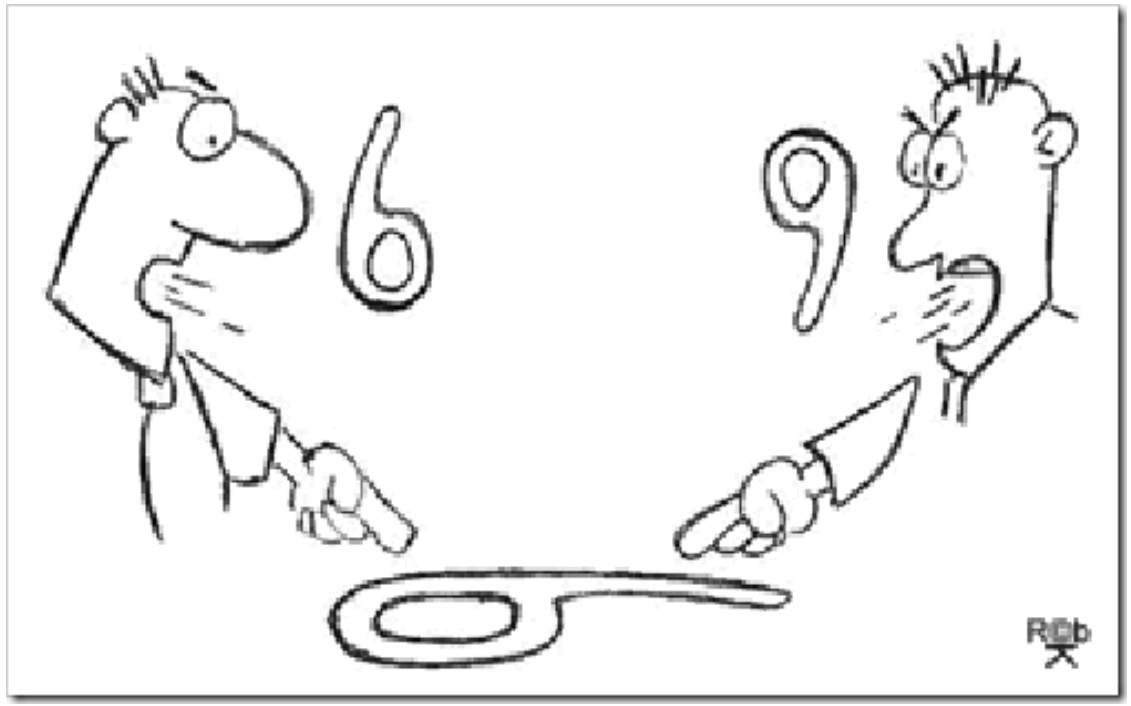
Model Learning and Model Checking of SSH Implementations

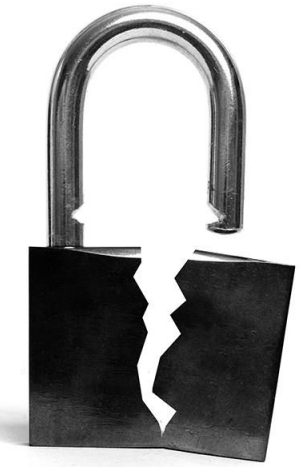
Radboud University

Paul Fiterau, Toon Lenaerts, Erik Poll,
Joeri de Ruiter, Frits Vaandrager,
Patrick Verleg

Introduction

- protocols: SSH, TLS, SMTP, FTP, TCP, UDP…
- many implementations per protocol
  - ➢implementations MUST/SHOULD/MAY ***adhere*** to the specifications…

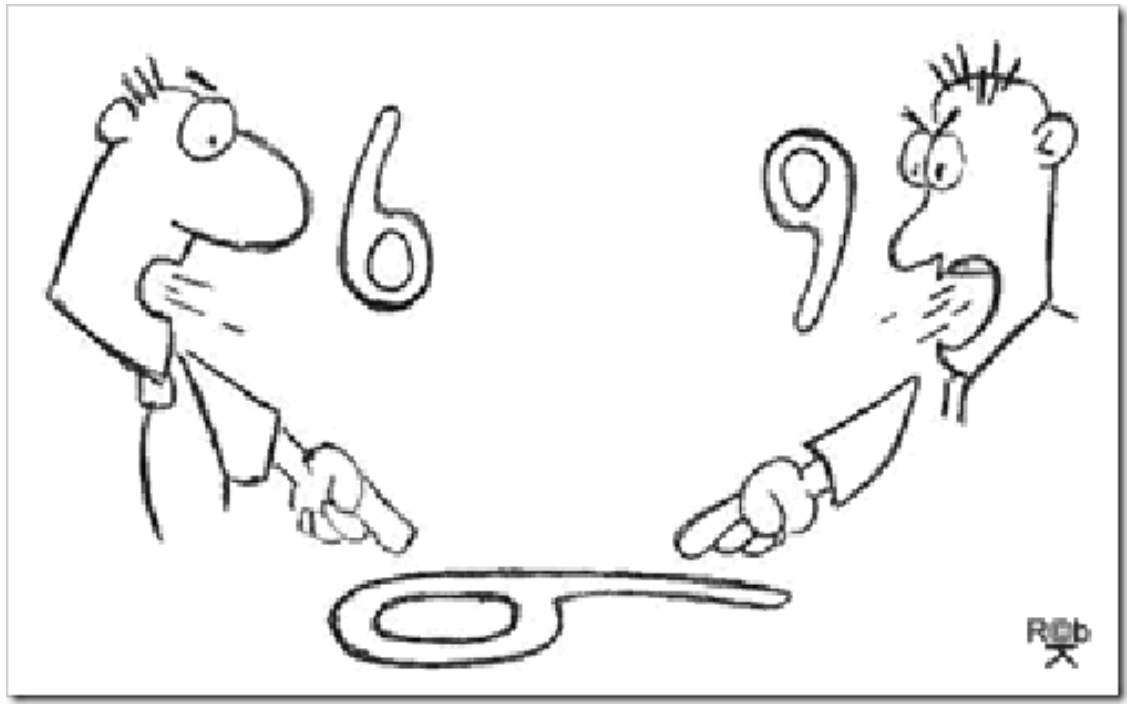


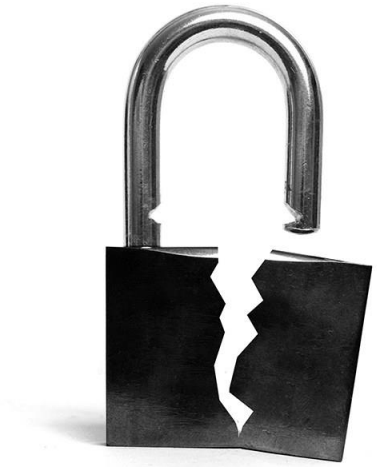


Introduction

- protocols: SSH, TLS, SMTP, FTP, TCP, UDP…
- many implementations per protocol
  - ➢implementations MUST/SHOULD/MAY ***adhere*** to the specifications…

conformance testing





Motivation

Model Learning

→*automatically* infers models for concrete implementations

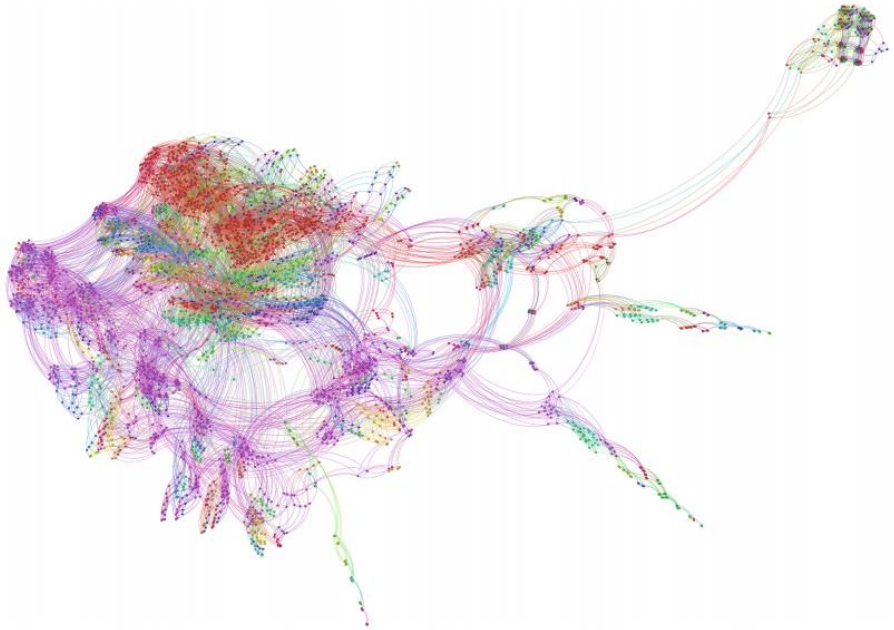
→checking conformance of models may be difficult

Motivation

Model Learning

→*automatically* infers models for concrete implementations

→checking conformance of models may be difficult



Motivation

Model Checking

→*automatically* checks conformance of models to specifications

→*requires* models and formalized specifications

Model Learning

→*automatically* infers models for concrete implementations

→checking conformance of models may be difficult

Motivation

Model Learning + Model Checking

→*automatically* infers models for concrete implementations

→*automatically* checks conformance of models to specifications

→*requires* formalized specifications

Model Checking

→*automatically* checks conformance of models to specifications

→requires models and formalized specifications

Model Learning

→*automatically* infers models for concrete implementations

→checking conformance of models may be difficult

What was done

Model Learning + Model Checking

→*automatically* infers models for concrete implementations

→*automatically* checks conformance of models to specifications

→*requires* formalized specifications

Application of ML+MC on SSH (a real world protocol):

1. use Model Learning to infer models of 3 SSH server implementations
2. formalize specifications from the SSH RFC standards
3. use Model Checking to verify models against these specification

What was done

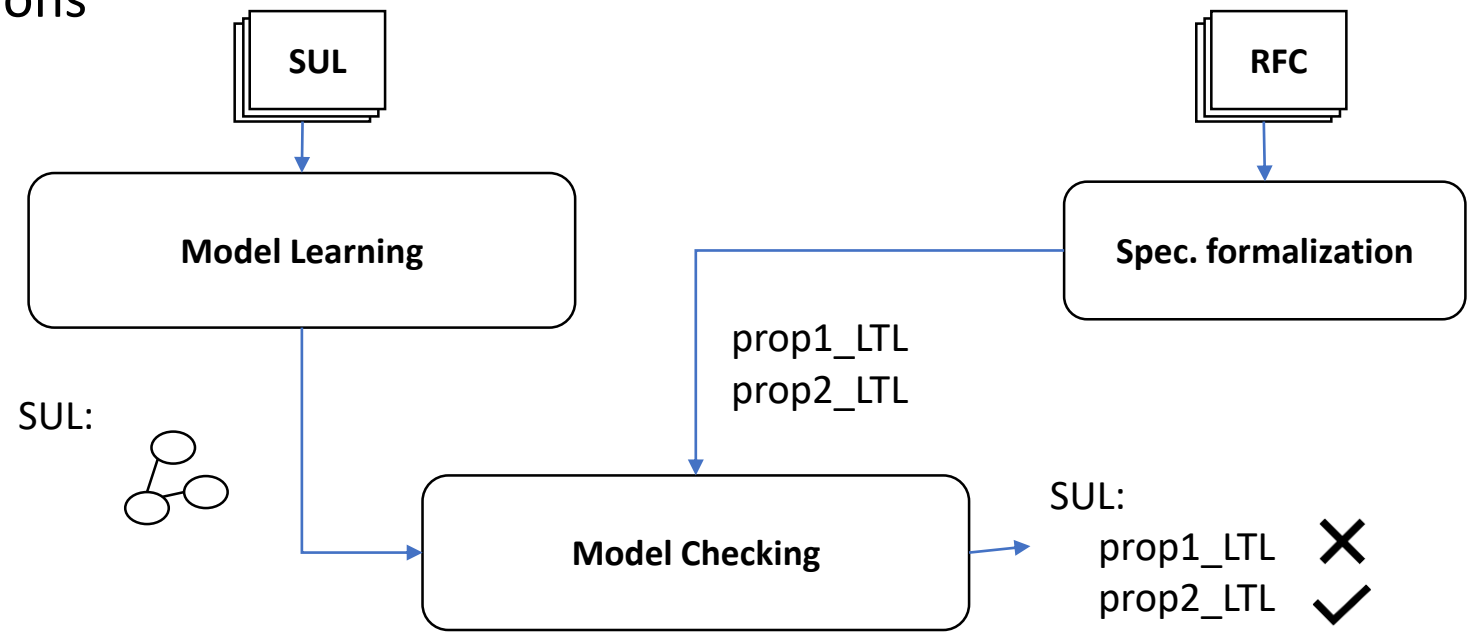
Model Learning + Model Checking

→*automatically* infers models for concrete implementations

→*automatically* checks conformance of models to specifications

→*requires* formalized specifications

Schematic Overview



What was done

Model Learning + Model Checking

→*automatically* infers models for concrete implementations

→*automatically* checks conformance of models to specifications

→*requires* formalized specifications

enough for a publication?

