LBT for
Procedural and Reactive Systems

Part 2: Reactive Systems –
Basic Theory

Karl Meinke,

karlm@kth.se

School of Computer Science and Communication

KTH Stockholm

0. Overview of the Lecture

1. Learning Based Testing for Reactive Systems
2. DFA learning with Angluin's L* algorithm
3. Complexity of learning DFA

based on:

D. Angluin, Learning regular sets from queries and counterexamples, Information and Computation, 75:87-106, 1987.

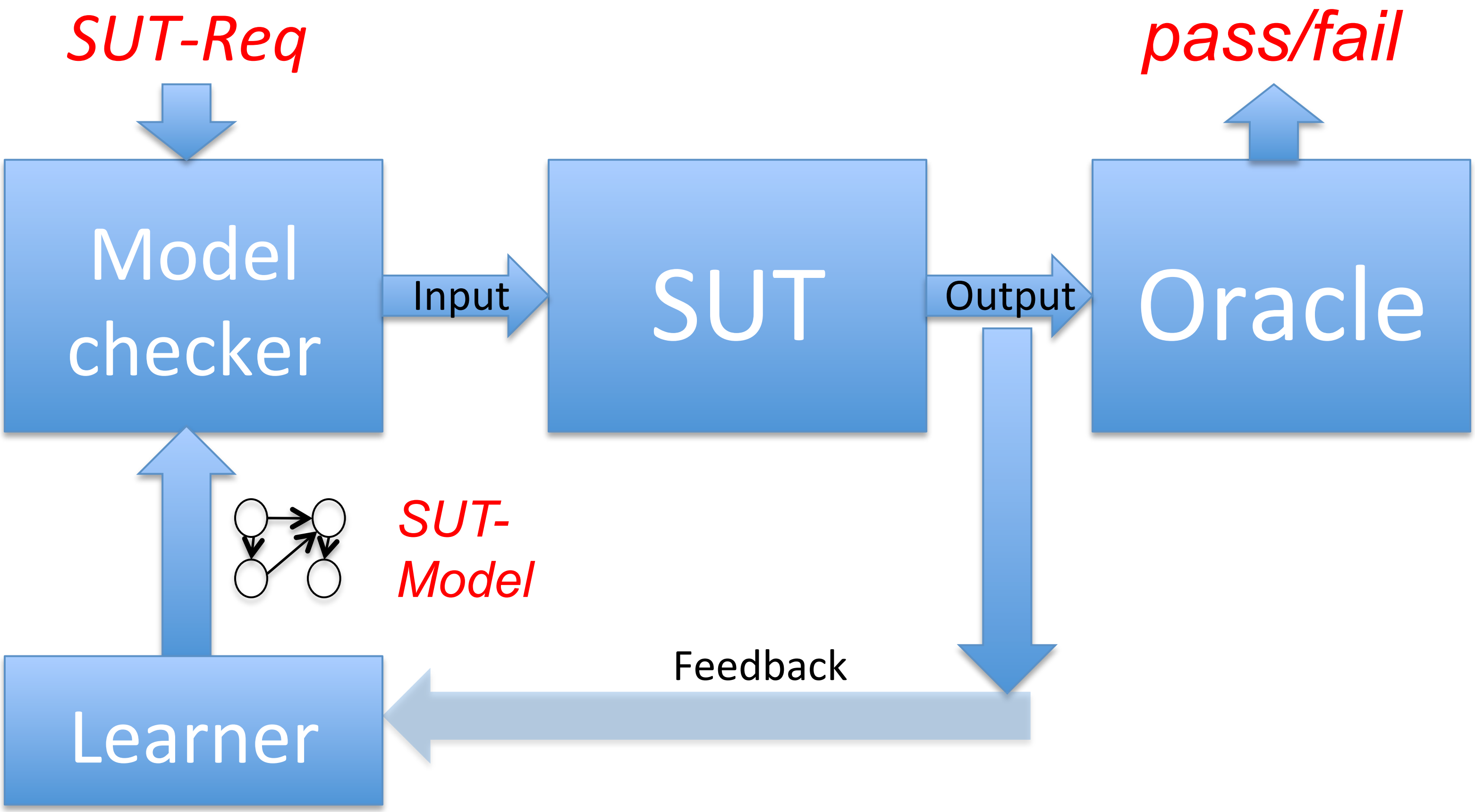
K. Meinke and M. Sindhu, *Incremental Learning-Based testing for Reactive Systems*, in: Proc. TAP 2011

K. Meinke and F. Niu, *Learning-Based Testing for Reactive Systems using Term Rewriting Technology* Proc. ICTSS 2011

M. Czerny, Learning-based software testing: Evaluation of Angluin's L* algorithm and adaptations in practice, Bachelor Thesis, KIT, 2014.

