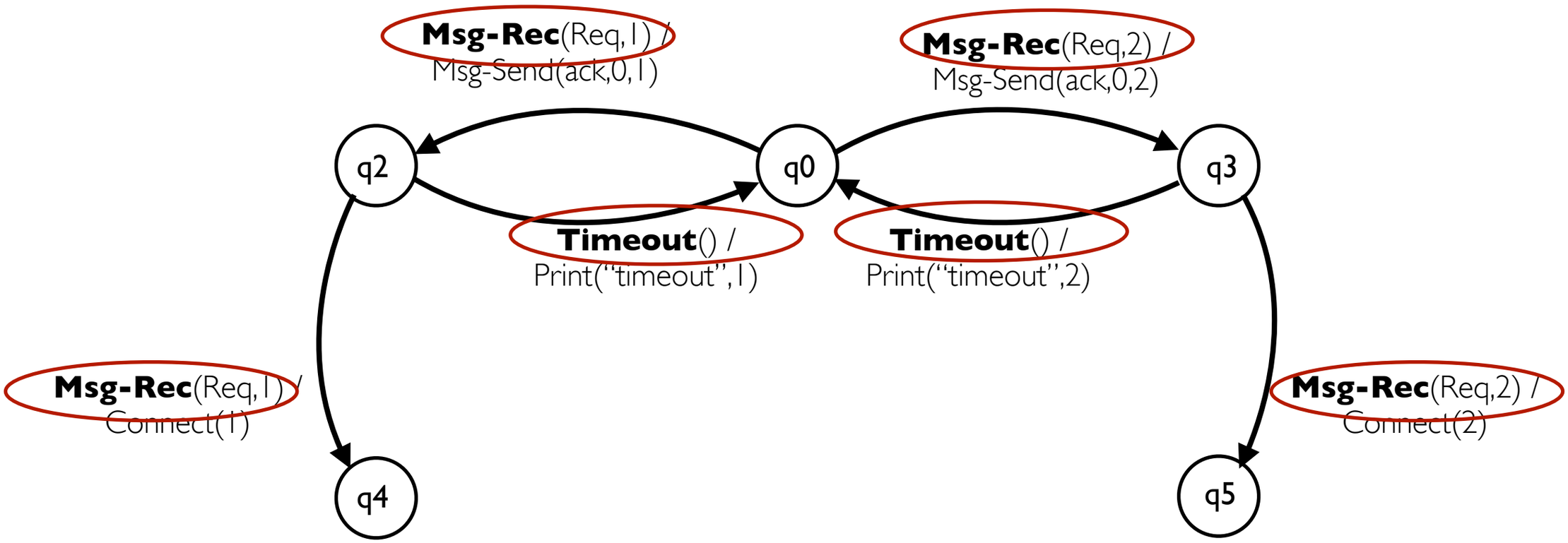
Symbolic Mealy Machine

- States: transformed at
  - control location
    - same sequence of input/output action types leading to them
  - state variables
    - record most recent value of each parameter
- Action Expressions: reaction to input message
  - for each location and each input action type
    - construct a decision tree from state variables & input parameters
      - transfer decision tree into a code-like syntax

State variables



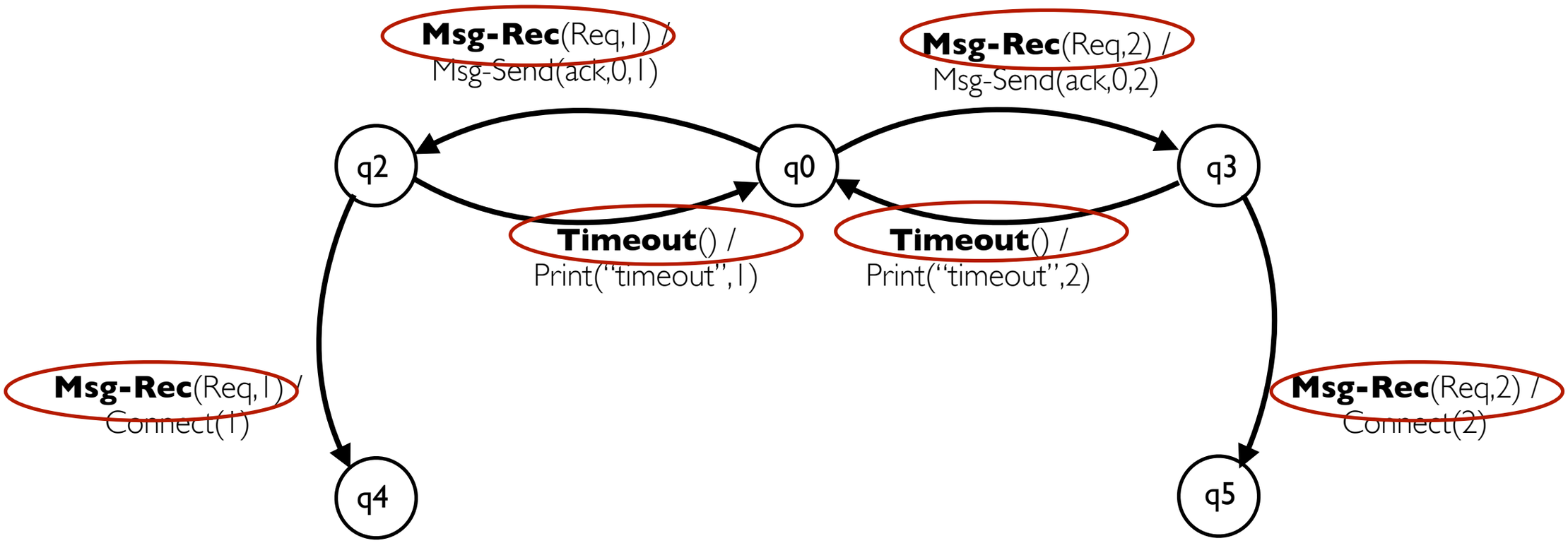
State variables



Distinct input action types

Msg-Rec(par1,par2) Timeout()

State variables

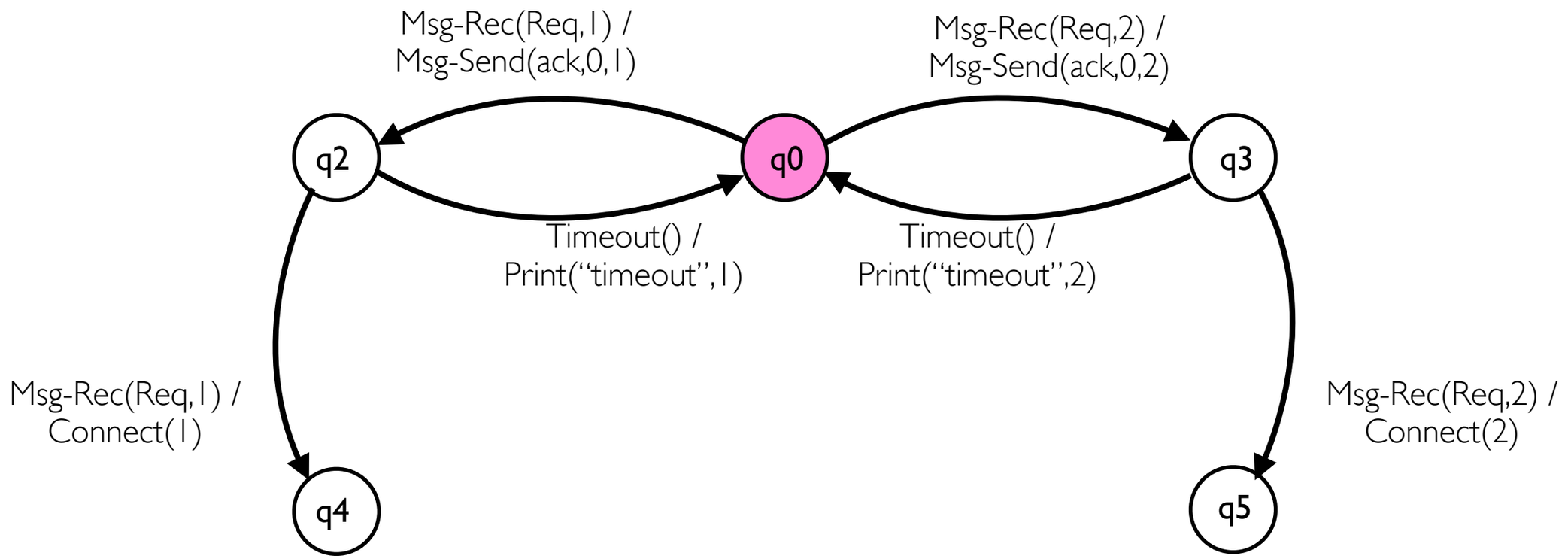


Distinct input action types

Msg-Rec(par1,par2) Timeout()