What was done

Model Learning + Model Checking

→*automatically* infers models for concrete implementations

→*automatically* checks conformance of models to specifications

→*requires* formalized specifications

Model Learning:

→*requires* mapper component

→*requires* thorough testing

Model Checking:

→*requires* model transformation

→*requires* counterexample validation

What was done

Model Learning + Model Checking

→*automatically* infers models for concrete implementations

→*automatically* checks conformance of models to specifications

→*requires* mapper component

→*requires* formalized specifications

→*requires* thorough testing

→*requires* model transformation

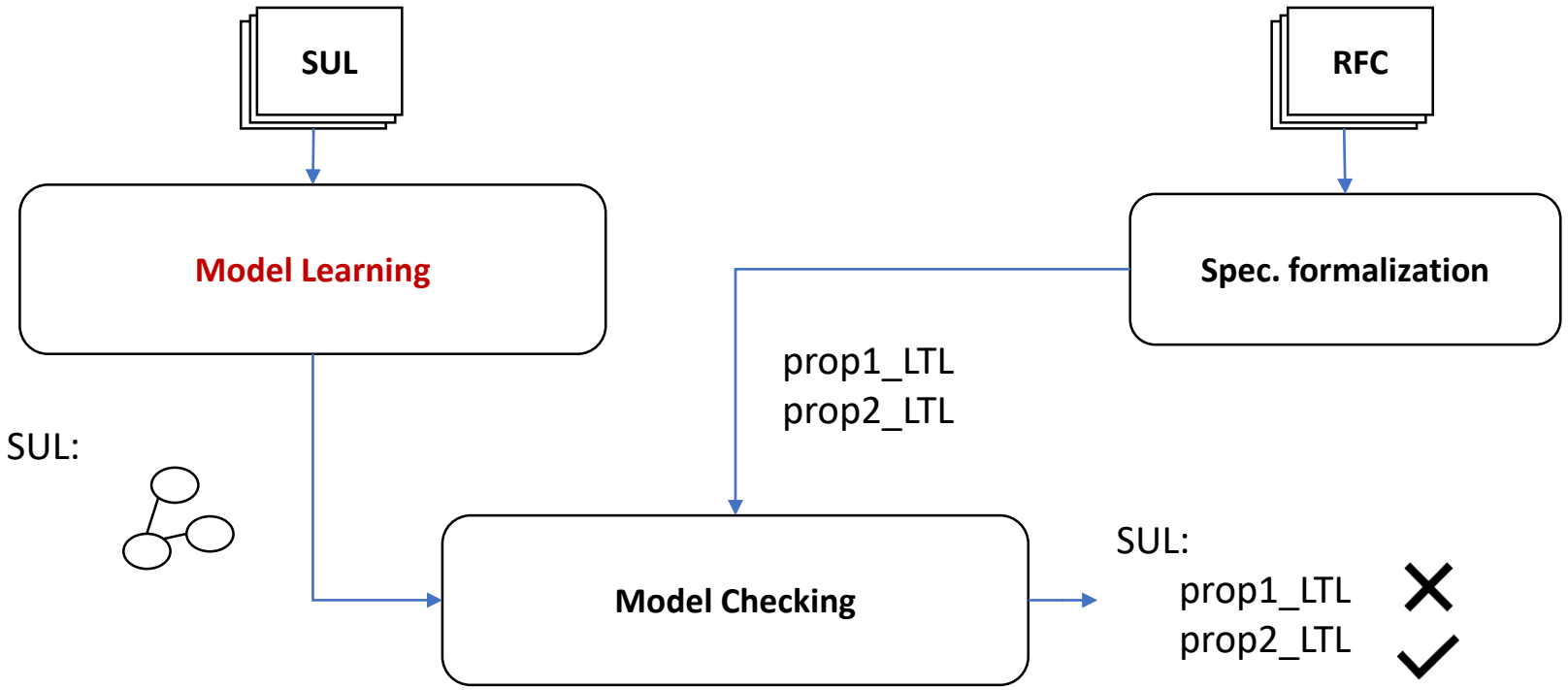
→*requires* counterexample validation

Patrick's M. Thesis

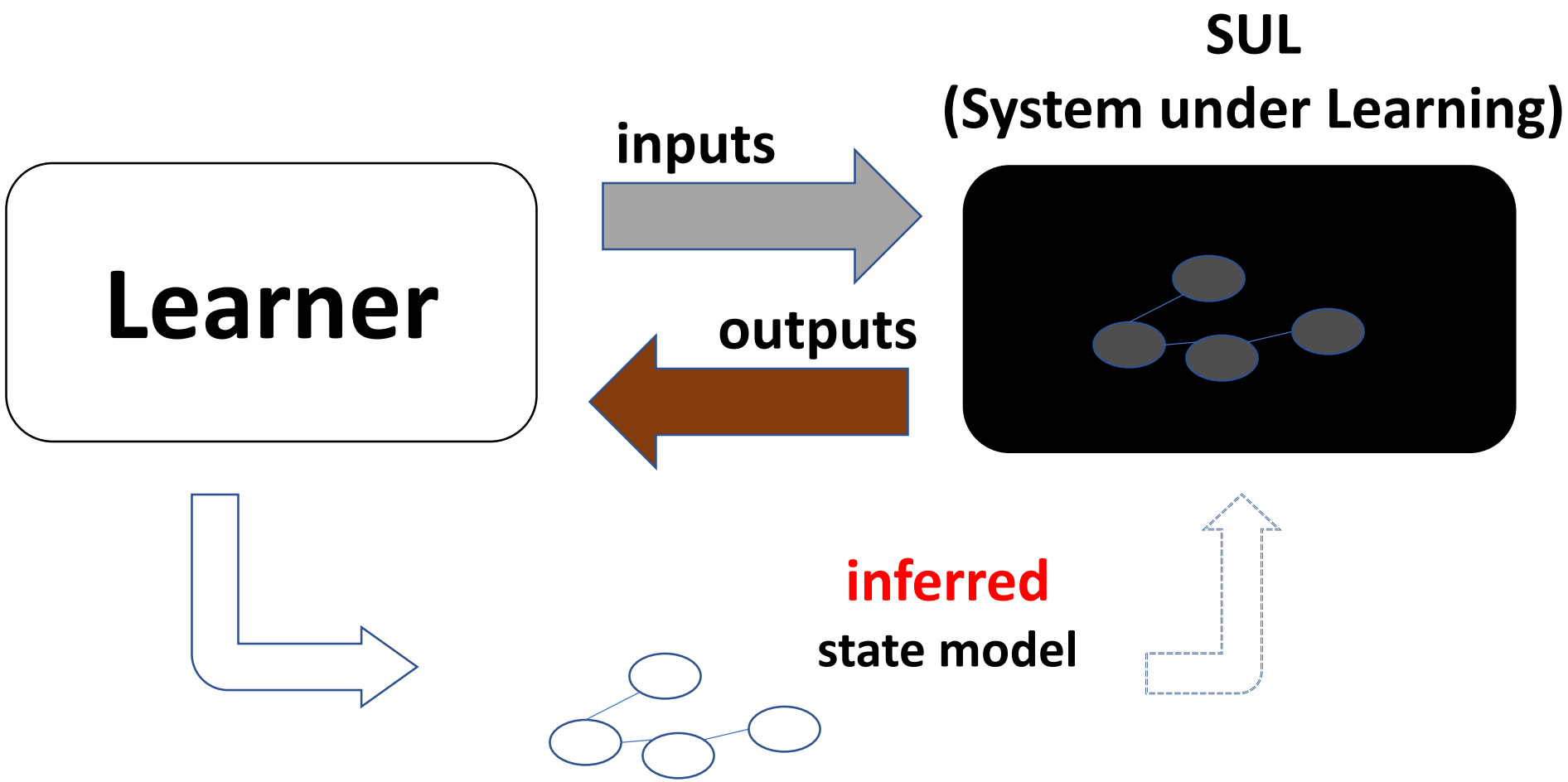
Toon's B. Thesis

Publication

What was done

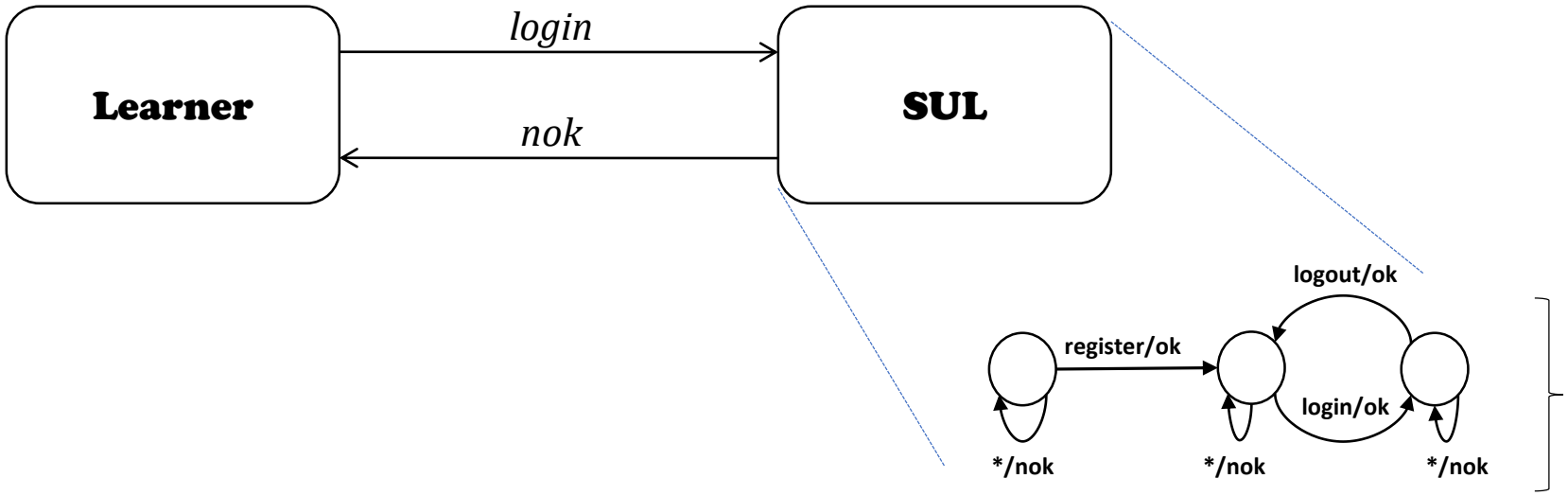


Model Learning



Model Learning

Learner Queries:
register/ok



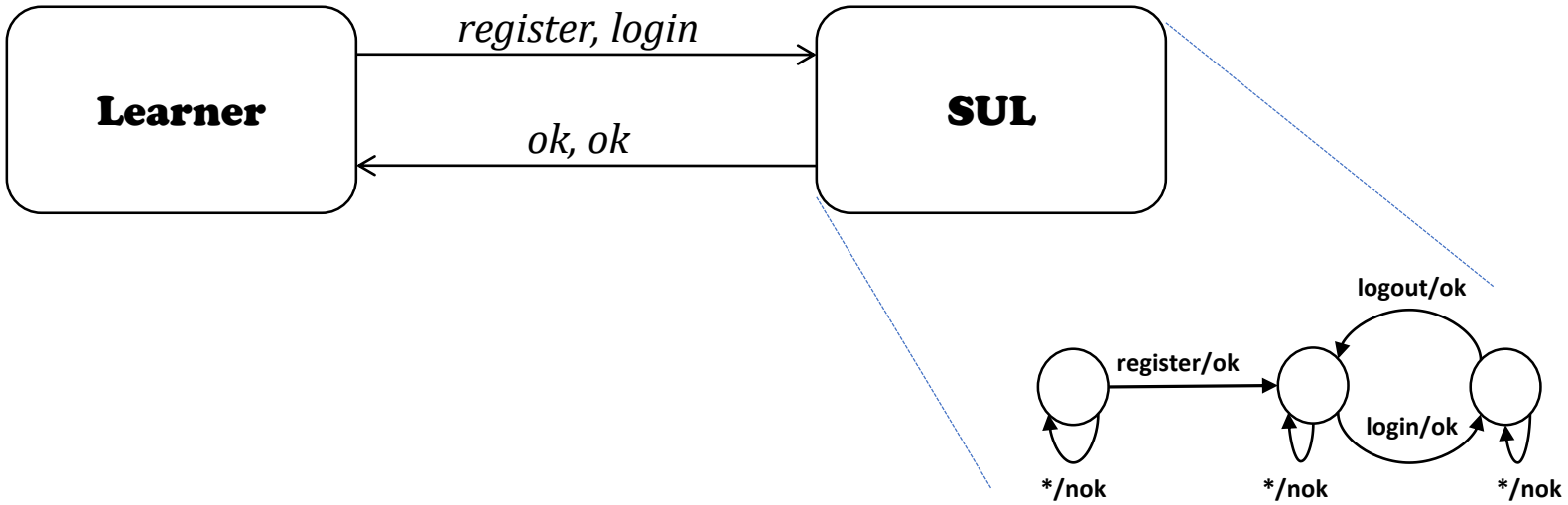
Mealy Machine

Input Alphabet:
[register, login, logout]
Output Alphabet: [ok, nok]

Model Learning

Learner Queries:

register/ok
login/nok
logout/nok
register/ok register/nok



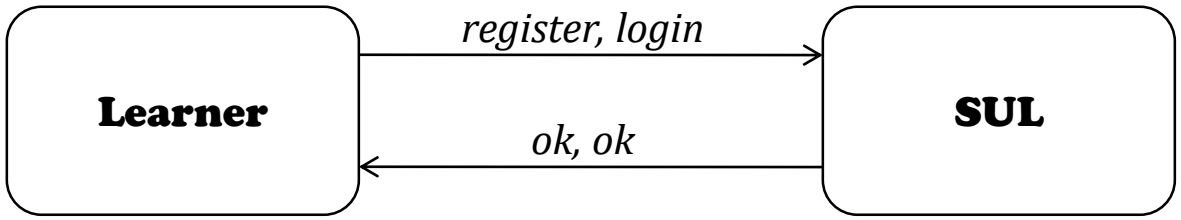
Input Alphabet:

[register, login, logout]

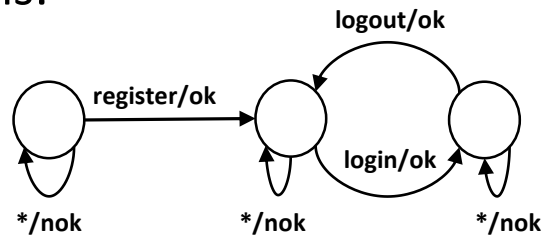
Model Learning

Learner Queries:

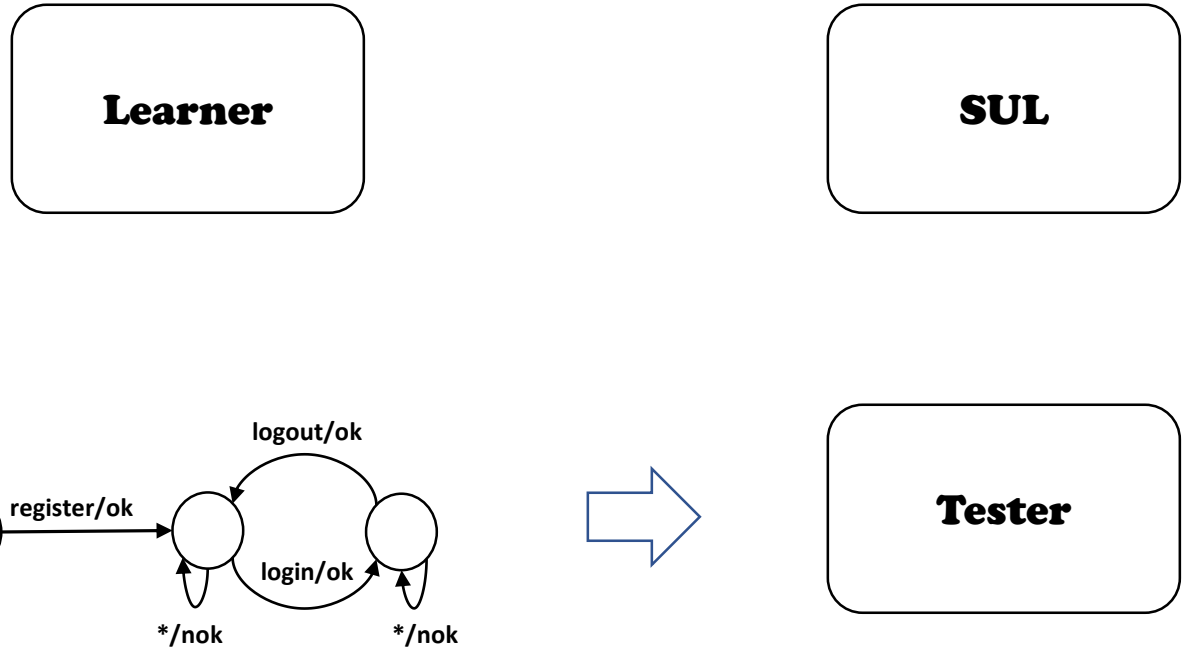
- register/ok
- login/nok
- logout/nok
- register register/ ok nok