2. Learning-Based Testing (LBT)

Meinke & Sindhu 2011, Proc. TAP 2011



"*aka*. Model based testing without a model"

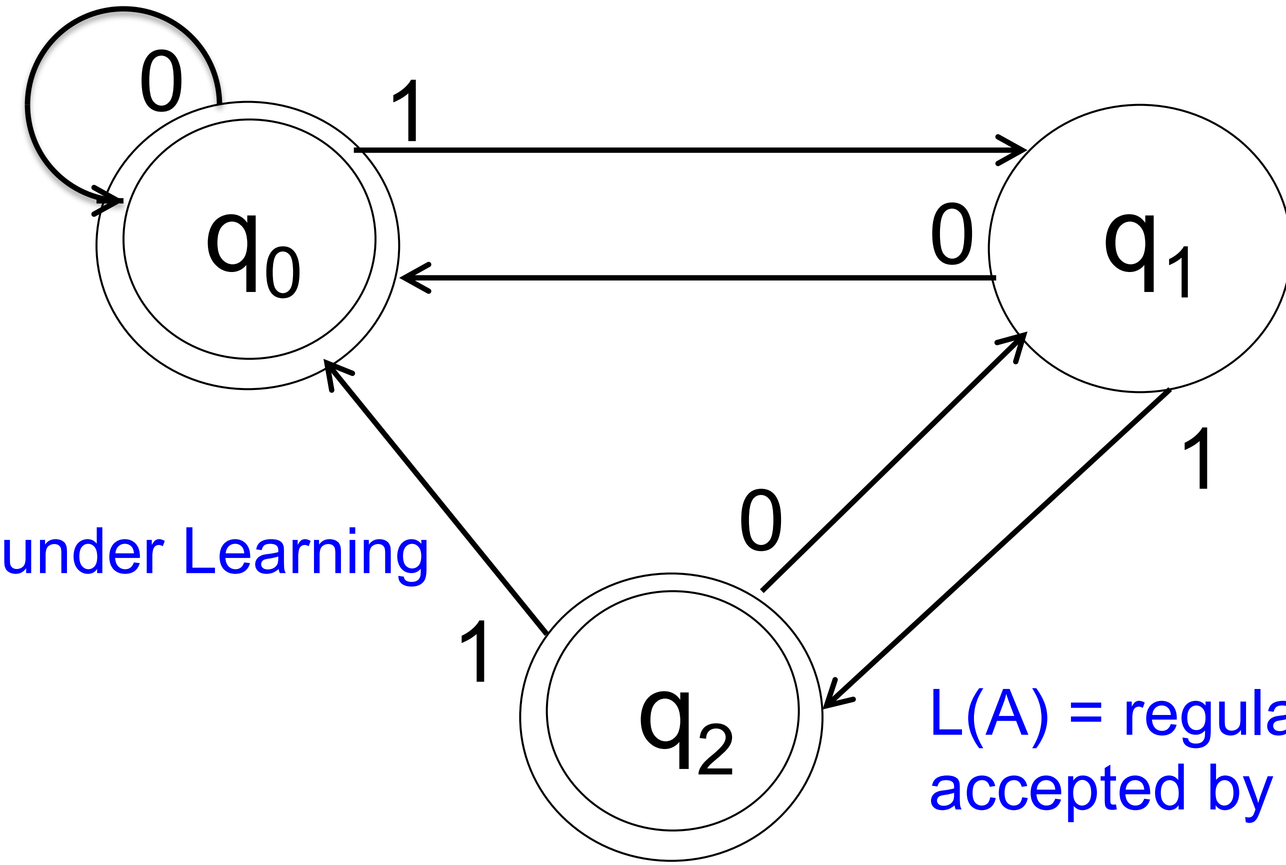
3. Framework for Study:
Reactive Systems

Generally *control-oriented* testing

1. Requirements language = propositional linear temporal logic (PLTL)
2. Model = FSM, Moore machine
3. Model checker = BDD/SAT-based checkers
4. Learning = regular inference algorithms