State variables = {V1msg, V2msg, Vt}

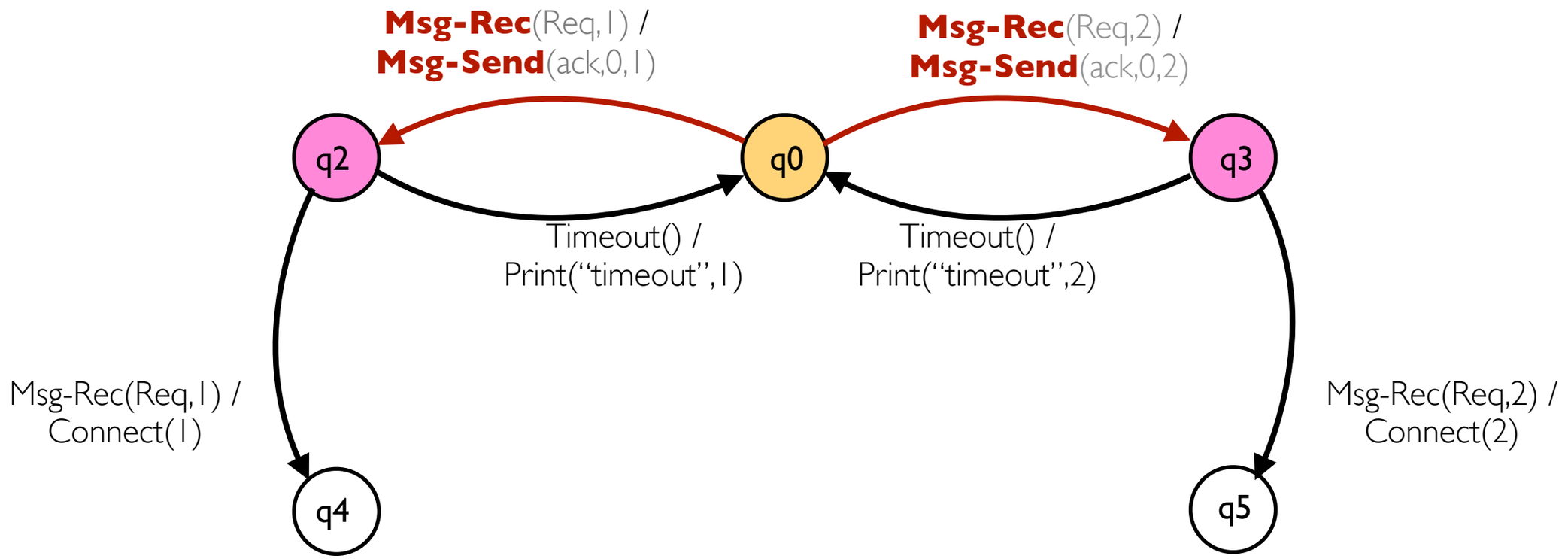
Location Construction



L0

q0: V1msg=None,V2msg=None,Vt=None

Location Construction



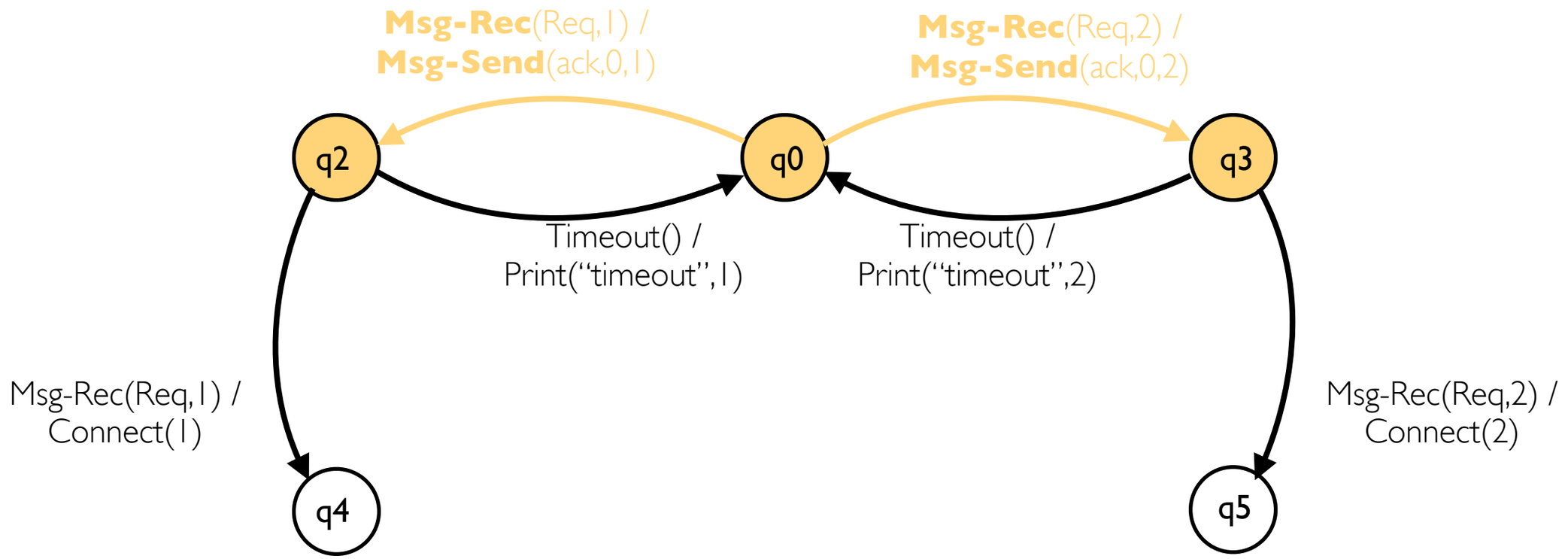
L0

q0: V1msg=None,V2msg=None,Vt=None

L1

q2: V1msg=Req,V2msg=1,Vt=None
q3: V1msg=Req,V2msg=2,Vt=None

Location Construction



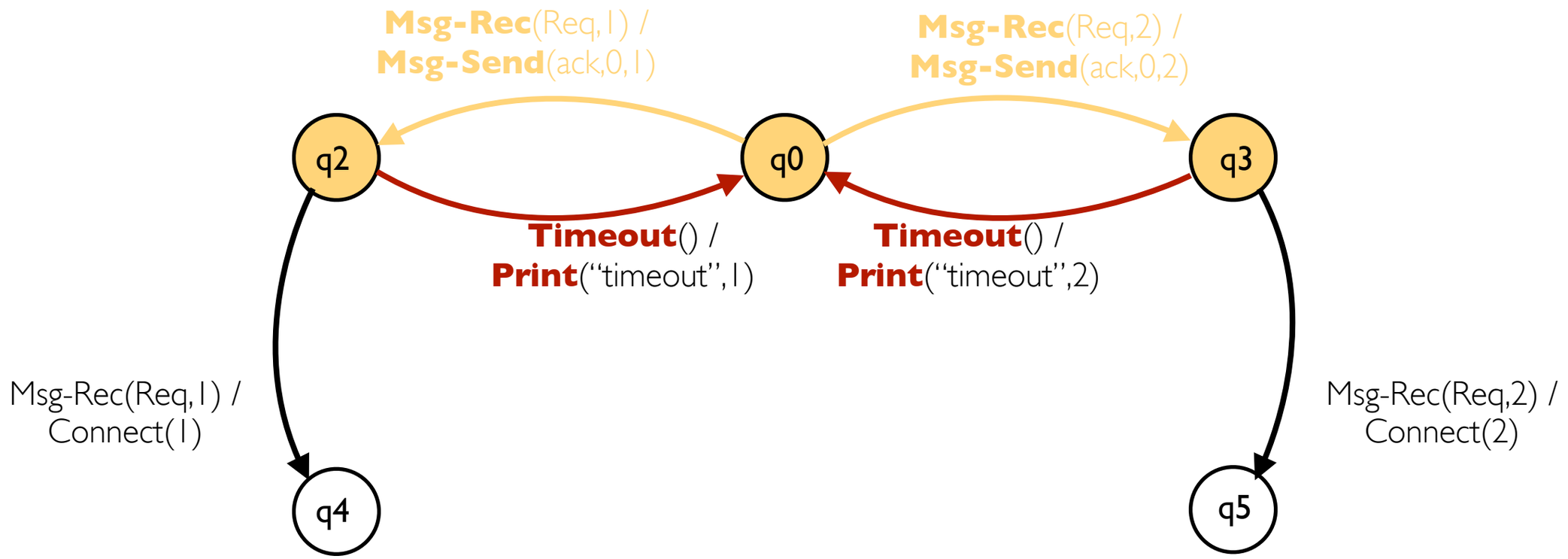
L0

q0: V1msg=None,V2msg=None,Vt=None

L1

q2: V1msg=Req,V2msg=1,Vt=None
q3: V1msg=Req,V2msg=2,Vt=None

Location Construction