Hypothesis:

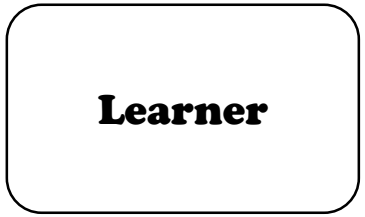


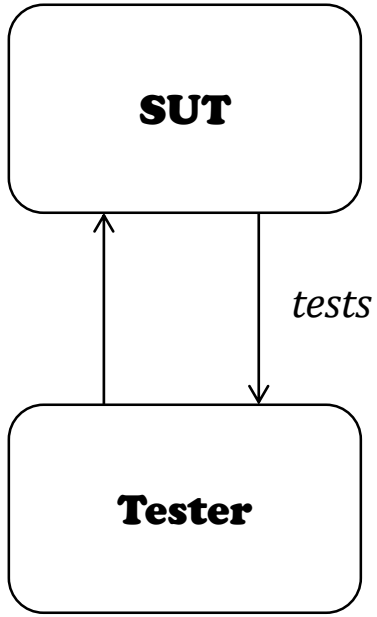
Model Learning



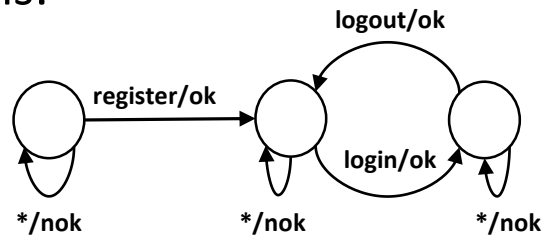
Model Learning

Test Queries:
register register login/ok nok nok

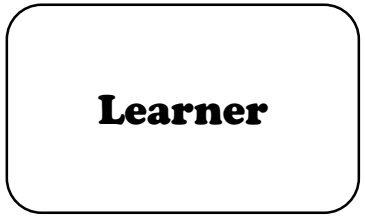


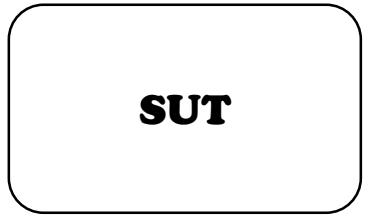


Hypothesis:

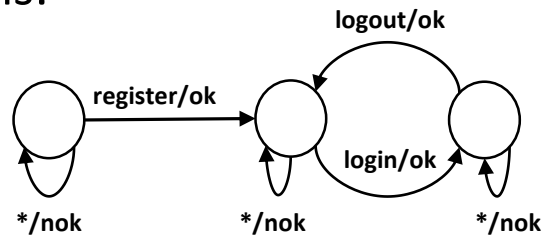


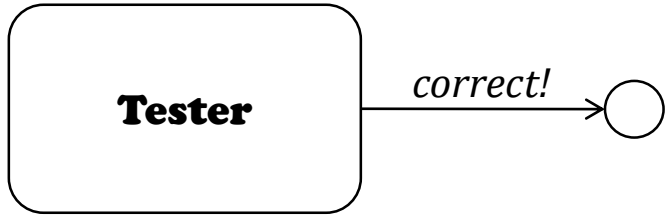
Model Learning



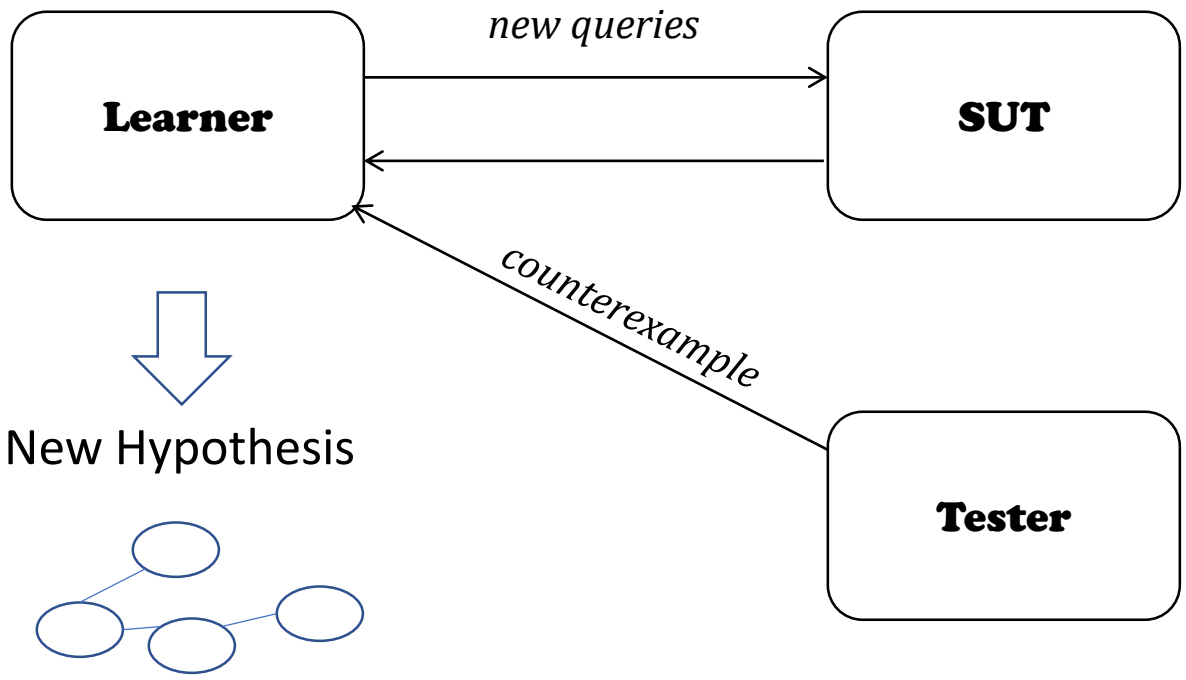


Hypothesis:

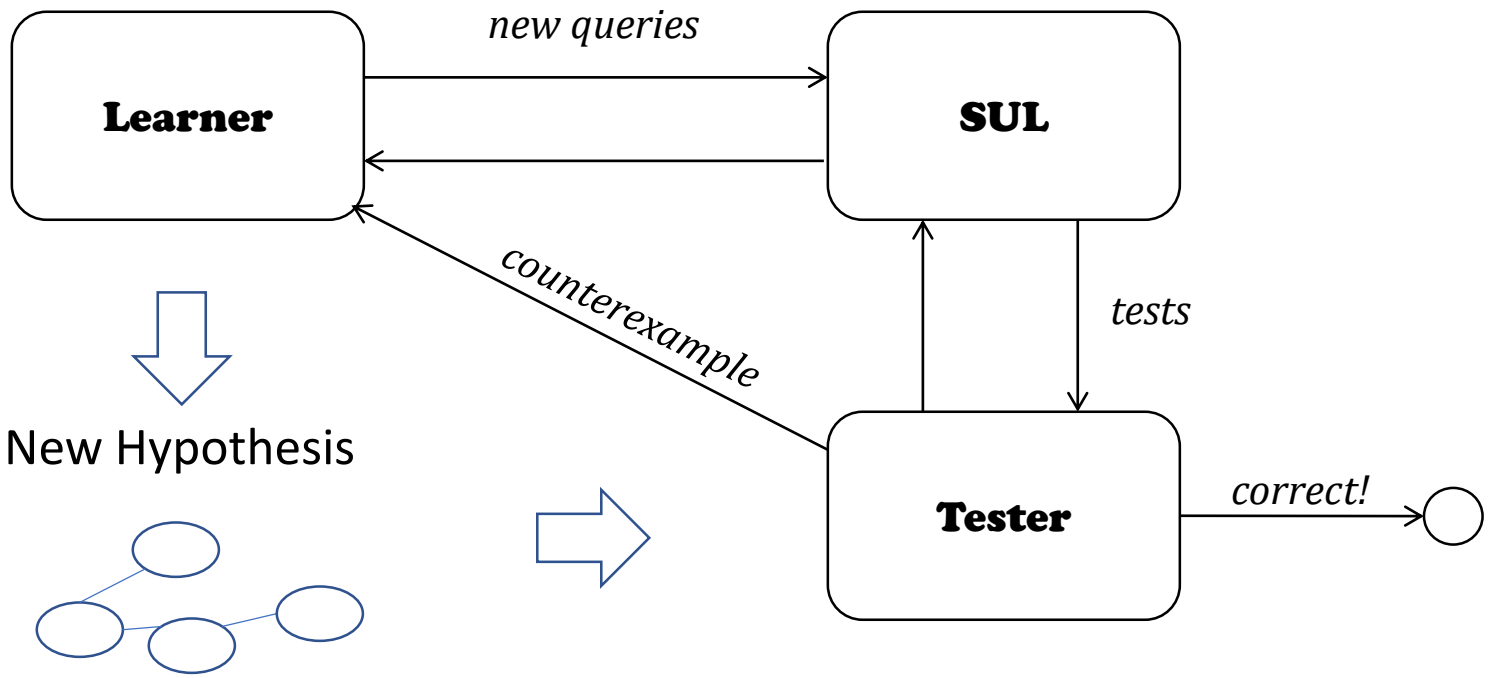




Model Learning

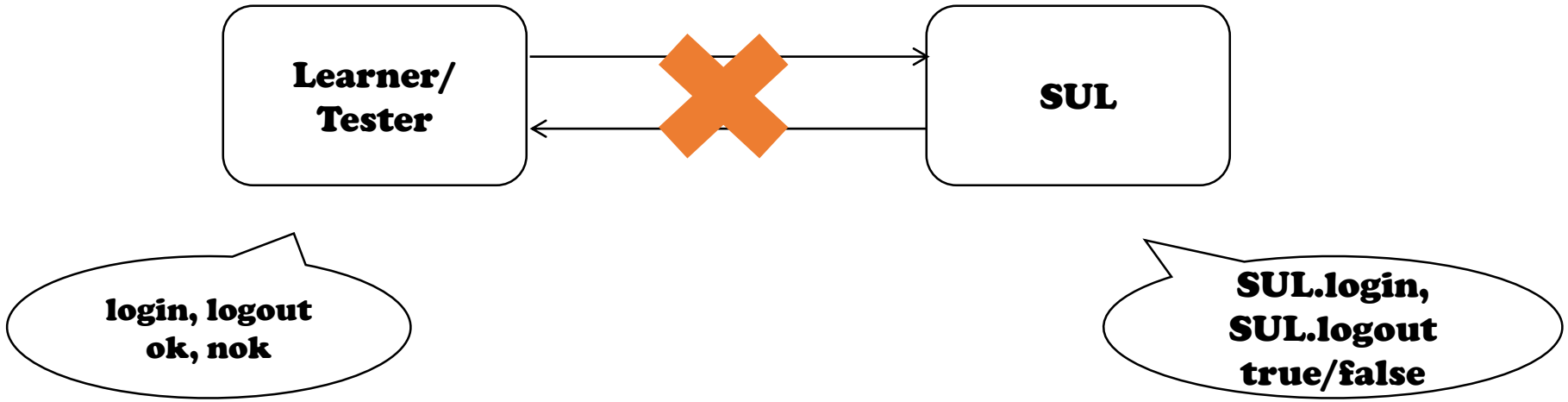


Model Learning



Model Learning

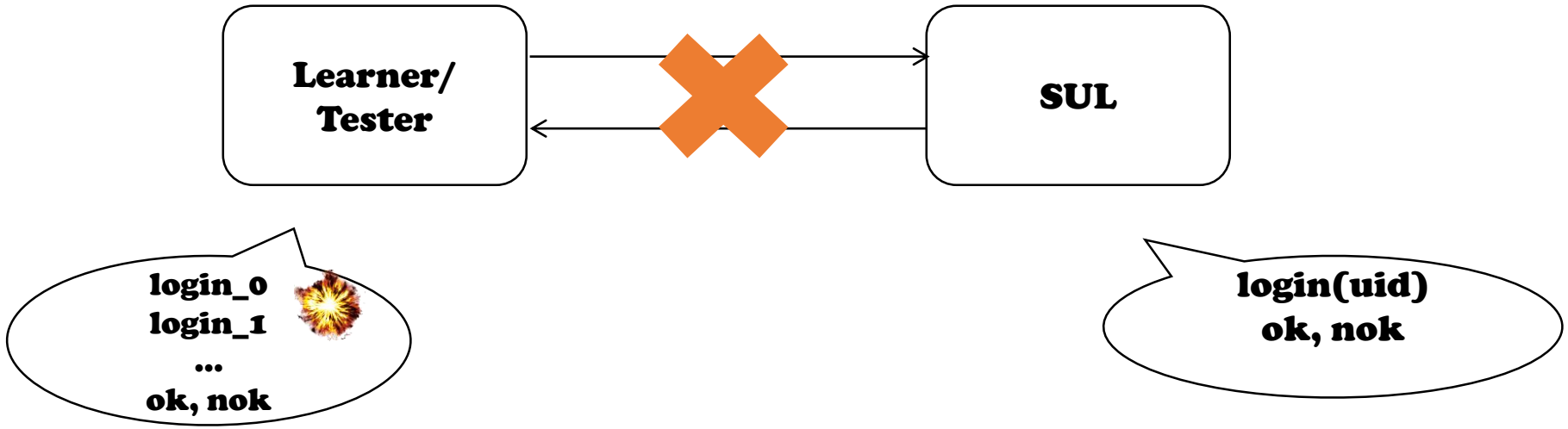
Model Learning



abstract i/o
(strings)

concrete i/o
method calls,
returned obj.
packets

Model Learning

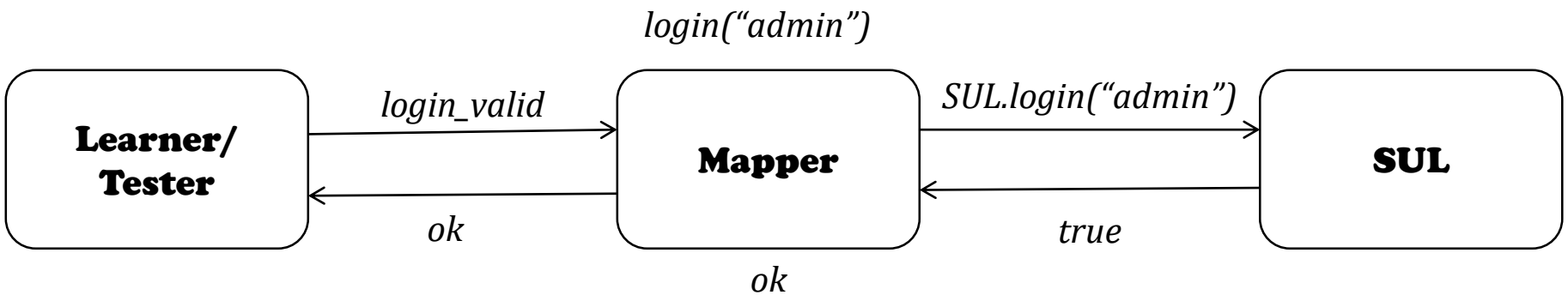


small abstract
i/o alphabet

parameterized
i/o alphabet

Model Learning

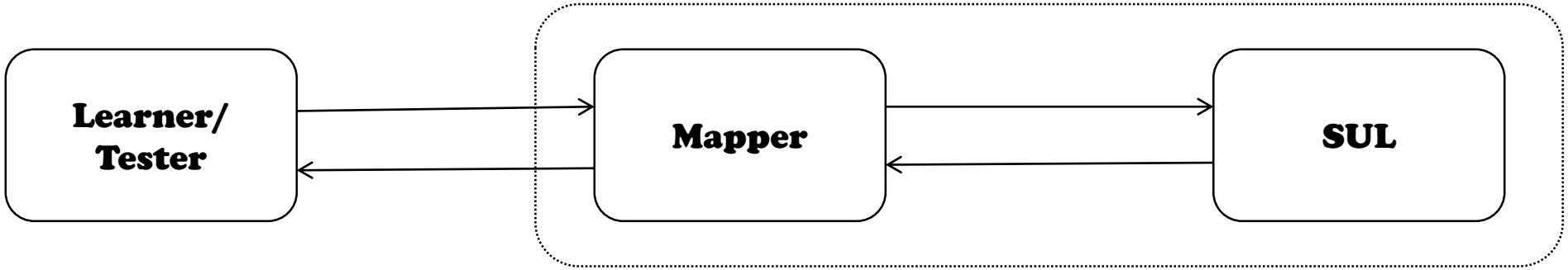
abstract i/o param i/o concrete i/o



Mapper

1. translates:
   - between abstract and param. i/o
   - between param. i/o and concrete i/o