DFA (Moore) Representation

DFA A= (Q, Σ, q_0, F ⊆ Q, δ : Q X Σ→Q)



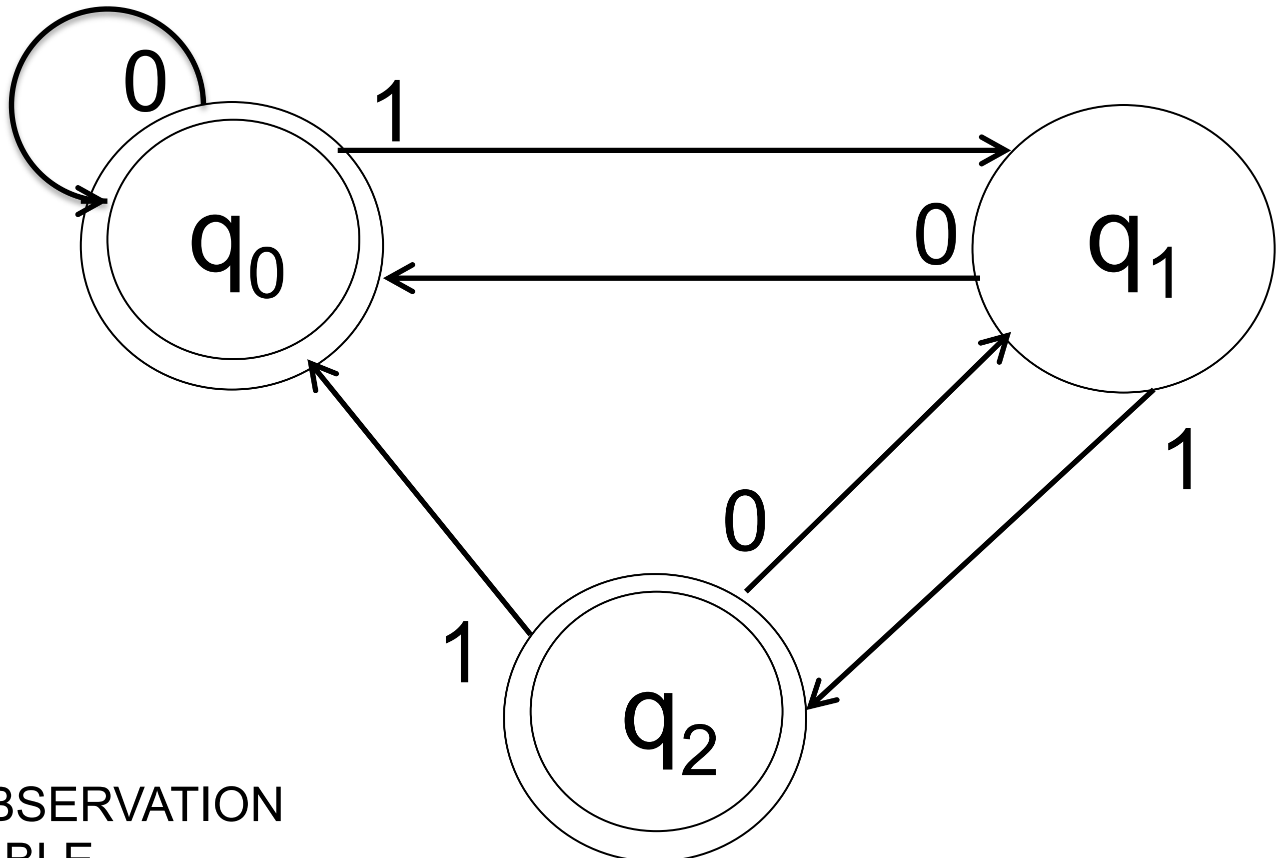
System under Learning
SUL

L(A) = regular language
accepted by A

DFA Learning with Observation Tables

- $P_A \subseteq \Sigma^*$ is a finite prefix-closed set of prefixes
- $S_A \subseteq \Sigma^*$ is a finite suffix-closed set of suffixes

- T_A: $P_A \cup (P_A . \Sigma) \times S_A \rightarrow \{1, 0, ?\}$ is the *observation table*
- Write $T_A(p)$ for a row $T_A(p,s_1), \ldots, T_A(p,s_n)$



OBSERVATION TABLE

	ε	0	1
ε	1	1	0
0	1	1	?