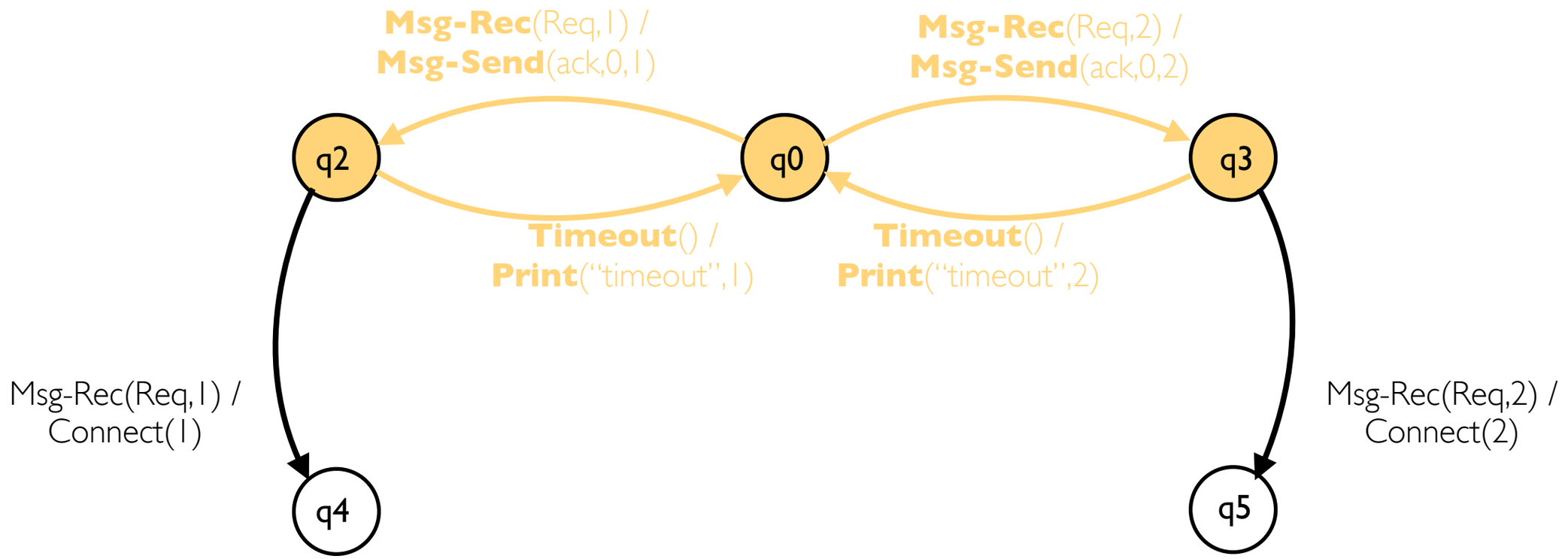
L0

q0: V1msg=None,V2msg=None,Vt=None
q0: V1msg=None,V2msg=None,Vt=True

L1

q2: V1msg=Req,V2msg=1,Vt=None
q3: V1msg=Req,V2msg=2,Vt=None

Location Construction



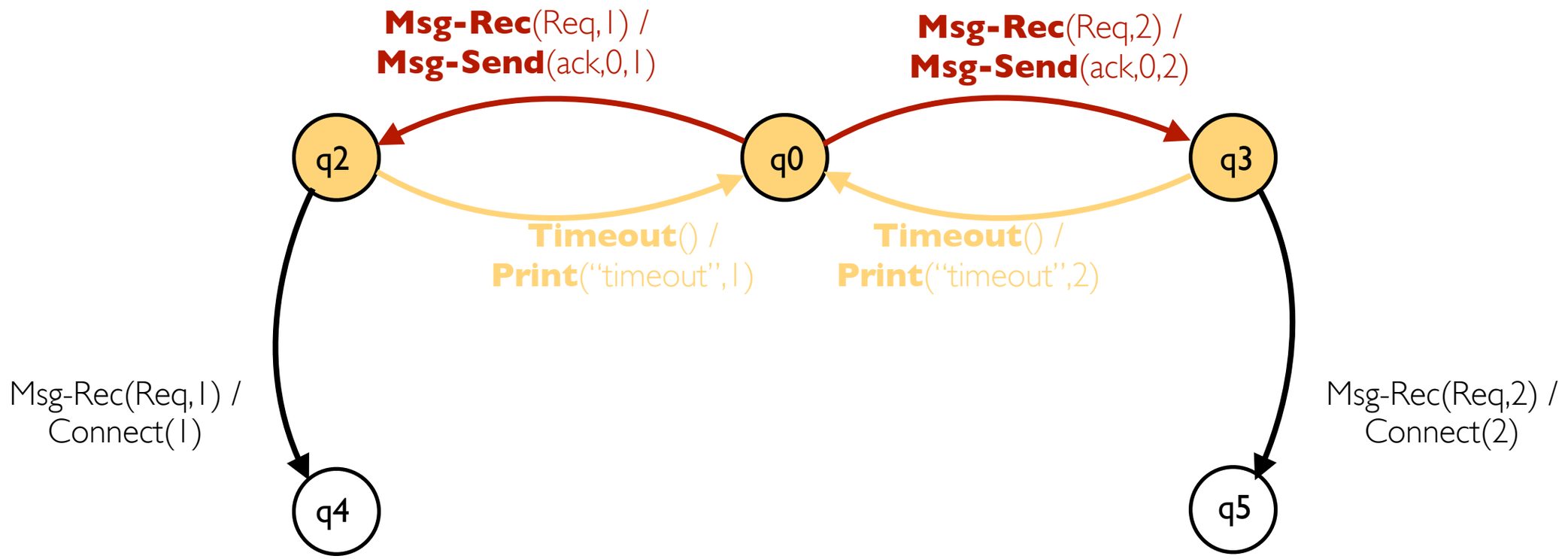
L0

q0: V1msg=None,V2msg=None,Vt=None
q0: V1msg=None,V2msg=None,Vt=True

L1

q2: V1msg=Req,V2msg=1,Vt=None
q3: V1msg=Req,V2msg=2,Vt=None

Location Construction



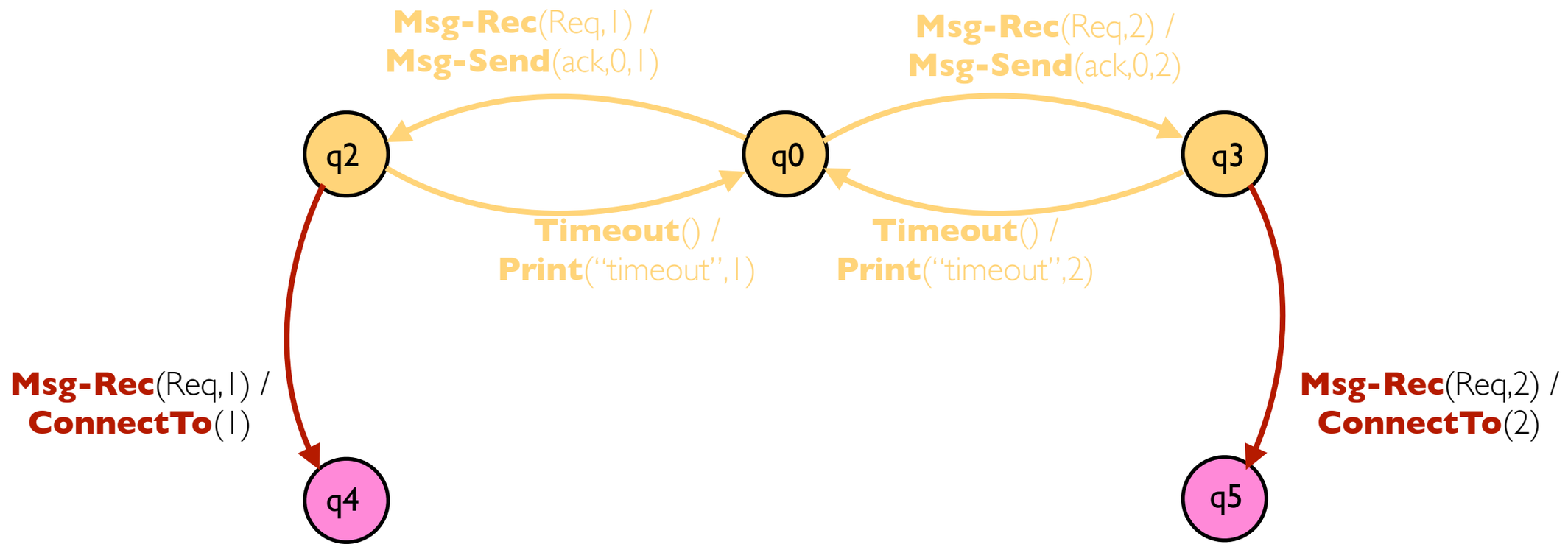
L0

q0: V1msg=None,V2msg=None,Vt=None
q0: V1msg=None,V2msg=None,Vt=True