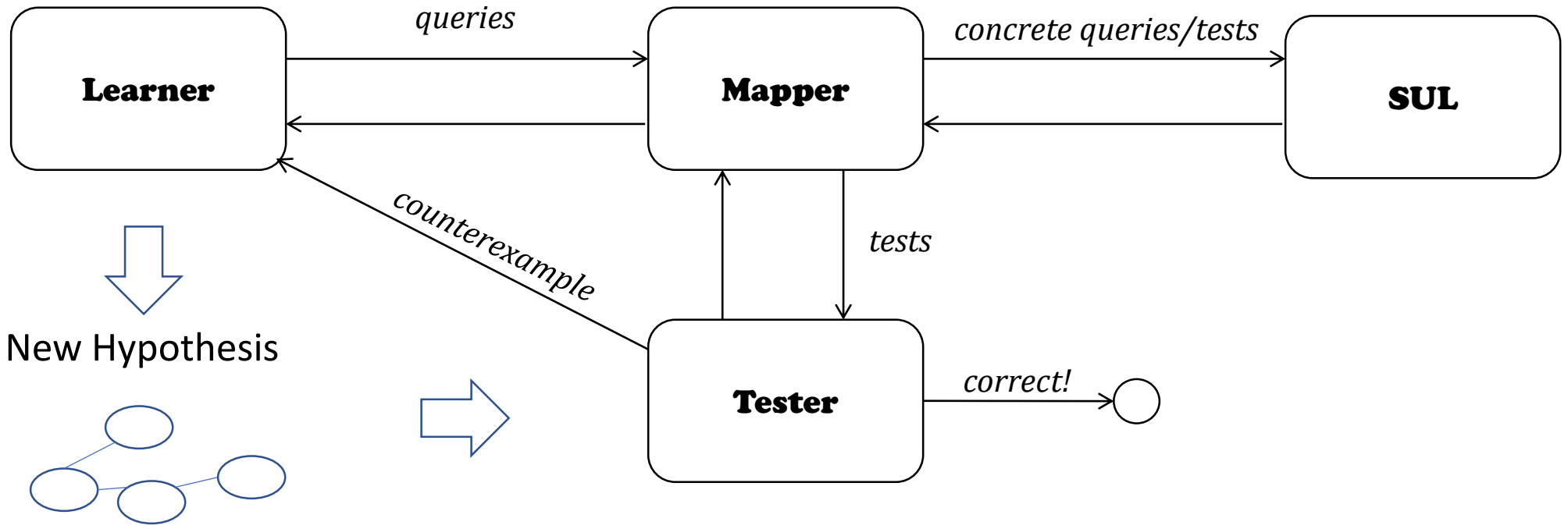
Model Learning

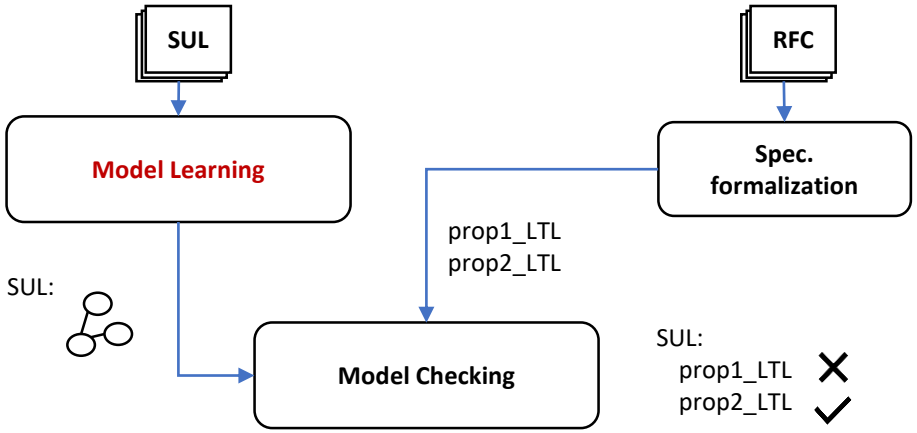


Mapper

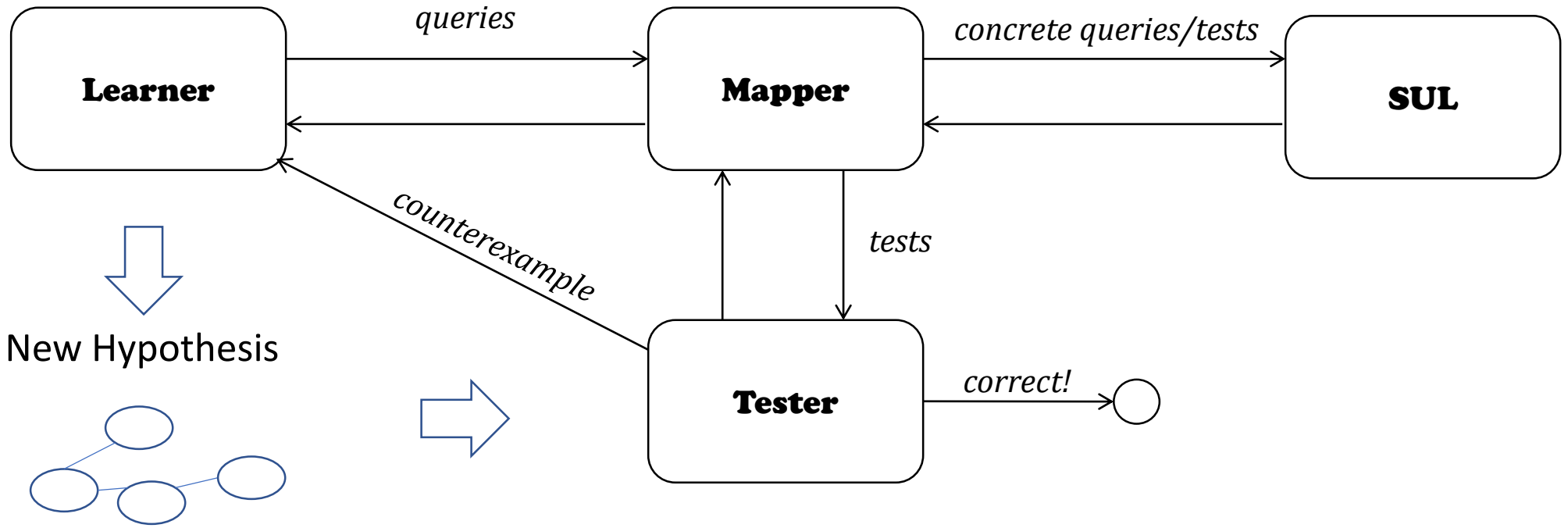
1. translates:
   - between abstract and param. i/o
   - between param. i/o and concrete i/o
2. gives a (deterministic) Mealy Machine representation
   - removes time dependencies, non-determinism..

Model Learning

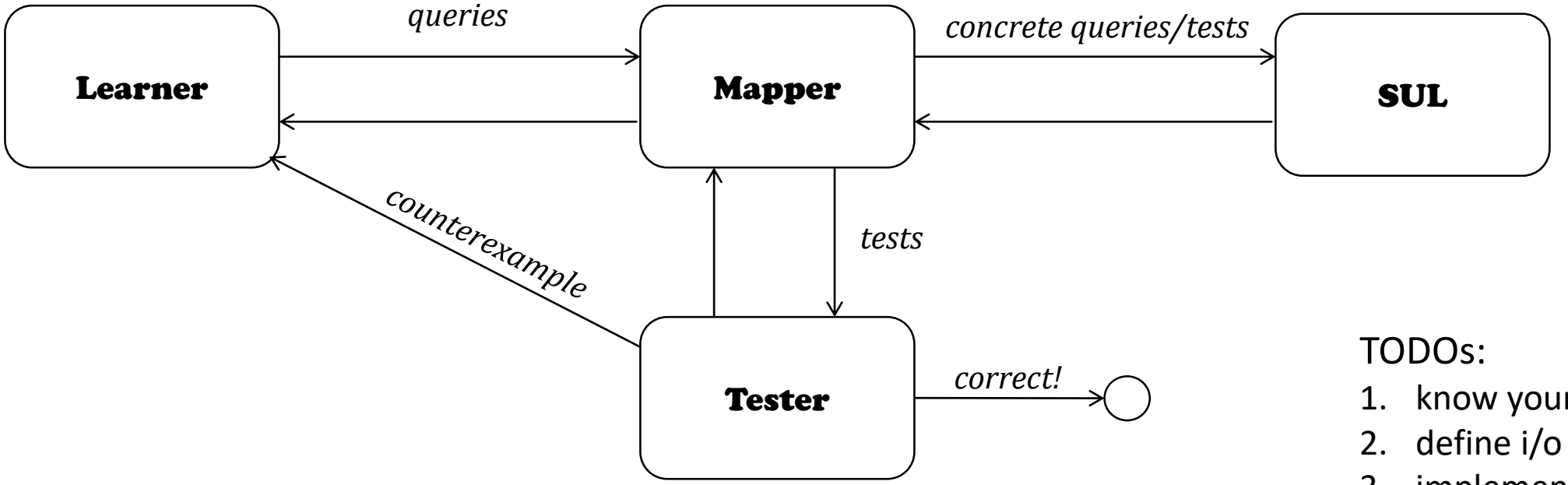




Model Learning



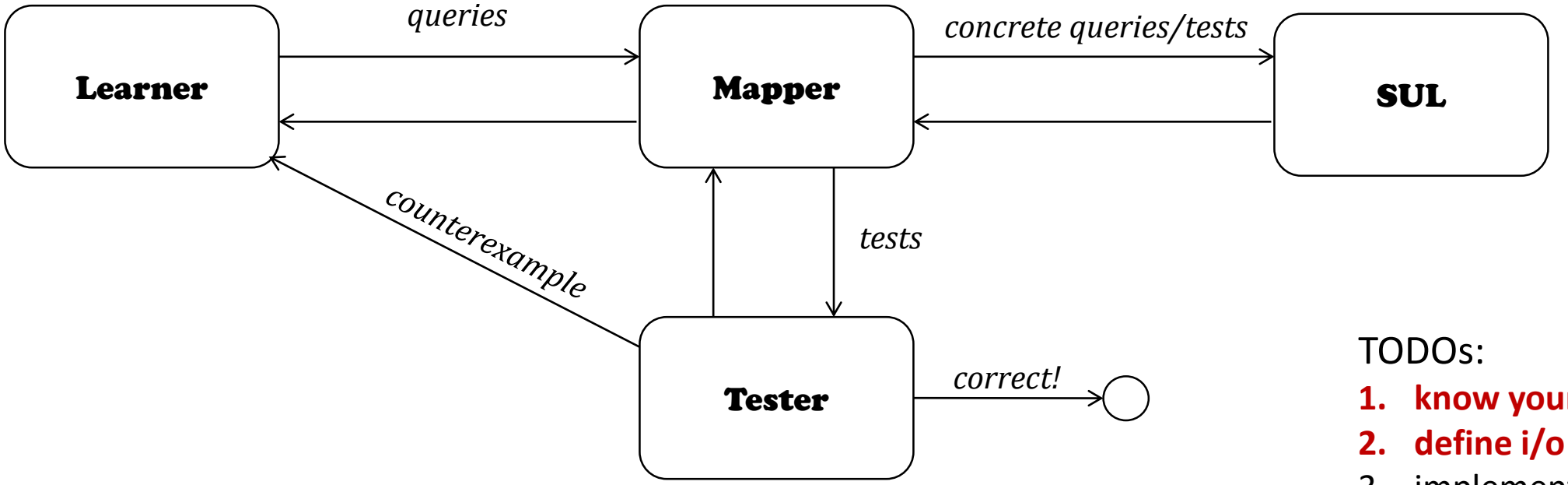
Model Learning



TODOs:

1. know your SUL
2. define i/o alphabet
3. implement mapper
4. choose learner and tester algorithms
5. connect and execute!

Model Learning



TODOs:

1. **know your SUL**
2. **define i/o alphabet**
3. implement mapper
4. choose learner and tester algorithms
5. connect and execute!