suffixes

prefixes

Basic Principle of DFA learning

- *Accessor strings* – prefixes p that reach each distinct state
- *Distinguishing strings* – suffixes s that separate distinct states

INFERENCE PRINCIPLE

If $T_A(p, s) \neq T_A(p', s)$ then p and p' cannot reach the same state in A;

s is a *distinguishing string* for p, p'

Closed & Consistent Tables

T_A is *closed* iff, for each $p \in P_A.\Sigma$ there exists $p' \in P_A$ st.

$$T_A(p) = T_A(p')$$

T_A is *consistent* iff, for each $p, p' \in P_A$ if

$$T_A(p) = T_A(p')$$

then for all $a \in \Sigma$,

$$T_A(p.a) = T_A(p'.a)$$

Algebraic Properties

- Being closed is an *algebraic closure condition*
- Being consistent is an *algebraic congruence condition*
- The automaton construction is a *quotient algebra construction*

- Learning DFA as *string rewriting systems*
- K. Meinke, *CGE: a Sequential Learning Algorithm for Mealy Automata*, in Proc. ICGI 2010
- K. Meinke and F. Niu, *Learning-Based Testing for Reactive Systems using Term Rewriting Technology in* Proc. ICTSS 2011

Equivalence Oracles

- Termination requires an *equivalence oracle*
- If A and SUL are *behaviourally equivalent*,

  i.e. L(A) = L(SUL) then

  `equivOracle(A, SUL) = true`
- Otherwise

  `equivOracle(A, SUL) = v` $\in \Sigma^*$

  where `A(v)` $\neq$ `SUL(v)`
- LBT uses *stochastic equivalence checking*

Complexity Observations

- Stochastic equivalence checking is more powerful than random test cases – Why?

Theorem: *active learning* of DFA is much more efficient than *passive learning :*
polynomial time [Angluin 1987] vs. NP-hard [Gold 1978].

- Using stochastic equivalence checking we can PAC learn DFA in polynomial time (c.f. Kearns and Vazirani 1994).

```
DFA function LStar(DFA: SUL)
```

$P_A \subseteq \Sigma^*$

$P_A . \Sigma \subseteq \Sigma^*$

$S_A \subseteq \Sigma^*$

$T_A: P_A \cup (P_A . \Sigma) X S_A \rightarrow \{accept, reject, ?\}$ // table

```
begin
   A = getInitialHypothesis()

   while(equivOracle(A, SUL)!= true)) do
   A =getNextHypothesis(equivOracle(A, SUL))

   return A
end
```

```
DFA function
getNextHypothesis(counterExample ∈ Σ* )

begin
P_A = P_A ∪ PrefixClosure{counterExample}
P_A.Σ = P_A × Σ - P_A
S_A = S_A ∪ SuffixClosure{counterExample}
// fill in any new table entries here …

while (P_A, S_A, T_A) is not closed or consistent do
    if !consistent(P_A,S_A,T_A) makeConsistent()
    if !closed(P_A,S_A,T_A) makeClosed()
end
```

```
DFA function getInitialHypothesis()
begin
    P_A = ∅    // emptyset
    P_A.Σ = ∅
    S_A = ∅
    return getNextHypothesis(ε)
end
```