L1

q2: V1msg=Req,V2msg=1,Vt=None
q3: V1msg=Req,V2msg=2,Vt=None
q2: V1msg=Req,V2msg=1,Vt=True
q3: V1msg=Req,V2msg=2,Vt=True

Location Construction



L0

q0: V1msg=None,V2msg=None,Vt=None
q0: V1msg=None,V2msg=None,Vt=True

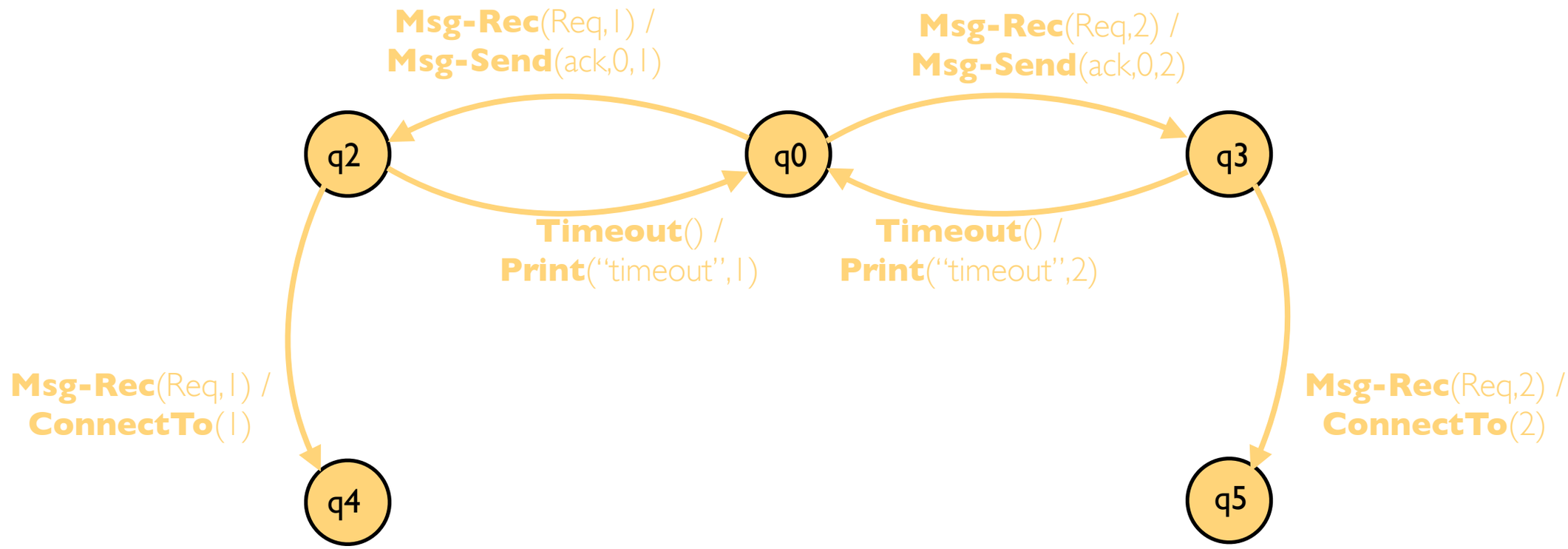
L1

q2: V1msg=Req,V2msg=1,Vt=None
q3: V1msg=Req,V2msg=2,Vt=None
q2: V1msg=Req,V2msg=1,Vt=True
q3: V1msg=Req,V2msg=2,Vt=True

L2

q4: V1msg=Req,V2msg=1,Vt=None
q5: V1msg=Req,V2msg=2,Vt=None
q4: V1msg=Req,V2msg=1,Vt=True
q5: V1msg=Req,V2msg=2,Vt=True