The SSH Protocol

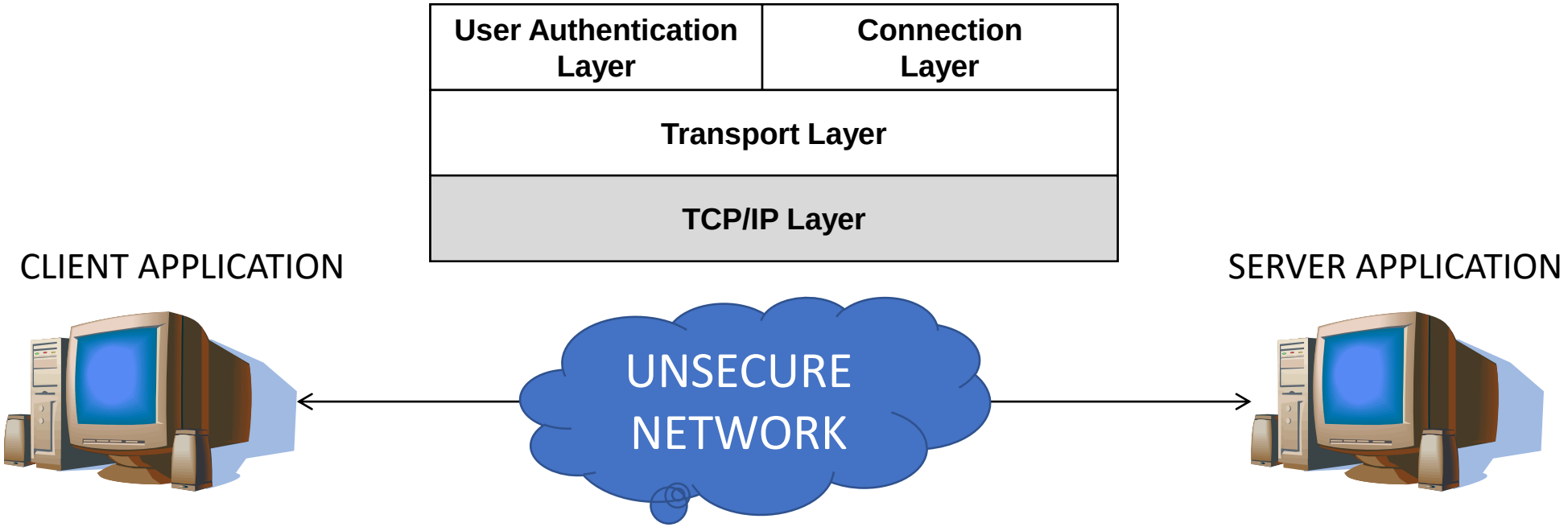
- protocol for operating network services (e.g. terminal) securely over an unsecured network
- client/server application layer protocol, runs on top of TCP

The SSH Protocol

- protocol for operating network services (e.g. terminal) securely over an unsecured network
- client/server application layer protocol, runs on top of TCP
- Learner + Mapper replaces the SSH CLIENT, goal learn the SSH Server!

The SSH Protocol

- comprises three layers which *interoperate (no encapsulation)*
- each layer responsible for each of the 3 protocol steps,
- for each we define the *happy flow* at an abstract level



The SSH Protocol

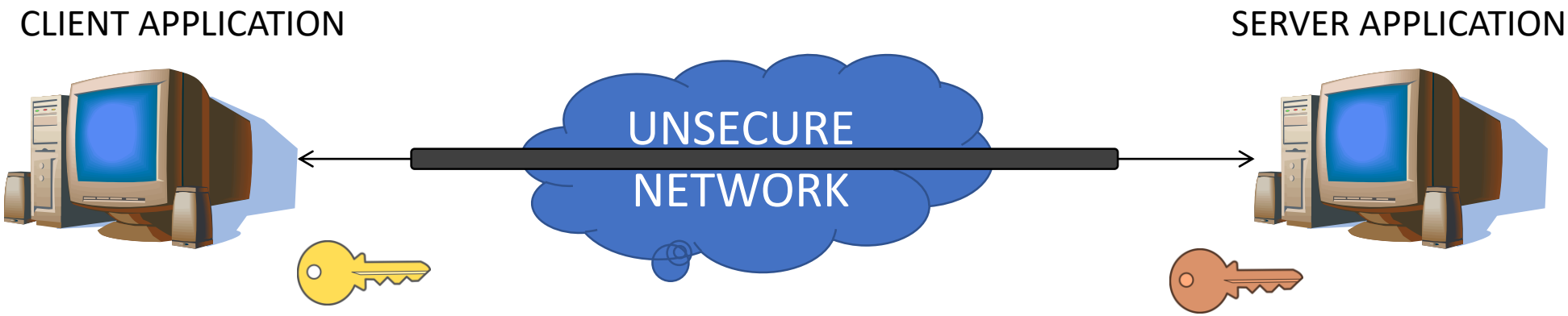
- 3 steps
  1. establish a secure connection (by exchanging keys)

User Authentication Layer	Connection Layer
Transport Layer	
TCP/IP Layer	

The SSH Protocol

- 3 steps
  1. establish a secure connection (by exchanging keys)

User Authentication Layer	Connection Layer
Transport Layer	
TCP/IP Layer	



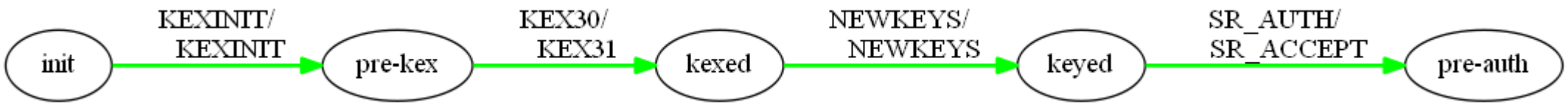
The SSH Protocol

User Authentication Layer	Connection Layer
Transport Layer	
TCP/IP Layer	

- 3 steps
  1. establish a secure connection (by exchanging keys)

     1. exchange preferences (KEXINIT)
     2. perform key exchange (KEXxx)
     3. put new keys to use (NEWKEYS)
     4. engage the auth. service (SR_AUTH)

Happy flow:

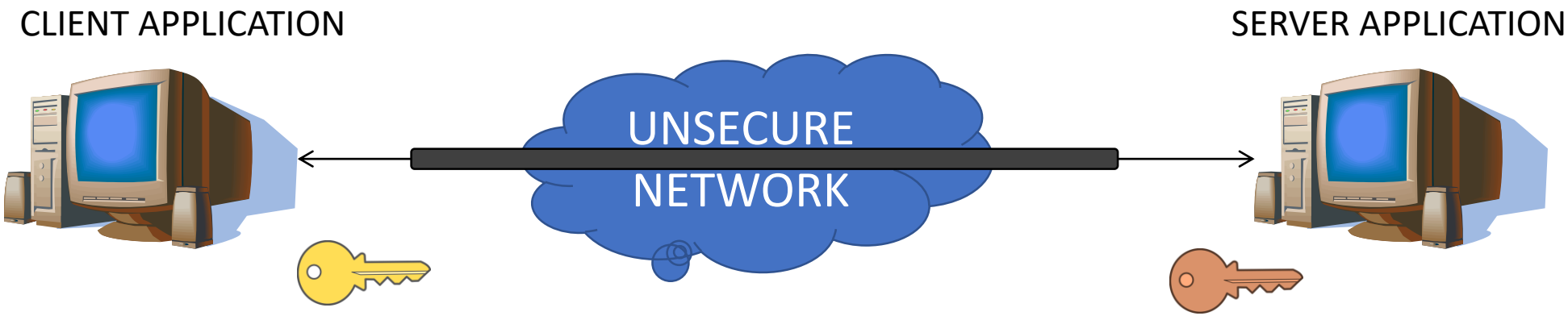


Other inputs: DEBUG, IGNORE, DISCONNECT..
Other outputs: DEBUG, IGNORE, DISCONNECT..

The SSH Protocol

- 3 steps
  1. establish a secure connection (by exchanging keys)

User Authentication Layer	Connection Layer
Transport Layer	
TCP/IP Layer	



The SSH Protocol

User Authentication Layer	Connection Layer
Transport Layer	
TCP/IP Layer	

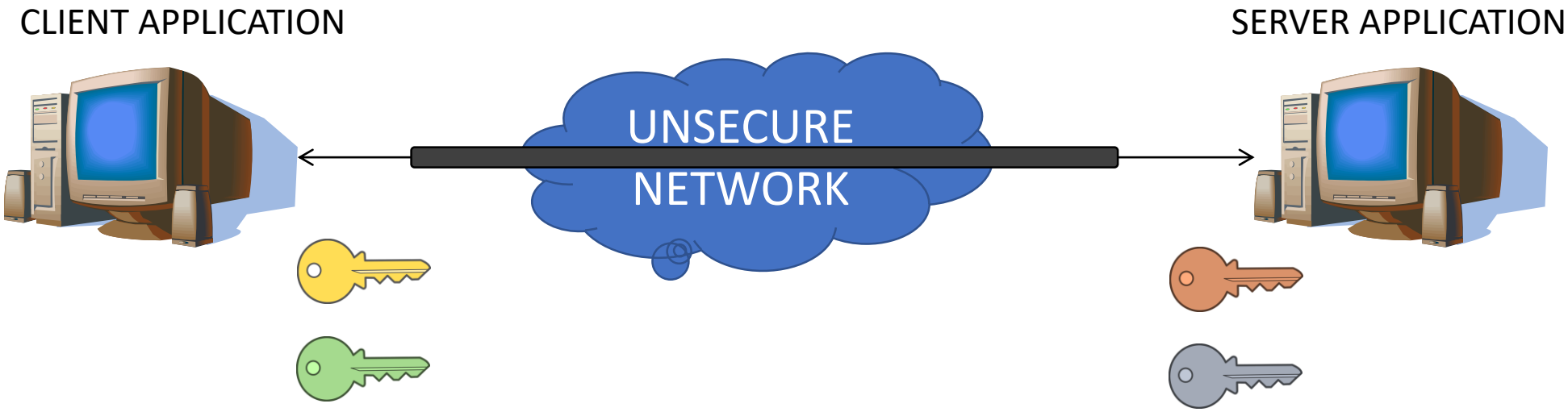
- 3 steps
  1. establish a secure connection (by exchanging keys)

     key re-exchange (rekey): same procedure, old keys

     are replaced by new ones

     can happen any time after the initial key exchange protocol

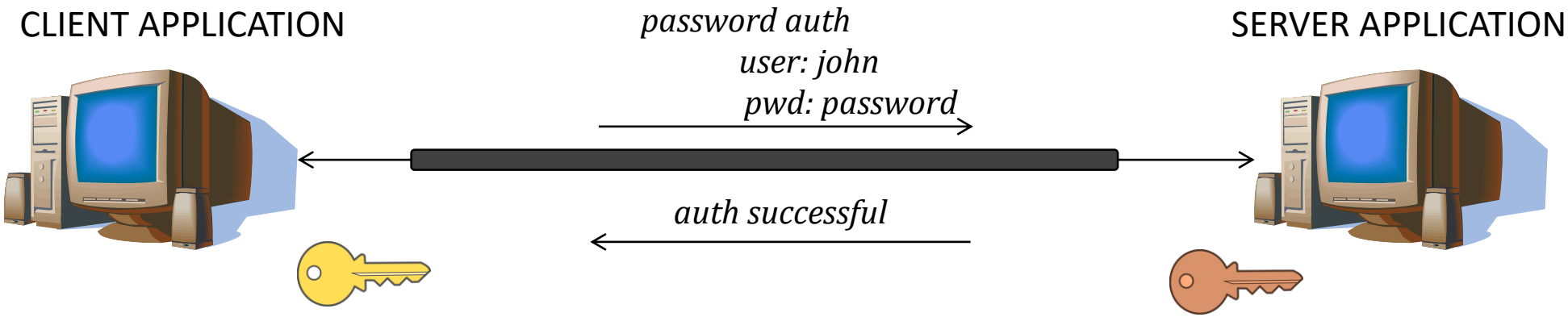
     should not affect operation of higher layer protocols



The SSH Protocol

User Authentication Layer	Connection Layer
Transport Layer	
TCP/IP Layer	

- 3 steps
  1. establish a secure connection (by exchanging keys)
  2. authentication with server

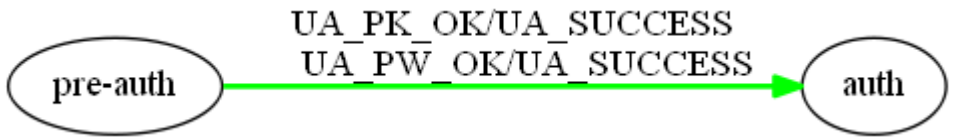


The SSH Protocol

User Authentication Layer	Connection Layer
Transport Layer	
TCP/IP Layer	

- 3 steps
  1. establish a secure connection (by exchanging keys)
  2. authentication with server
     - user/public key auth. UA_PK_OK
     - user/password auth. UA_PW_OK
     - none auth. UA_NONE

Happy flow:

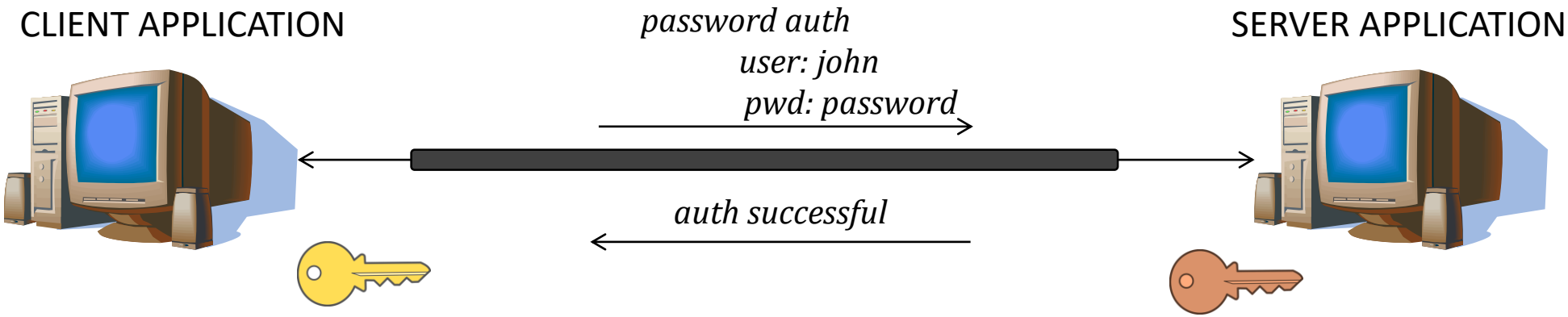


Other inputs: UA_NONE, UA_PK_NOK, UA_PW_NOK…
Other outputs: UA_FAILURE

The SSH Protocol

User Authentication Layer	Connection Layer
Transport Layer	
TCP/IP Layer	

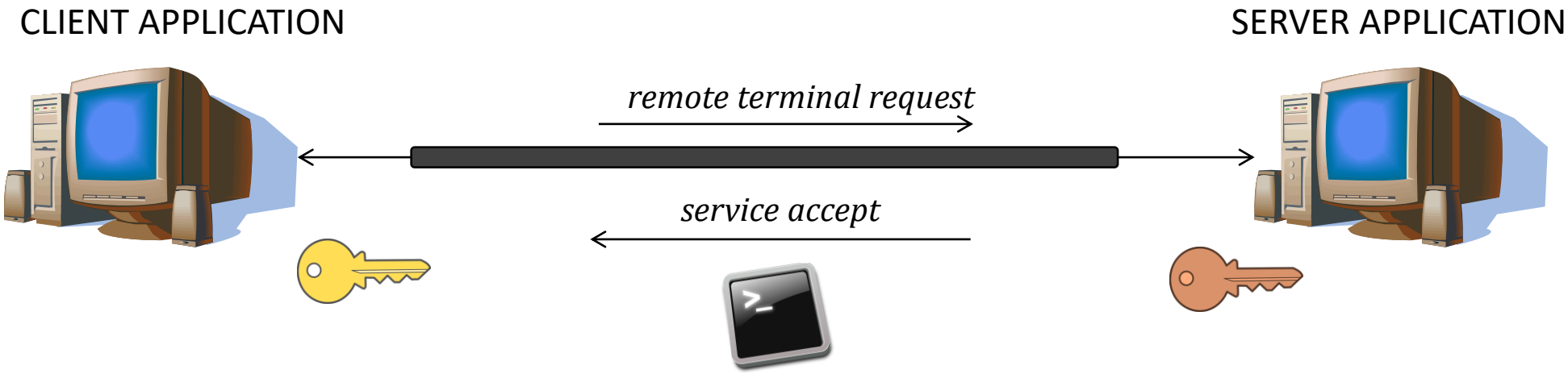
- 3 steps
  1. establish a secure connection (by exchanging keys)
  2. authentication with server