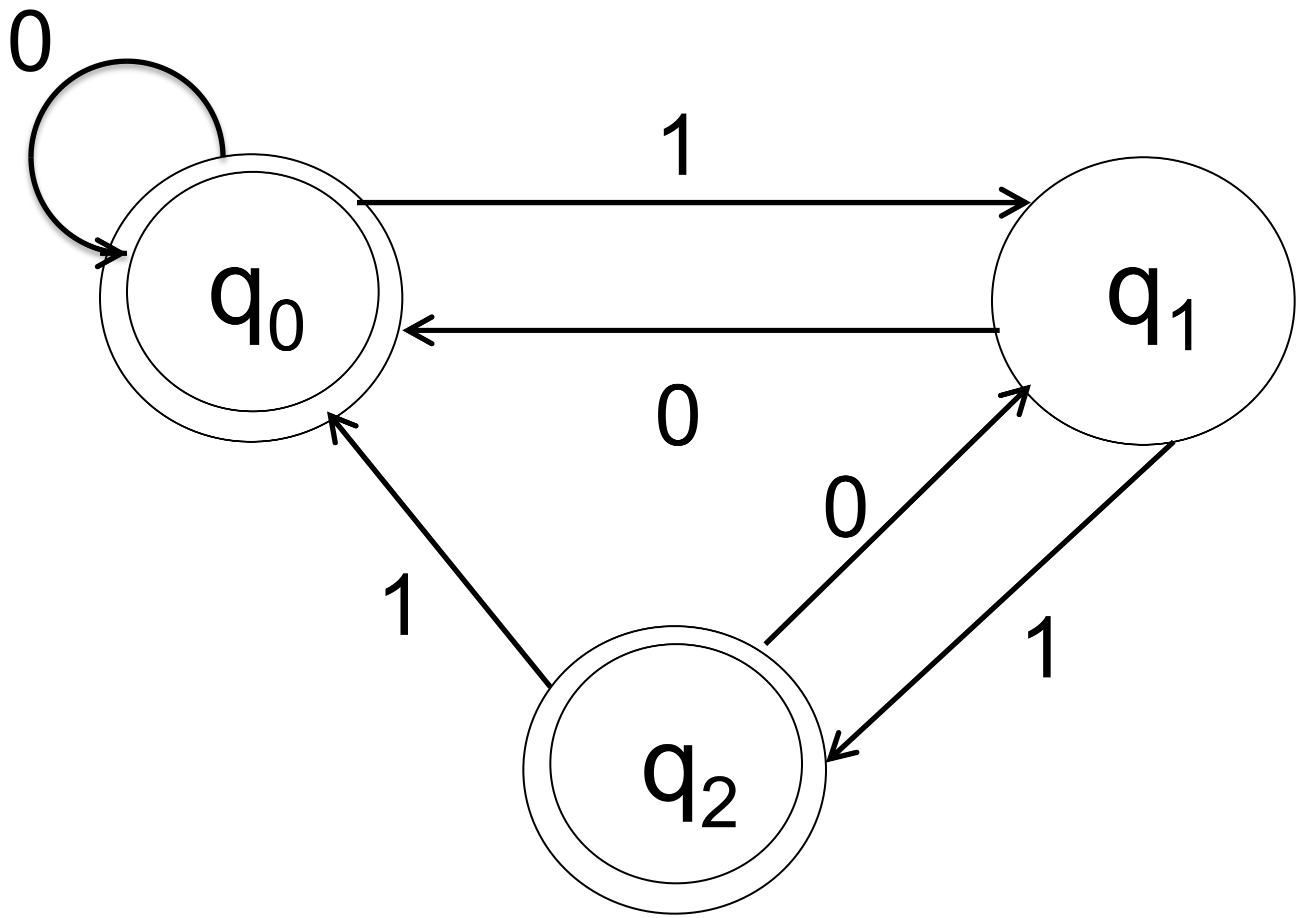
```
DFA function
makeConsistent()
begin
   find p, p′ ∈ P_A, a∈ Σ, s∈S_A  st.
   T_A (p) = T_A (p′) and
   T_A(p.a,s) != T_A(p′.a,s)
   add a.s to S_A // suffix extension
   extend T_A to P_A∪ (P_A.Σ) X S_A using
   active membership queries
end
```

```
DFA function
makeClosed()
begin
   find p ∈ P_A, a ∈ Σ st.
   T_A (p.a) != T_A (p') for all p'∈ P_A
   P_A = P_A ∪ {p.a} // prefix extension
   P_A.Σ = P_A.Σ ∪ {p}×Σ

   extend T_A to P_A∪ (P_A.Σ) X S_A using
   membership queries
end
```

```
DFA function DFASynthesis()
begin
Q = {u : u ∈ P_A, ∀v<u, T_A(u)≠ T_A(v) }
q_0 = ε
F = {u : u ∈ Q, T_A(u,ε)= 1}
foreach u ∈ Q do
   foreach a ∈ Σ  do
      δ(u, a) = v ∈ Q st. T_A(u.a)=T_A(v)

return A=(Q, Σ, q_0, F, δ)
end
```