Location Construction



L0

q0: V1msg=None,V2msg=None,Vt=None
q0: V1msg=None,V2msg=None,Vt=True

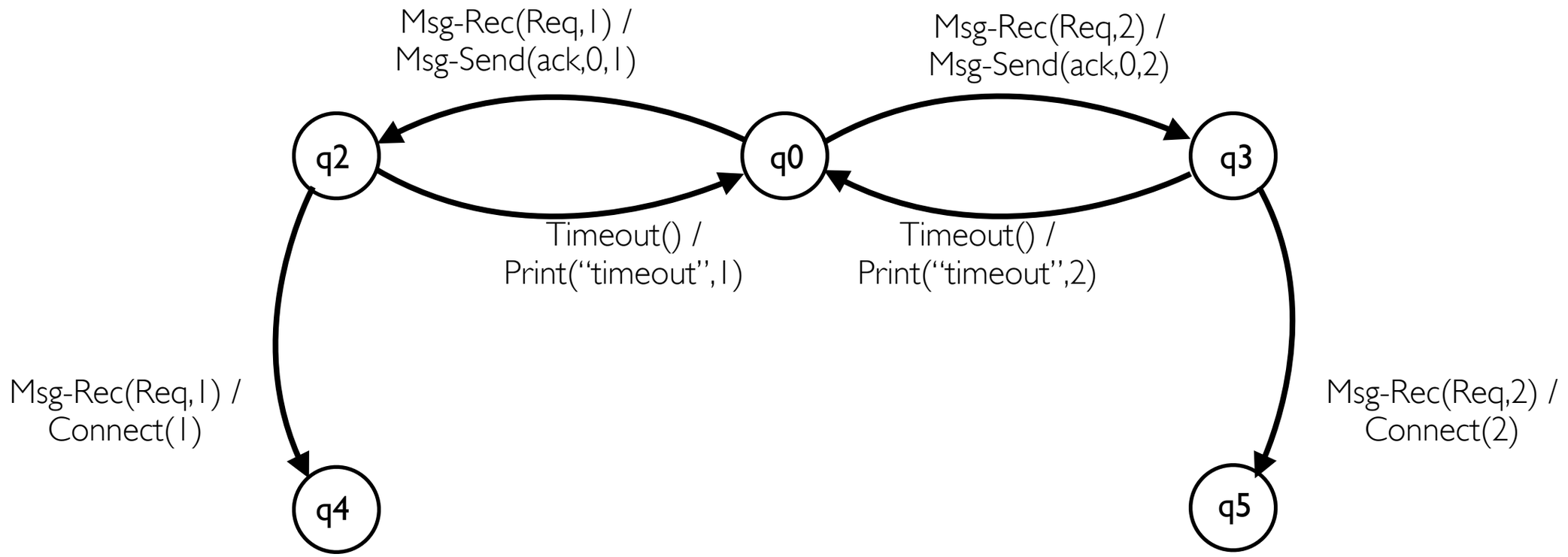
L1

q2: V1msg=Req,V2msg=1,Vt=None
q3: V1msg=Req,V2msg=2,Vt=None
q2: V1msg=Req,V2msg=1,Vt=True
q3: V1msg=Req,V2msg=2,Vt=True

L2

q4: V1msg=Req,V2msg=1,Vt=None
q5: V1msg=Req,V2msg=2,Vt=None
q4: V1msg=Req,V2msg=1,Vt=True
q5: V1msg=Req,V2msg=2,Vt=True

Action Expressions



L0

q0: V1msg=None,V2msg=None,Vt=None
q0: V1msg=None,V2msg=None,Vt=True

L1

q2: V1msg=Req,V2msg=1,Vt=None
q3: V1msg=Req,V2msg=2,Vt=None
q2: V1msg=Req,V2msg=1,Vt=True
q3: V1msg=Req,V2msg=2,Vt=True

L2

q4: V1msg=Req,V2msg=1,Vt=None
q5: V1msg=Req,V2msg=2,Vt=None
q4: V1msg=Req,V2msg=1,Vt=True
q5: V1msg=Req,V2msg=2,Vt=True

Action Expressions

L0

q0: V1msg=None,V2msg=None,Vt=None
q0: V1msg=None,V2msg=None,Vt=True

L1

q2: V1msg=Req,V2msg=1,Vt=None
q3: V1msg=Req,V2msg=2,Vt=None
q2: V1msg=Req,V2msg=1,Vt=True
q3: V1msg=Req,V2msg=2,Vt=True

L2

q4: V1msg=Req,V2msg=1,Vt=None
q5: V1msg=Req,V2msg=2,Vt=None
q4: V1msg=Req,V2msg=1,Vt=True
q5: V1msg=Req,V2msg=2,Vt=True

Action Expressions

L0

q0: V1msg=None,V2msg=None,Vt=None
q0: V1msg=None,V2msg=None,Vt=True

L1

q2: V1msg=Req,V2msg=1,Vt=None
q3: V1msg=Req,V2msg=2,Vt=None
q2: V1msg=Req,V2msg=1,Vt=True
q3: V1msg=Req,V2msg=2,Vt=True

Msg-Rec / ConnectTo

L2

q4: V1msg=Req,V2msg=1,Vt=None
q5: V1msg=Req,V2msg=2,Vt=None
q4: V1msg=Req,V2msg=1,Vt=True
q5: V1msg=Req,V2msg=2,Vt=True

Action Expressions

L0

q0: V1msg=None,V2msg=None,Vt=None
q0: V1msg=None,V2msg=None,Vt=True

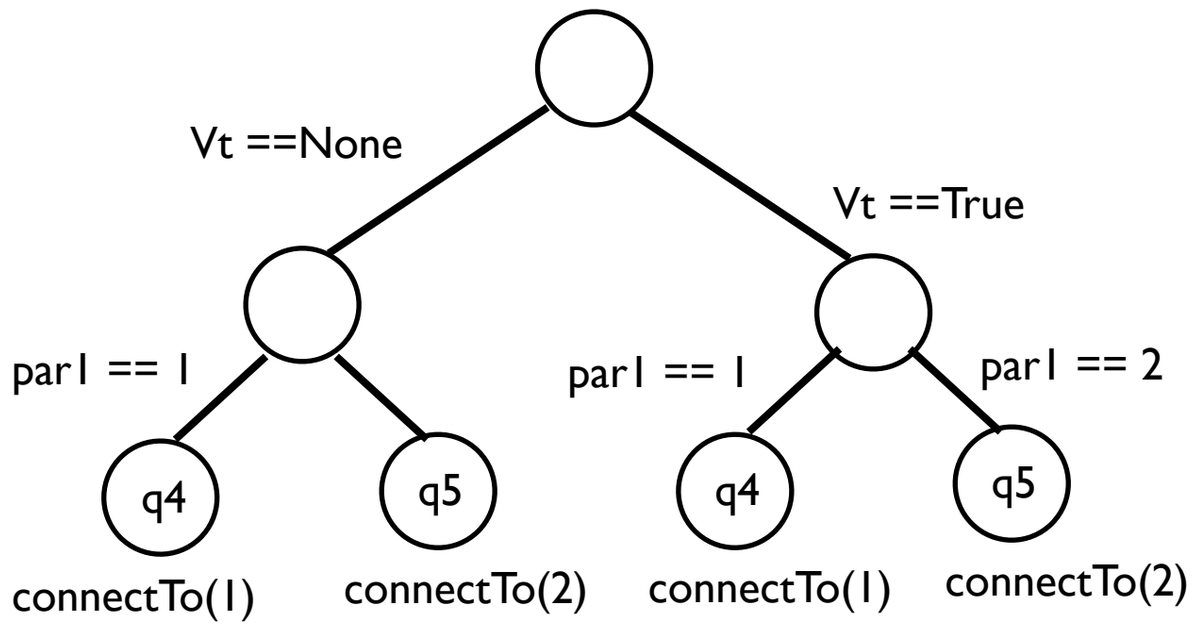
L1

q2: V1msg=Req,V2msg=1,Vt=None
q3: V1msg=Req,V2msg=2,Vt=None
q2: V1msg=Req,V2msg=1,Vt=True
q3: V1msg=Req,V2msg=2,Vt=True

Msg-Rec / ConnectTo

L2

q4: V1msg=Req,V2msg=1,Vt=None
q5: V1msg=Req,V2msg=2,Vt=None
q4: V1msg=Req,V2msg=1,Vt=True
q5: V1msg=Req,V2msg=2,Vt=True



Action Expressions

L0

q0: V1msg=None,V2msg=None,Vt=None
q0: V1msg=None,V2msg=None,Vt=True

```
in location L1
  when Msg-Rec(par1,par2)
     case Vt   of
      True:
         case par1 of
           1:
              output connectTo(1);
              nextloc L2;
           2:
              output connectTo(2);
              nextloc L2;
        endcase;
      None:
        case par1 of
           1:
              output connectTo(1);
              nextloc L2;
           2:
              output connectTo(2);
              nextloc L2;
        endcase;
    endcase;
    V1msg = par1;V2msg = par2;
end.
```