The SSH Protocol

User Authentication Layer	Connection Layer
Transport Layer	
TCP/IP Layer	

- 3 steps
  1. establish a secure connection (by exchanging keys)
  2. authentication with server
  3. access network services (say remote terminal)

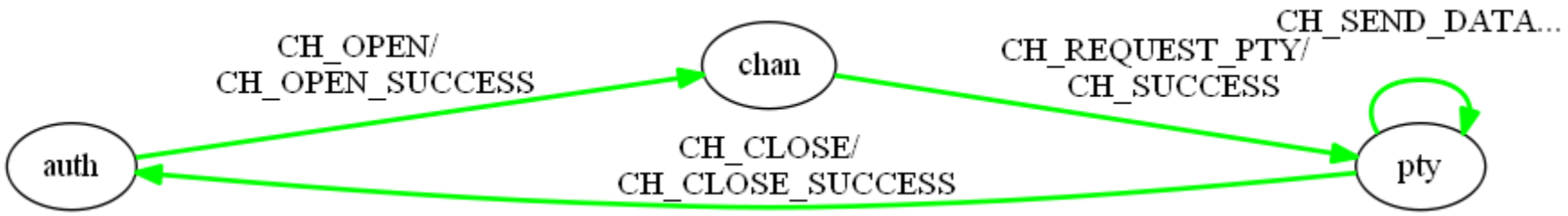


Learning SSH

User Authentication Layer	Connection Layer
Transport Layer	
TCP/IP Layer	

- 3 steps
  1. establish a secure connection (by exchanging keys)
  2. authentication with server
  3. access network services (say remote terminal)

     1) open channel (CH_OPEN)
     2) request term. service over channel (CH_REQUEST_PTY)
     3) channel data management (CH_SEND_DATA)
     4) close channel (CH_CLOSE)



Learning SSH

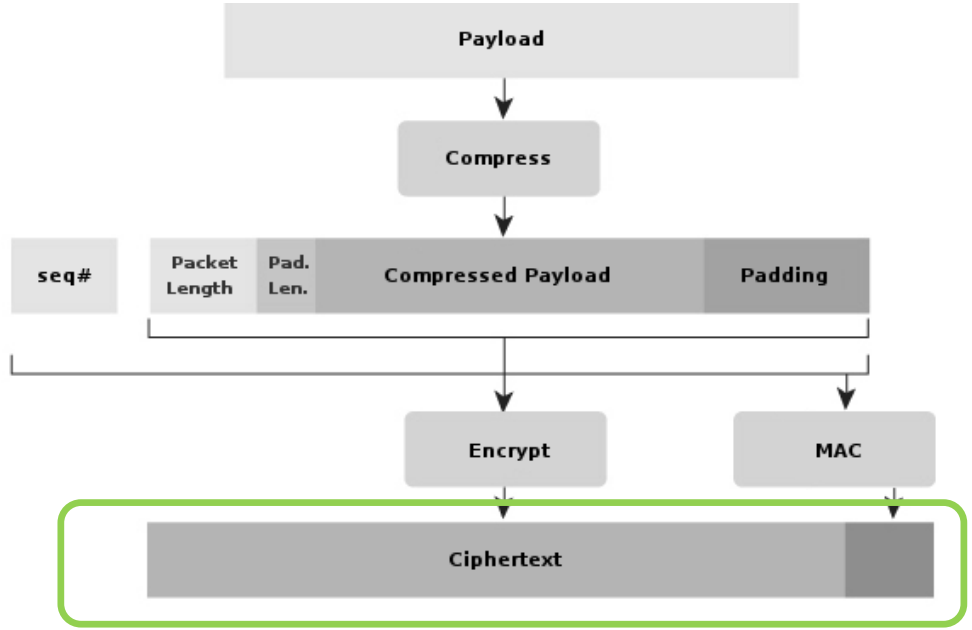
TODOs:

1. know your SUL
2. define i/o alphabet
3. **implement mapper**
4. choose learner and tester algorithms
5. connect and execute!

Mapper task

1. *translate between abstract, parametrized and concrete i/o*

AUTH_PW_OK AUTH_REQUEST(“password”, “john”…)



Learning SSH

TODOs:

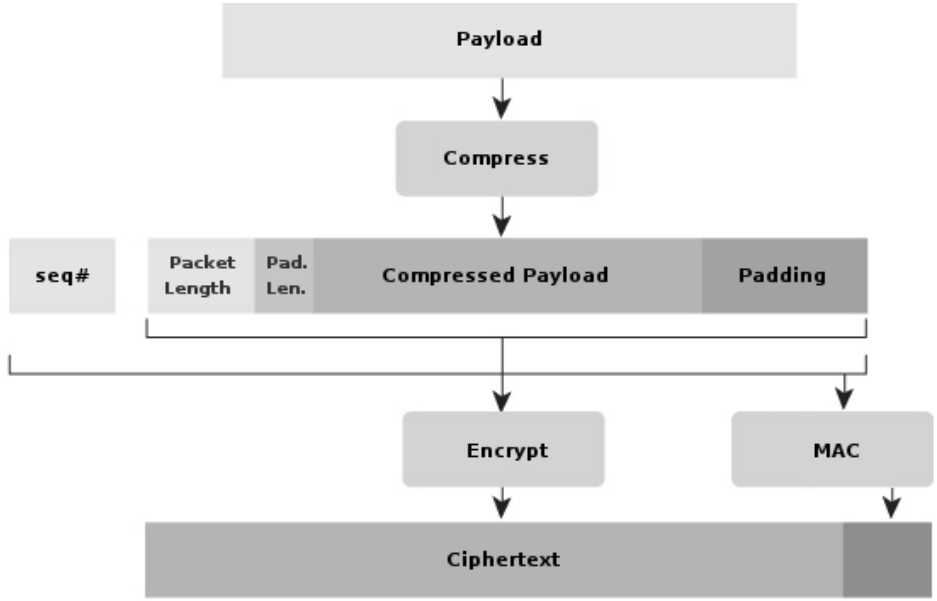
1. know your SUL
2. define i/o alphabet
3. **implement mapper**
4. choose learner and tester algorithms
5. connect and execute!

Mapper task

1. *translate between abstract, parametrized and concrete i/o*
   - *needs to be able to encrypt/decrypt compress/decompress*
   - *stores information in variables: encryption keys, session ID, sequence number…*

     ➔ *implemented by adapting an existing SSH suite implementation (Paramiko)*

AUTH_PW_OK AUTH_REQUEST(“password”, “john”…)



Learning SSH

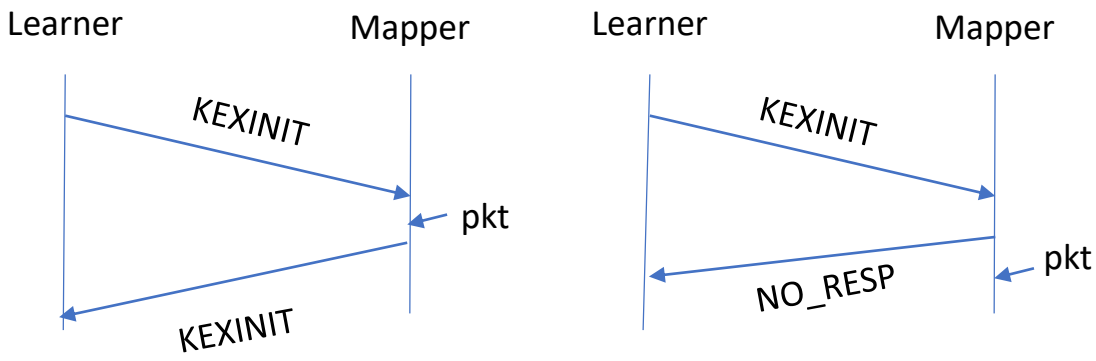
TODOs:

1. know your SUL
2. define i/o alphabet
3. **implement mapper**
4. choose learner and tester algorithms
5. connect and execute!

Mapper task

1. *translate between abstract, parametrized and concrete i/o*
   - *needs to be able to encrypt/decrypt compress/decompress*
   - *stores information in variables: encryption keys, session ID, sequence number…*

     ➔ *implemented by adapting an existing SSH suite implementation (Paramiko)*
2. *ensure deterministic Mealy Machine representation*
   - *reliable setting of timing parameters (e.g. NO_RESP timing parameter)*



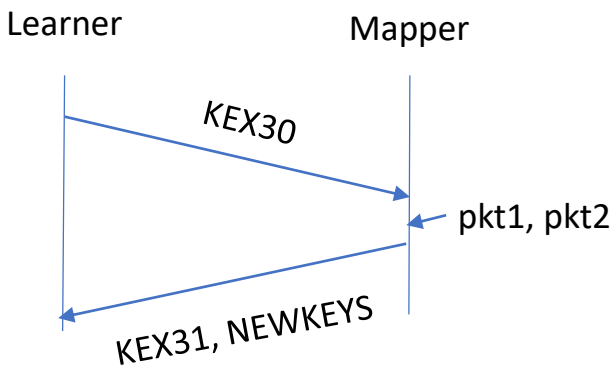
false NO_RESP, mapper should have waited longer

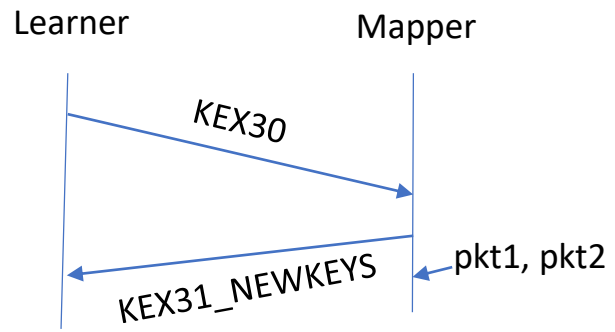
Learning SSH

TODOs:
1. know your SUL
2. define i/o alphabet
3. **implement mapper**
4. choose learner and tester algorithms
5. connect and execute!

Mapper task

1. *translate between abstract, parametrized and concrete i/o*
   - *needs to be able to encrypt/decrypt compress/decompress*
   - *stores information in variables: encryption keys, session ID, sequence number…*
     - ➔ *implemented by adapting an existing SSH suite implementation (Paramiko)*
2. *ensure deterministic Mealy Machine representation*
   - *reliable setting of timing parameters (e.g. NO_RESP timing parameter)*
   - *enforce one output per input by concatenating ('_') multiple responses to an input into one output*



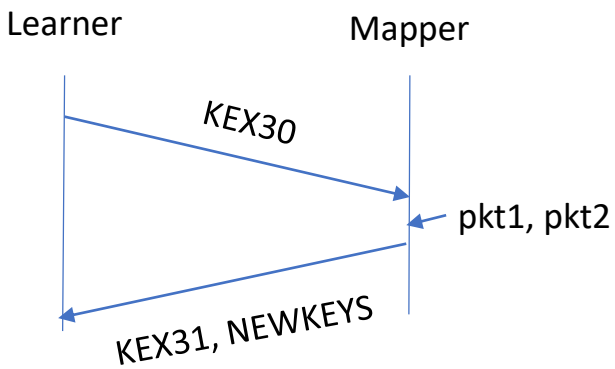


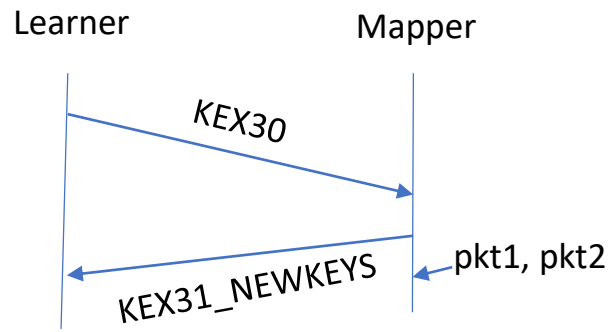
Learning SSH

TODOs:
1. know your SUL
2. define i/o alphabet
3. **implement mapper**
4. choose learner and tester algorithms
5. connect and execute!

Mapper task

1. *translate between abstract, parametrized and concrete i/o*
   - *needs to be able to encrypt/decrypt compress/decompress*
   - *stores information in variables: encryption keys, session ID, sequence number…*
     - ➔ *implemented by adapting an existing SSH suite implementation (Paramiko)*
2. *ensure deterministic Mealy Machine representation*
   - *reliable setting of timing parameters (e.g. NO_RESP timing parameter)*
   - *enforce one output per input by concatenating ('_') multiple responses to an input into one output*





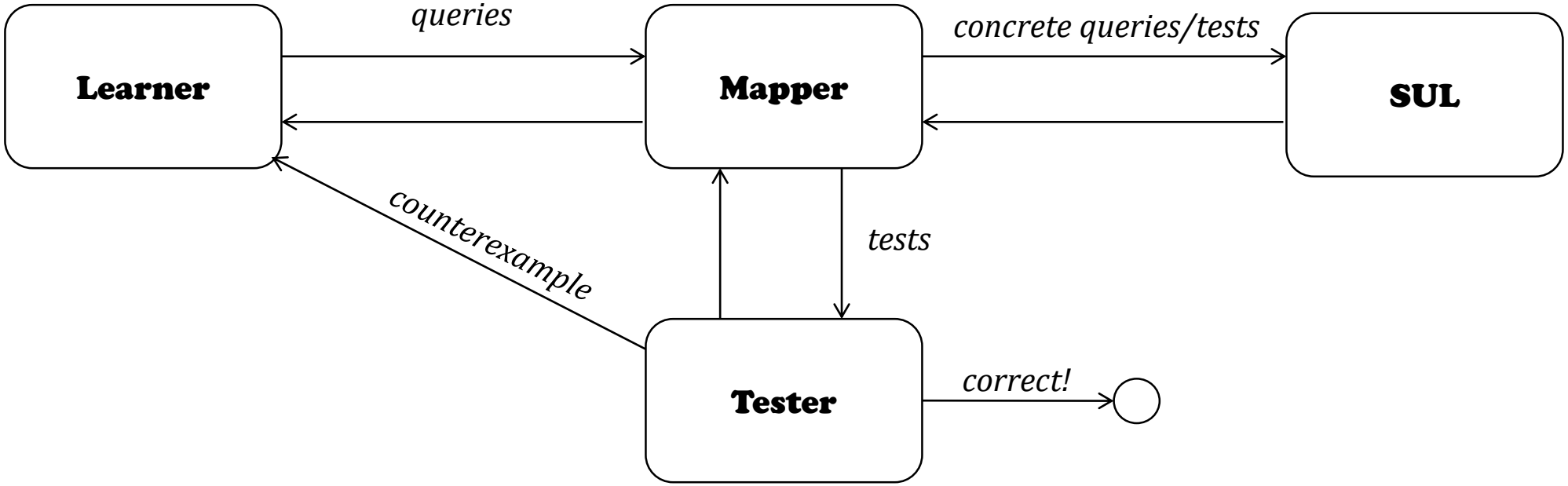
Learning SSH

TODOs:

1. know your SUL
2. define i/o alphabet
3. implement mapper
4. **choose learner and tester algorithms**
5. connect and execute!