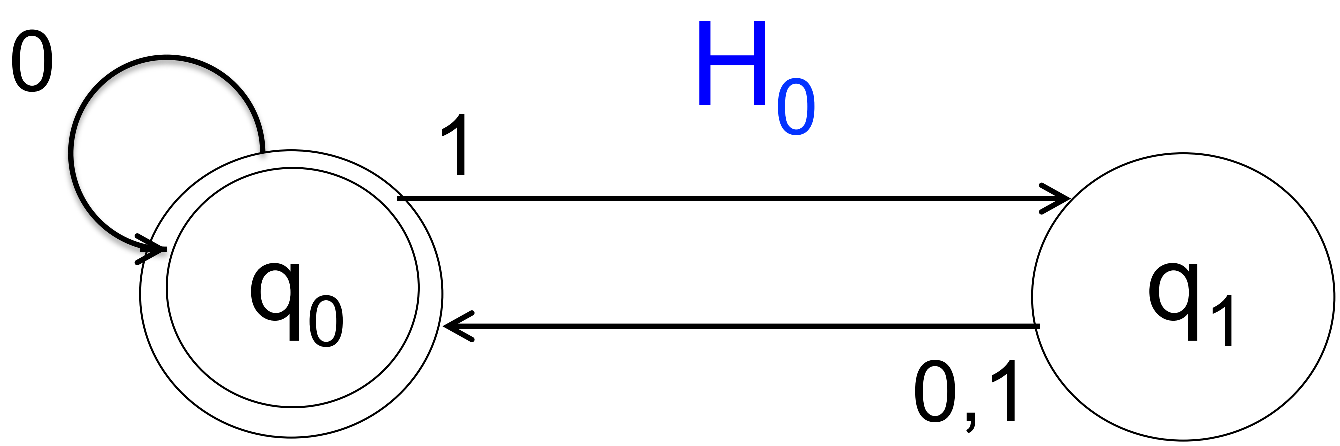
Let's L* learn this DFA

	ε
ε	1
0	1
1	0

Closed: no
Consistent: yes

	ε
ε	1
1	0
0	1
10	1
11	1

Closed: yes
Consistent: yes



`equivOracle(H_0, SUL)` = 110 = counterExample

	ε
ε	1
1	0
11	1
110	0
0	1
10	1
1100	1
1101	1
111	1

← counterExample (row 110)

Not consistent because

$T_A(\varepsilon) = T_A(11) = 1$

but

$T_A(1) \neq T_A(111)$ since $0 \neq 1$

Also
$T_A(0) \neq T_A(110)$ since $0 \neq 1$

Closed: yes
Consistent: no

equivOracle(H_0, SUL) = 110 = counterExample

	ε
ε	1
1	0
11	1
110	0
0	1
10	1
1100	1
1101	1
111	1

Closed: yes
Consistent: no

	ε	0	1
ε	1	1	0
1	0	1	1
11	1	0	1
110	0	1	1
0	1	1	0
10	1	1	0
1100	1	1	0
1101	1	0	1
111	1	1	0

Closed: yes
Consistent: yes

H_1 = FINISHED!

Theoretical Complexity

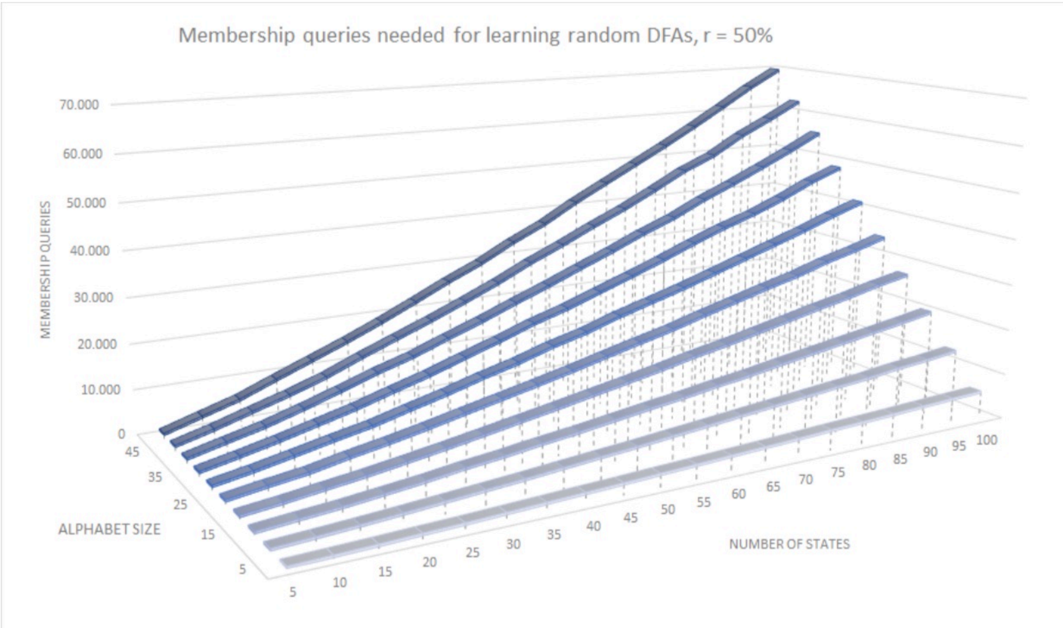
Membership queries = $O(m|\Sigma||Q|^2)$