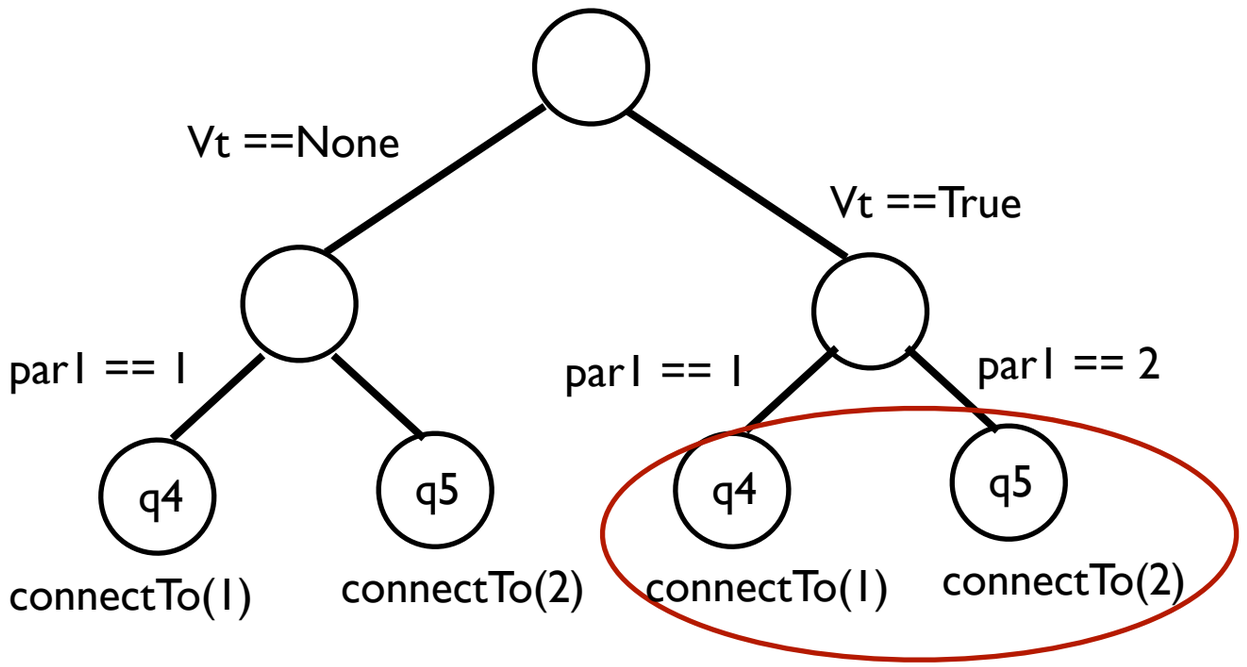
L1

q2: V1msg=Req,V2msg=1,Vt=None
q3: V1msg=Req,V2msg=2,Vt=None
q2: V1msg=Req,V2msg=1,Vt=True
q3: V1msg=Req,V2msg=2,Vt=True

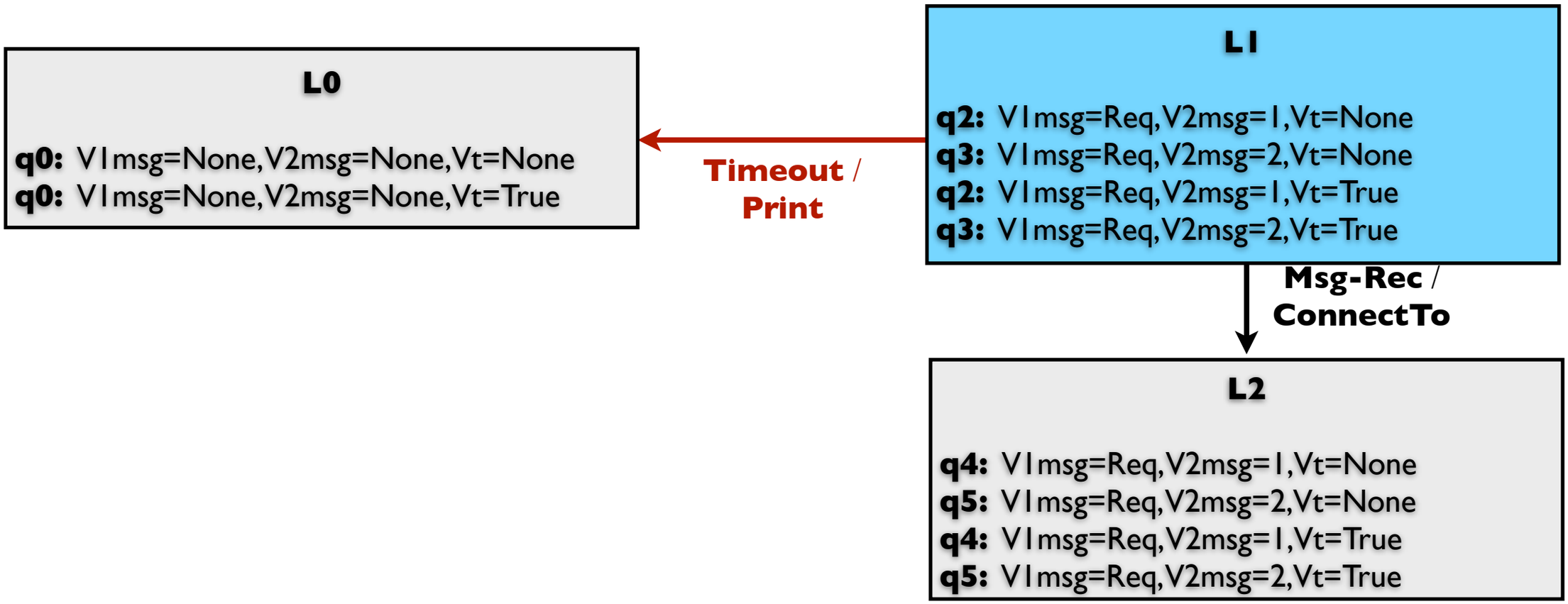
Msg-Rec / ConnectTo

L2

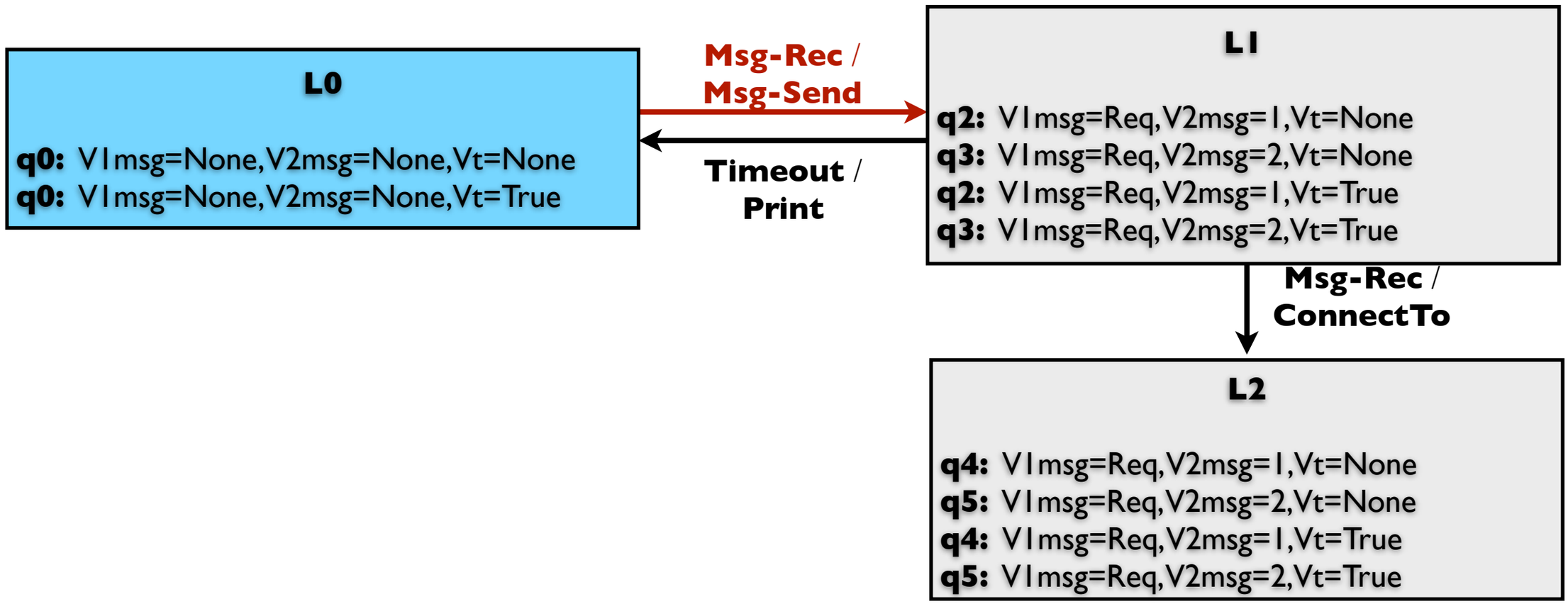
q4: V1msg=Req,V2msg=1,Vt=None
q5: V1msg=Req,V2msg=2,Vt=None
q4: V1msg=Req,V2msg=1,Vt=True
q5: V1msg=Req,V2msg=2,Vt=True



Action Expressions



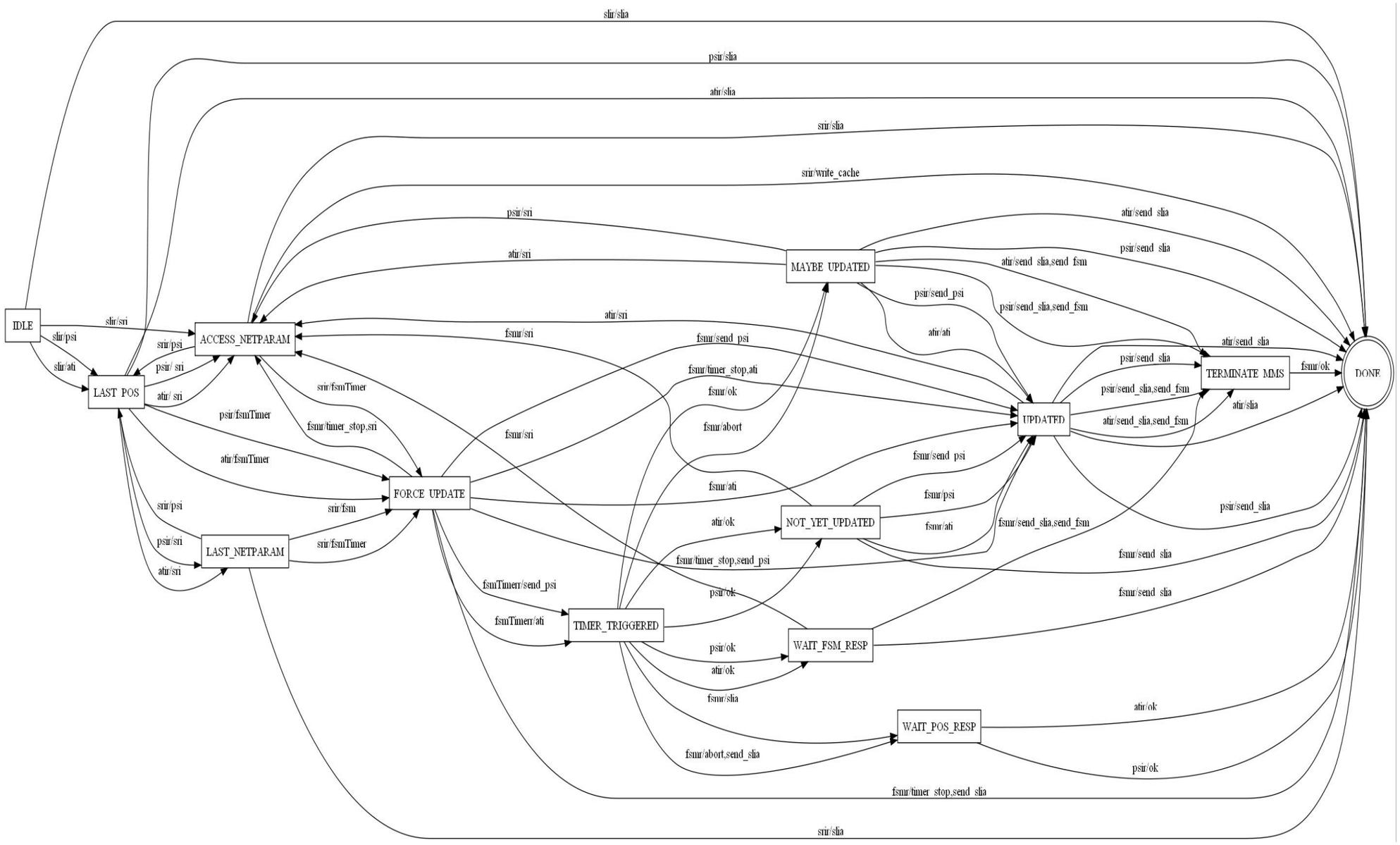
Action Expressions



Experiments

- A-MLC protocol developed by mobile arts
  - 130,000 lines of Erlang and 5,500 lines of C code
- We used executable specification of A-MLC
  - models behavior for individual client request
  - 13 control states

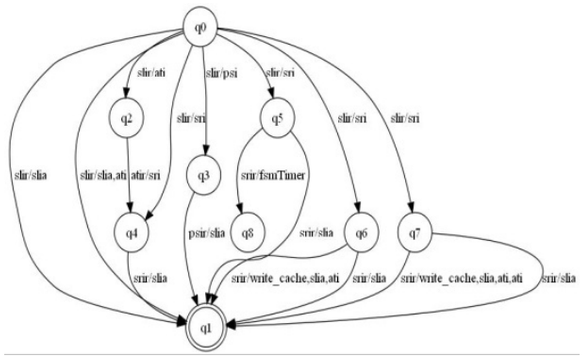
Executable Specification of A-MLC

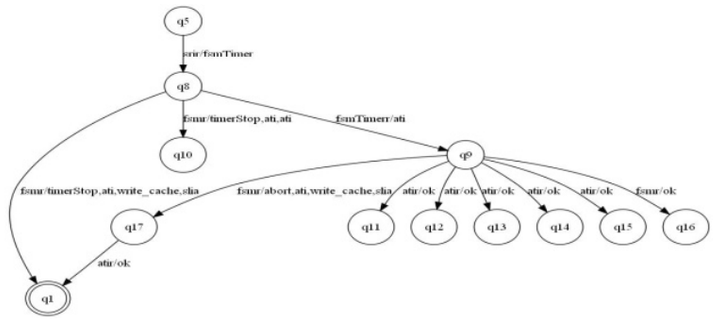


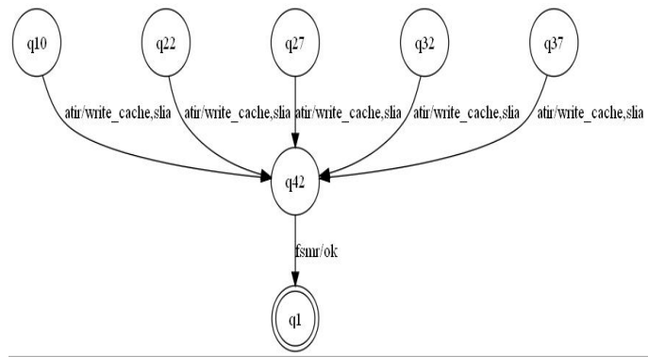
Task (I)