where m is the maximum length of any counterexample (Angluin 87).

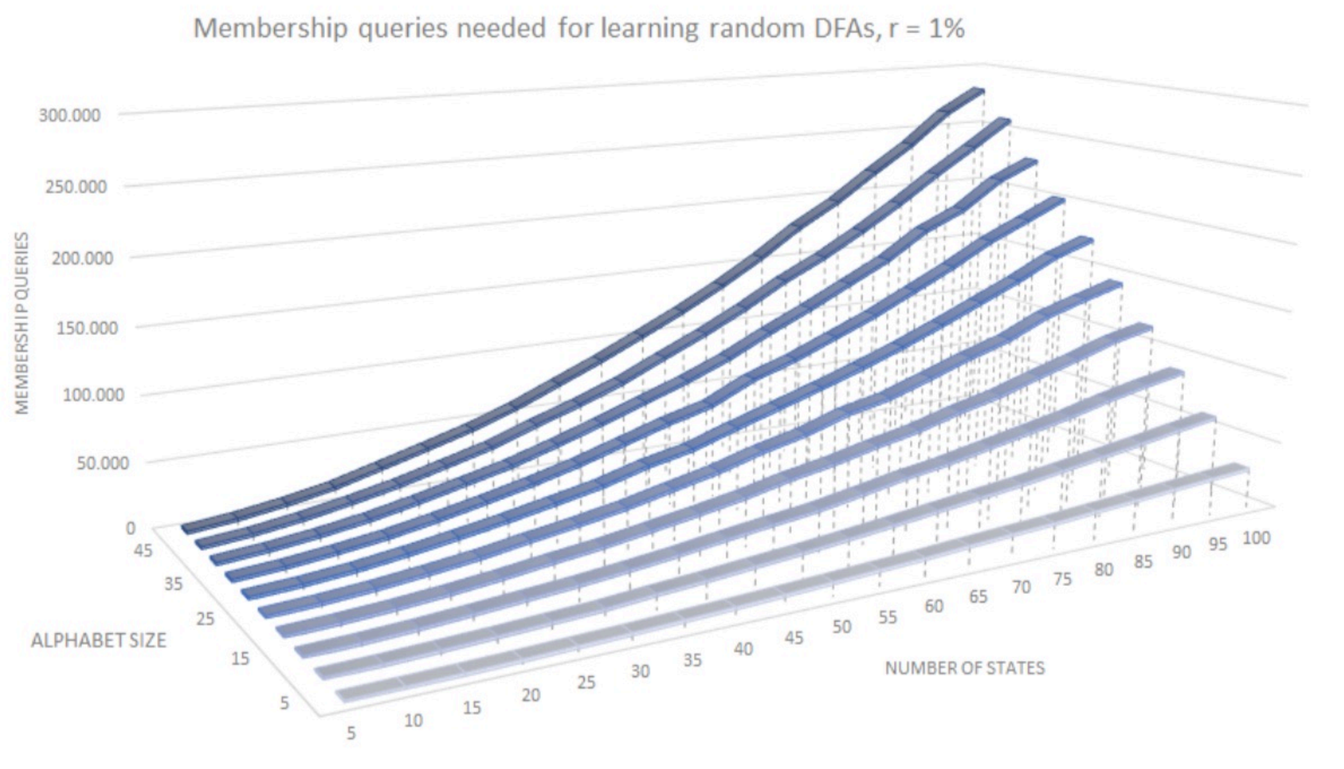
Assume oracle returns shortest counterexample