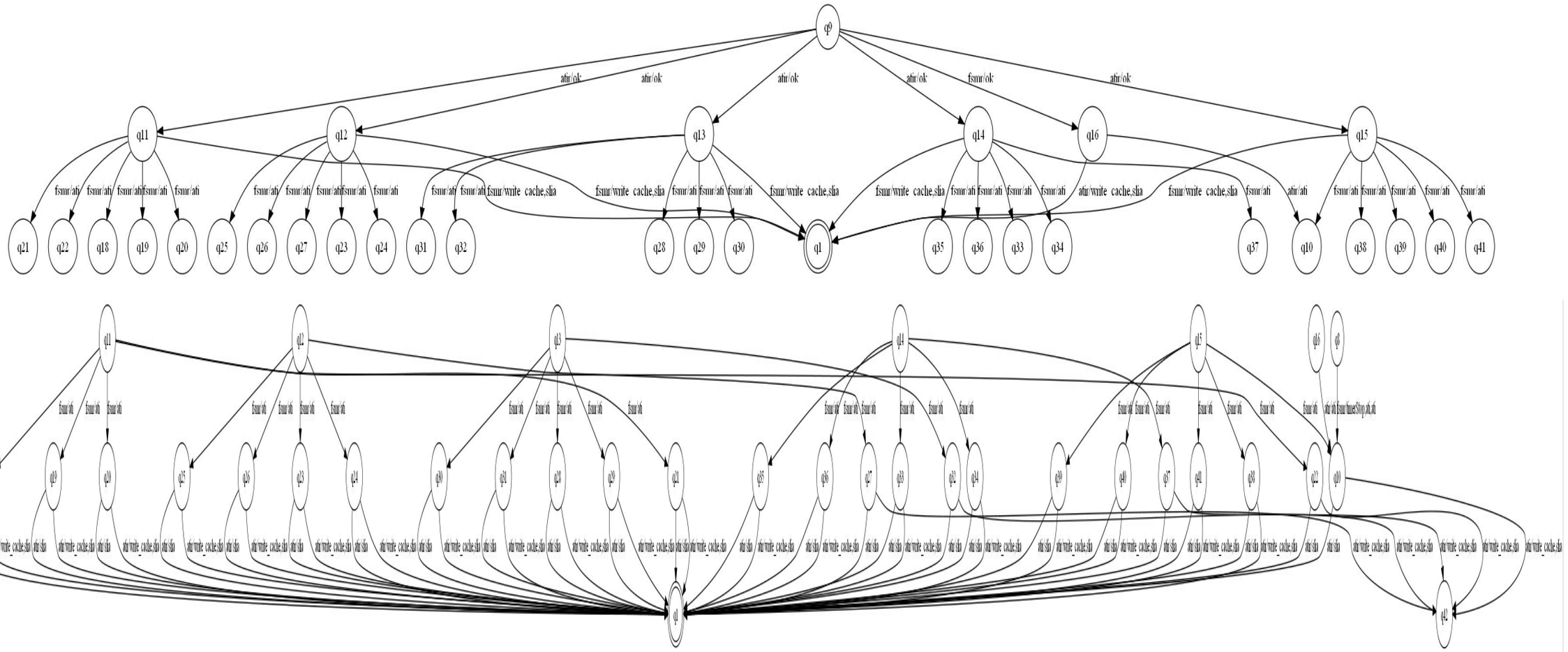
- we explored a **very small set of data values**
- **LearnLib,** efficient implementation of L^*
- 175 million membership queries in 43 hours
- the result Mealy machine has 42 states

Mealy Machine Model







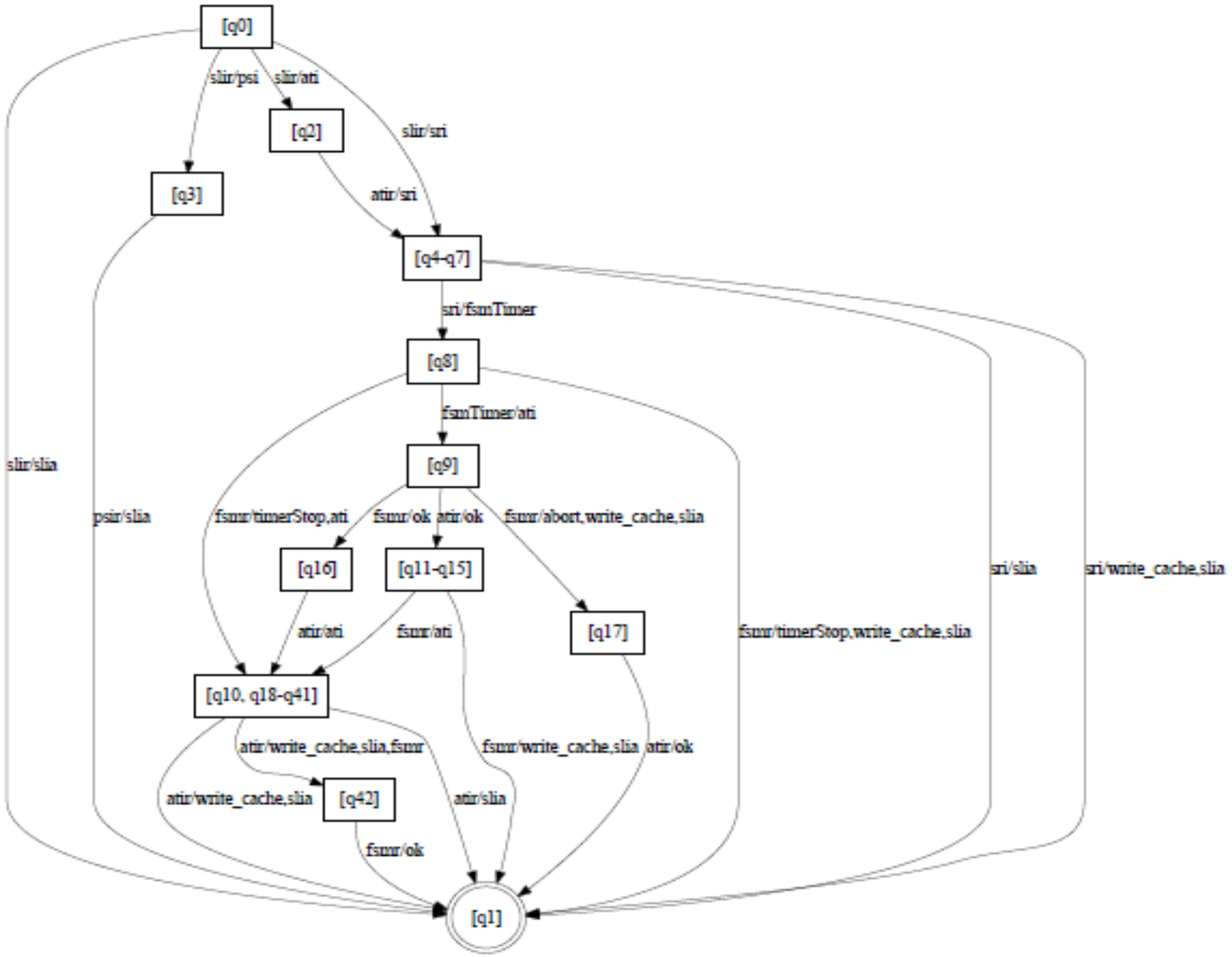


Task (II), Symbolic Model

- For second task, we have developed a tool

Task (II), Symbolic Model

- For second task, we have developed a tool



Evaluation

- Coverage
  - 12 control states out of 13
  - 26 out of 40 edges
- Similarity
  - correspondence of locations to control locations:
- Readability
  - the actual code is smaller, uses more complex structure

Evaluation

- Coverage
  - 12 control states out of 13
  - 26 out of 40 edges
- Similarity
  - correspondence of locations to control locations:
- Readability
  - the actual code is smaller, uses more complex structure

10	1->1
1	2->1
1	1->2
1	0->1

Conclusions

- Angluin's L* algorithm for inferring a model of communication protocols
  - LearnLib
    - 42 states, 1600 non-error edges
- Heuristic for folding the model
  - similar to actual protocol structure

Future Works

- To investigate alternative principles for construction locations
- Look for some advanced ways for generating action expressions
  - code transformation
  - reduce redundancies

Questions?