then $Q \geq m$, so

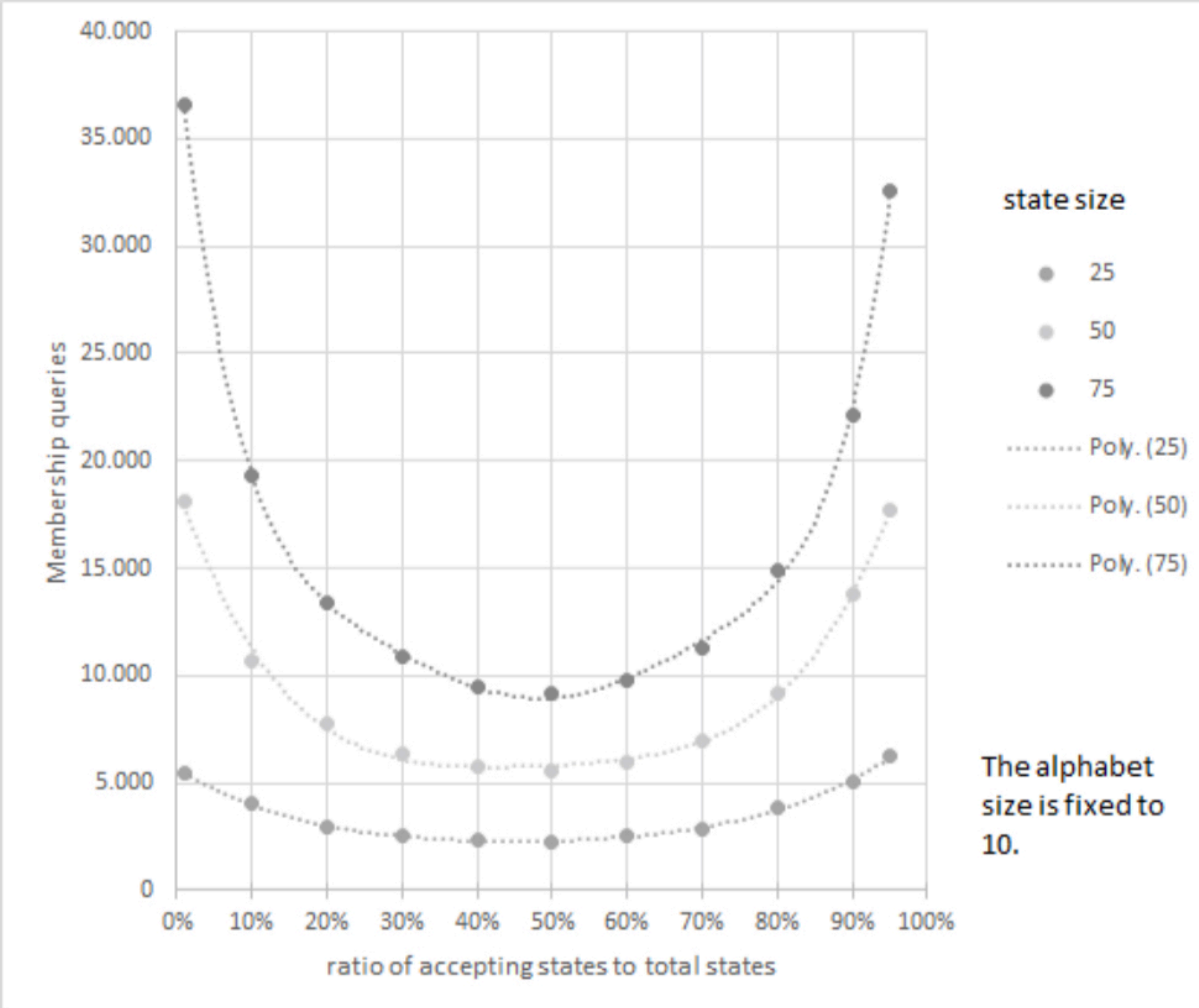
Membership queries = $O(|\Sigma||Q|^3)$



Essentially linear



Essentially quadratic



Membership queries
40.000
35.000
30.000
25.000
20.000
15.000
10.000
5.000
0
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
ratio of accepting states to total states
state size
25
50
75
Poly. (25)
Poly. (50)
Poly. (75)
The alphabet size is fixed to 10.

5. Conclusions

- L* is pedagogically easy to learn
- some of its principles are universal
- looks promising on paper
- emphasis on "complete learning"
- LBT needs "incremental learning"

Open Questions

- How complex are real SUTs?
- Can try to benchmark other learning algorithms in same framework