LearnLib algorithms:

L* , **Observation Pack**



Tester Algorithms:

Random Walk, W Method,

Yannakakis (Random + Exhaustive)

Learning SSH

TODOs:

1. know your SUL
2. define i/o alphabet
3. implement mapper
4. **choose learner and tester algorithms**
5. connect and execute!

Note on testing:

- testing can never guarantee correctness
- exhaustive test algs. ensure a well defined level of confidence
  - *but lack penetration*
- random test algs. have penetration ➔ more likely to find CEs
  - *but give no formal confidence*

Learning SSH

TODOs:

1. know your SUL
2. define i/o alphabet
3. implement mapper
4. **choose learner and tester algorithms**
5. connect and execute!

Example Yannakakis

→random:

- choose bigger k
- random mid sequences

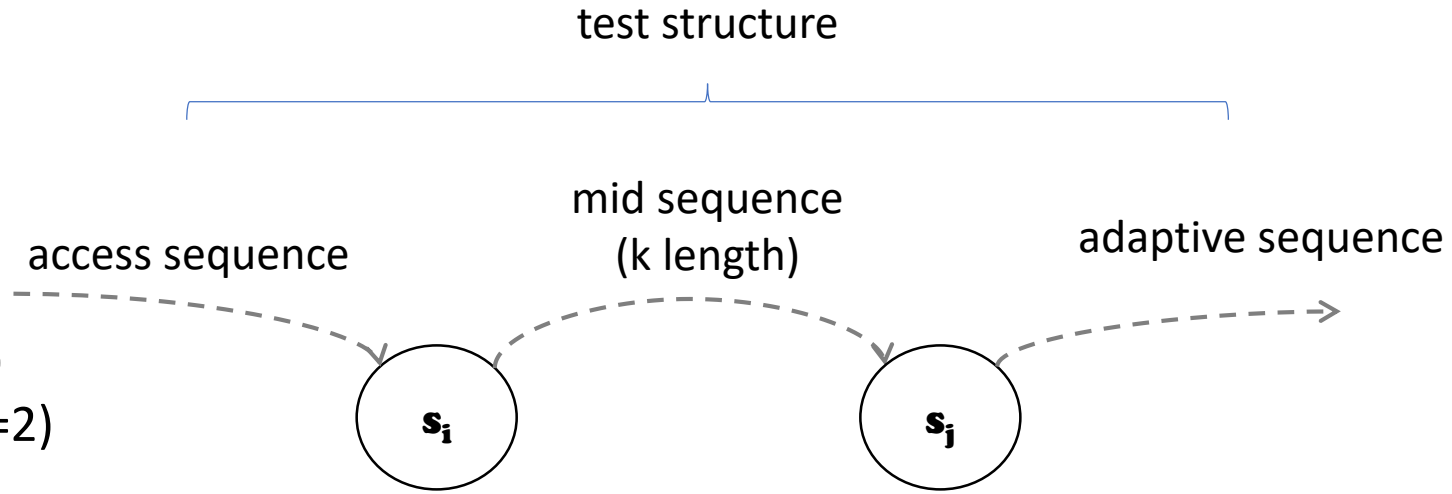
→exhaustive

- choose smaller k
- generate for all mid-sequences
- attain confidence

We used:

Random Yannakakis(k=4)

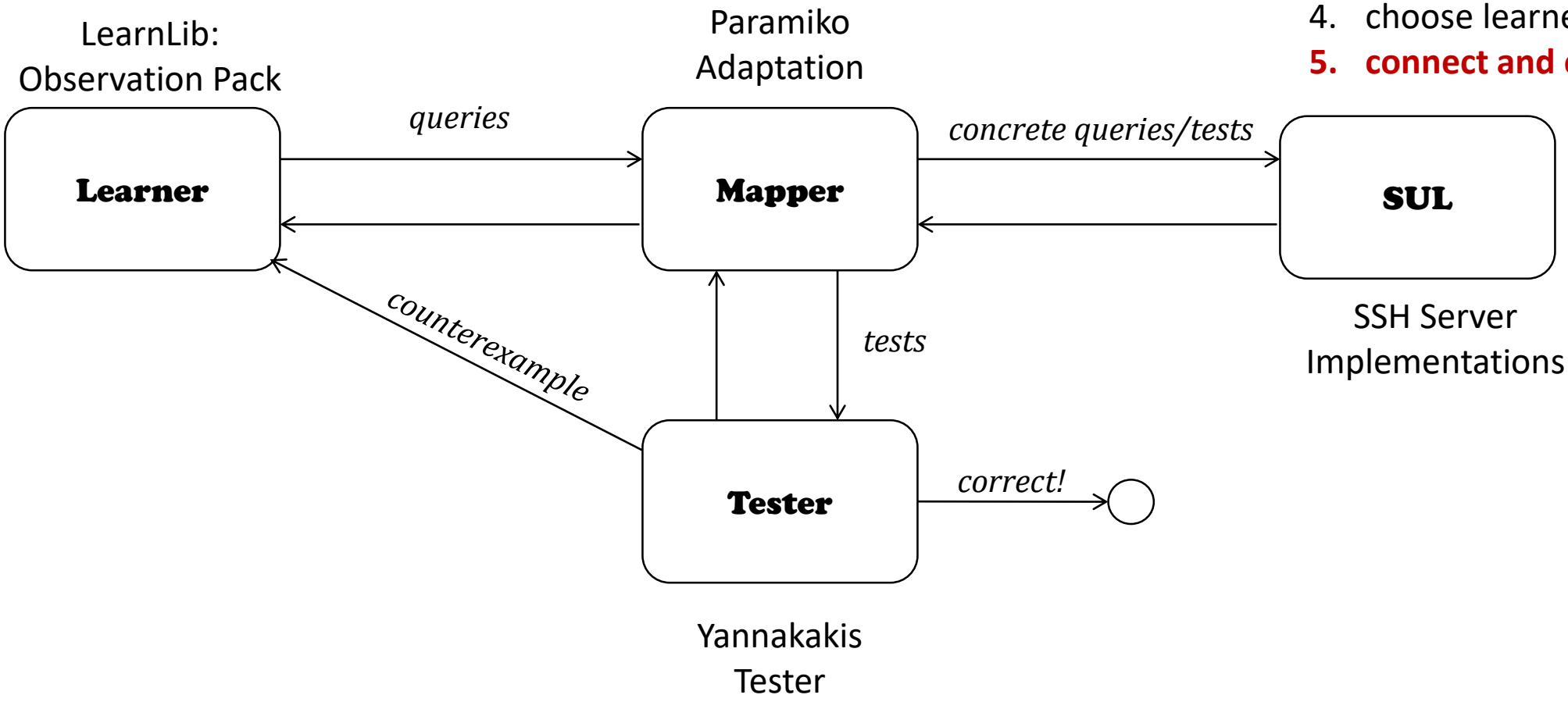
Exhaustive Yannakakis(k=2)



Learning SSH

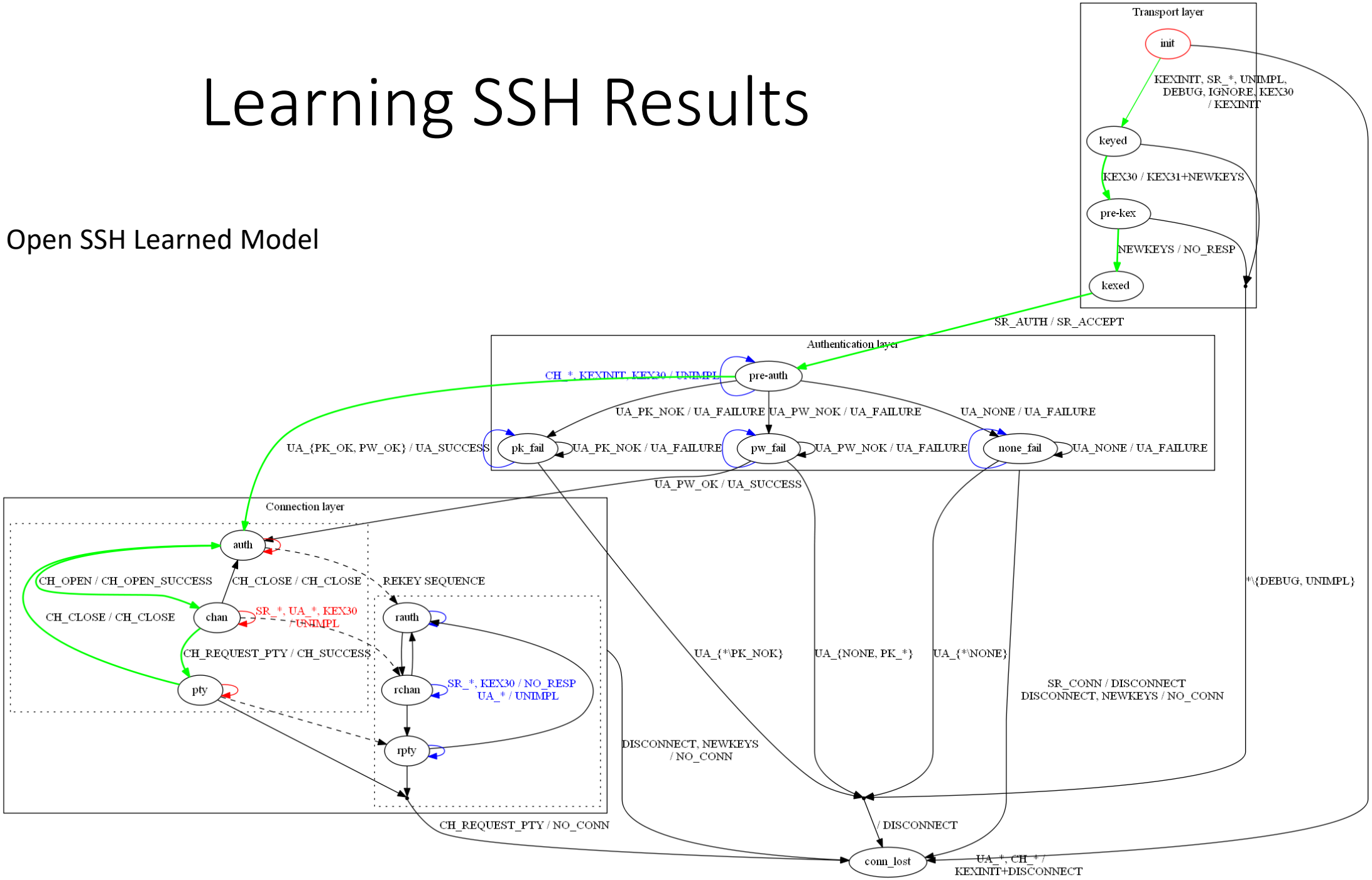
TODOs:

1. know your SUL
2. define i/o alphabet
3. implement mapper
4. choose learner and tester algorithms
5. **connect and execute!**



Learning SSH Results

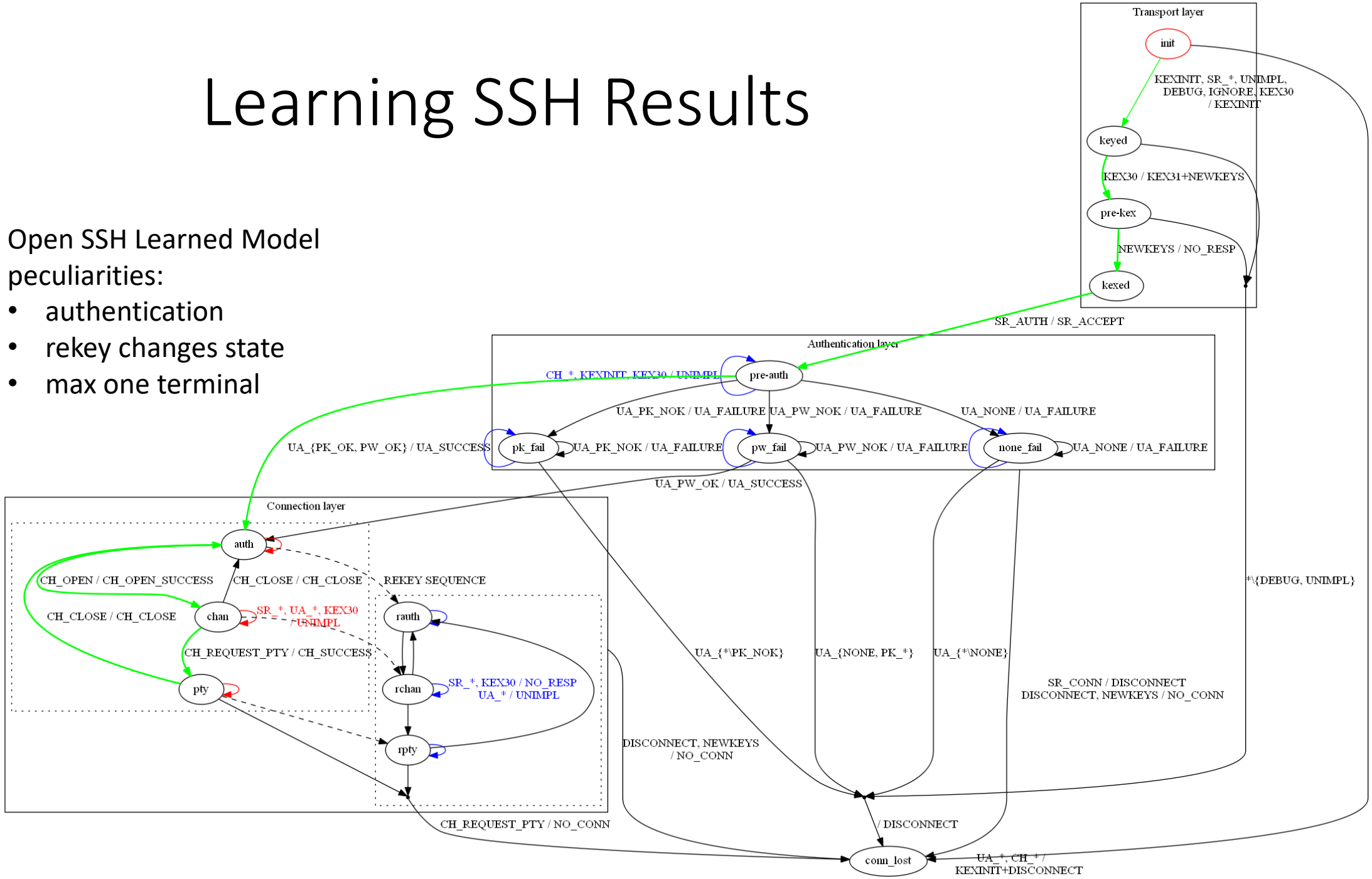
Open SSH Learned Model



Learning SSH Results

Open SSH Learned Model peculiarities:

- authentication
- rekey changes state
- max one terminal



Learning SSH Results

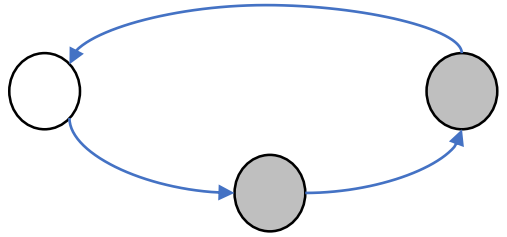
SUT	States	Hypotheses	Mem. Q.	Test Q.
OpenSSH 6.9p1-2	31	4	19836	76418
BitVise 7.23	65	15	24996	58423
DropBear v2014.65	29	8	8357	64478

- rekey (3 step sequence)
- buffering
- mapper induced behavior

Learning SSH Results

SUT	States	Hypotheses	Mem. Q.	Test Q.
OpenSSH 6.9p1-2	31	4	19836	76418
BitVise 7.23	65	15	24996	58423
DropBear v2014.65	29	8	8357	64478

- **rekey (3 step sequence)**
- buffering
- mapper induced behavior



state permitting rekey

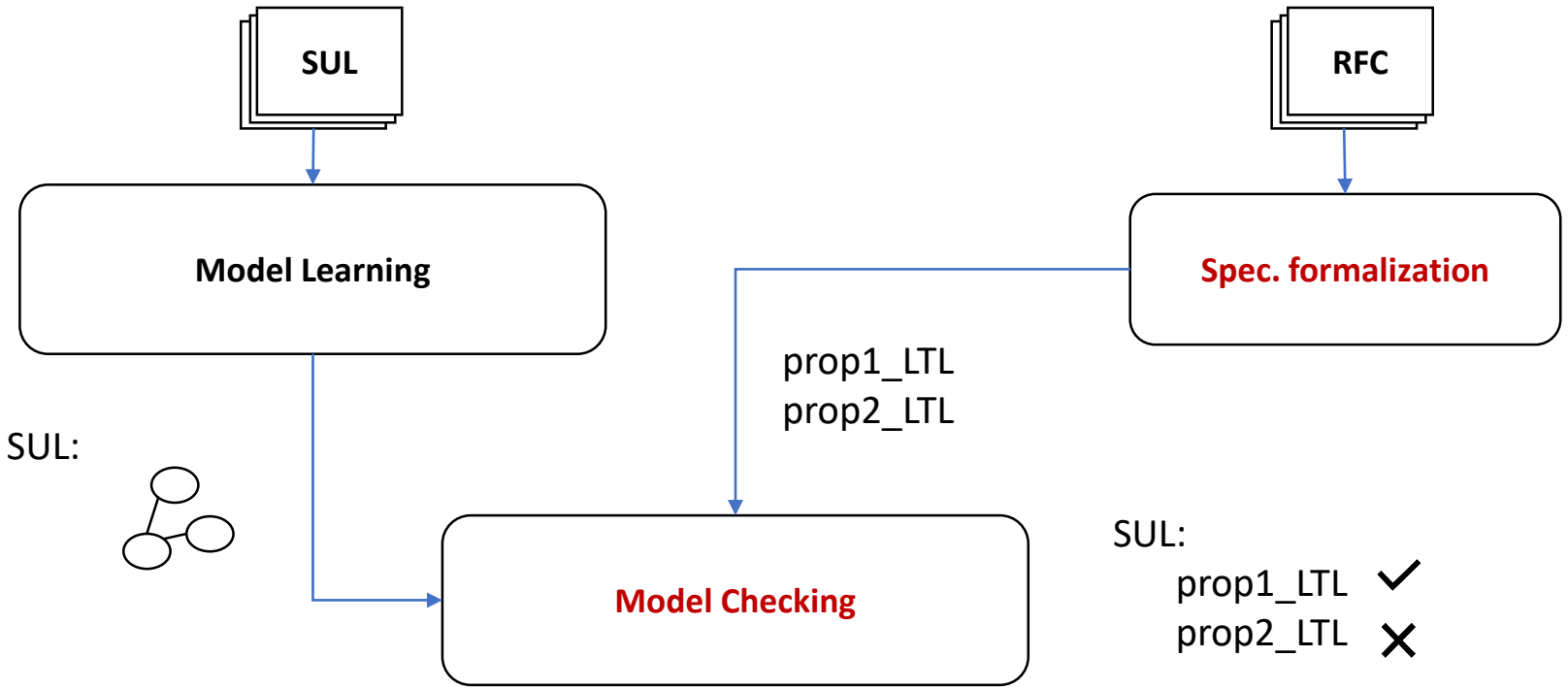
rekey state

Learning SSH Results

SUT	States	Hypotheses	Mem. Q.	Test Q.
OpenSSH 6.9p1-2	31	4	19836	76418
BitVise 7.23	65	15	24996	58423
DropBear v2014.65	29	8	8357	64478

- rekey (3 step sequence)
- buffering
- **mapper induced behavior**
  - remember, we learn SUL + mapper, not SUL alone

What was done



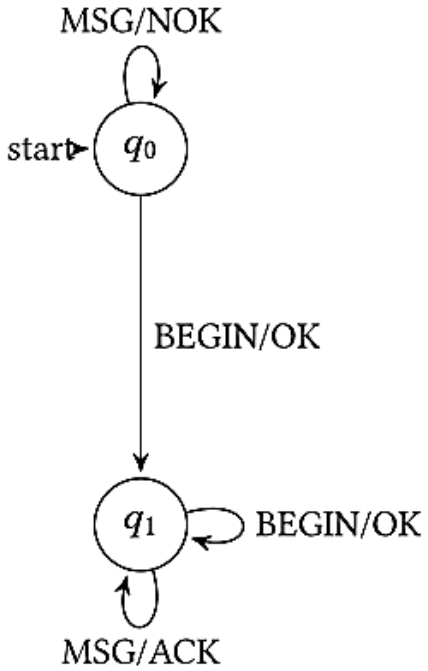
Model Checking

- we used NuSMV:
  - supports LTL, CTL and Real Time CTL specifications
  - requires conversion to a .SMV model

Model Checking

- we used NuSMV:
  - supports LTL, CTL and Real Time CTL specifications
  - requires conversion to a .SMV model
  - wrote script to automatically perform this conversion

Mealy Machine



NuSMV Model

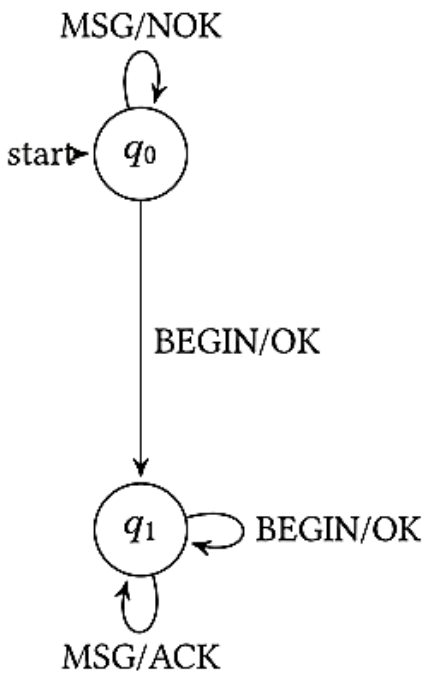
```
MODULE main
  VAR state : {q0, q1};
  inp : {BEGIN, MSG};
  out : {OK, NOK, ACK};
  ASSIGN
  init(state) := q0;
  next(state) := case
    state = q0 & inp = BEGIN: q1;
    state = q0 & inp = MSG: q0;
    state = q1 & inp = BEGIN: q1;
    state = q1 & inp = MSG: q1;
    esac;
  out := case
    state = q0 & inp = BEGIN: OK;
    state = q0 & inp = MSG: NOK;
    state = q1 & inp = BEGIN: OK;
    state = q1 & inp = MSG: ACK;
  esac;
```

➔kripke structure with:
- state function(next)
- output function(out)

Model Checking

- we used NuSMV:
  - supports LTL, CTL and Real Time CTL specifications
  - requires conversion to a .SMV model

Mealy Machine



NuSMV Model

```
MODULE main
  VAR state : {q0, q1};
  inp : {BEGIN, MSG};
  out : {OK, NOK, ACK};
  ASSIGN
  init(state) := q0;
  next(state) := case
    state = q0 & inp = BEGIN: q1;
    state = q0 & inp = MSG: q0;
    state = q1 & inp = BEGIN: q1;
    state = q1 & inp = MSG: q1;
    esac;
  out := case
    state = q0 & inp = BEGIN: OK;
    state = q0 & inp = MSG: NOK;
    state = q1 & inp = BEGIN: OK;
    state = q1 & inp = MSG: ACK;
  esac;
```

➔kripke structure with:
- state function(next)
- output function(out)

G (inp=BEGIN -> out=OK)

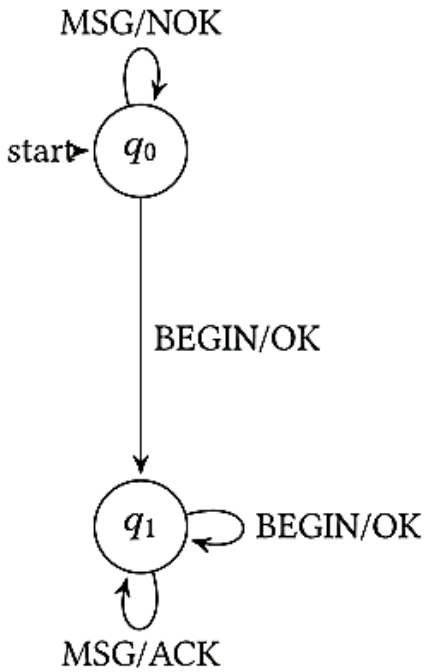
G (out=OK ->
X (inp=MSG -> out=ACK))

G U (out=OK ->
X (inp=MSG -> out=NOK))

Model Checking

- we used NuSMV:
  - supports LTL, CTL and Real Time CTL specifications
  - requires conversion to a .SMV model

Mealy Machine



NuSMV Model

```
MODULE main
  VAR state : {q0, q1};
  inp : {BEGIN, MSG};
  out : {OK, NOK, ACK};
  ASSIGN
  init(state) := q0;
  next(state) := case
    state = q0 & inp = BEGIN: q1;
    state = q0 & inp = MSG: q0;
    state = q1 & inp = BEGIN: q1;
    state = q1 & inp = MSG: q1;
    esac;
  out := case
    state = q0 & inp = BEGIN: OK;
    state = q0 & inp = MSG: NOK;
    state = q1 & inp = BEGIN: OK;
    state = q1 & inp = MSG: ACK;
  esac;
```

➔kripke structure with:
- state function(next)
- output function(out)

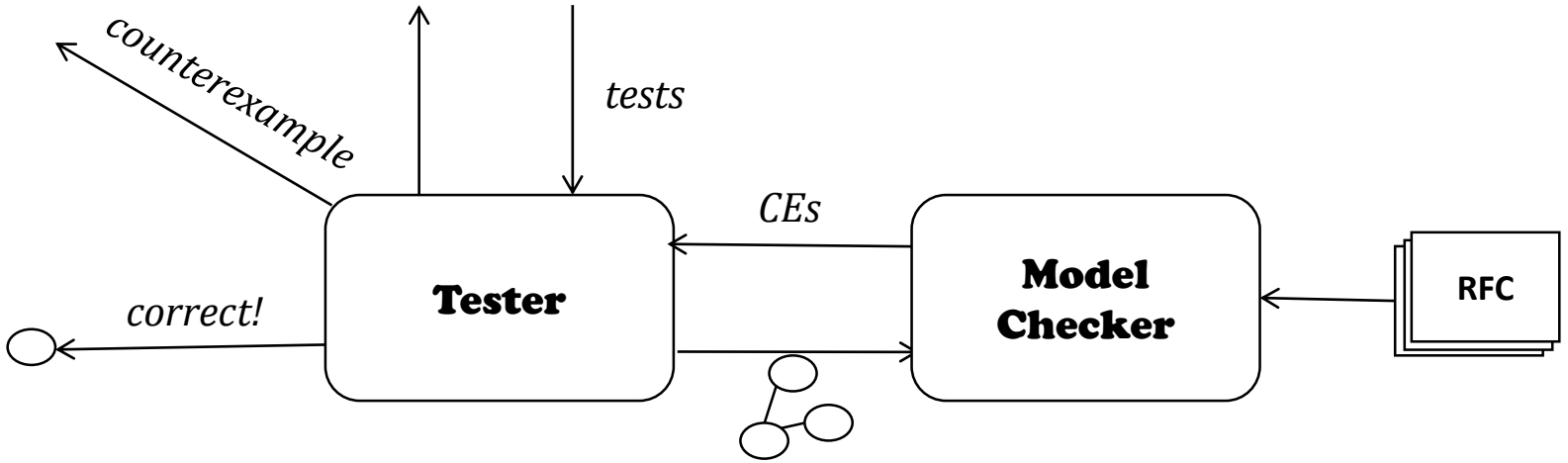
G (inp=BEGIN -> out=OK) ✓

G (out=OK ->
X (inp=MSG -> out=ACK)) ✓

G U (out=OK ->
X (inp=MSG -> out=NOK)) ✗

Model Checking

- we used NuSMV:
  - supports LTL, CTL and Real Time CTL specifications
  - requires conversion to a .SMV model
- specification either holds or counterexample (CE) given
  - CE may
    - agree with the SUL ➔ non-conformance
    - disagree with the SUL ➔ a CE for the learner
  - thus, all CEs must first be confirmed by running it on the system
    - integrated model checker into testing s.t. all CEs are confirmed



Formalizing SSH Specifications

- LTL formulas with both forward and past modalities
- checked on the mapper + SUL assembly (not only on the SUL itself), thus
  results not fully translatable
- 4 types:
  - basic properties: describe the SUL + mapper setup, all true
  - security properties: define the overriding goal of each layer
  - rekey properties: is rekey allowed (does it not disconnect)
    does rekey preserve state?
  - functional properties: are MUST/SHOULD statements met

Formalizing SSH Specifications

Only one SSH connection is made
and once it is gone, it is gone.

- basic properties
- security properties
- rekey properties
- functional properties

connection
is gone

mapper outputs
(w/o SUL intervention)

G (out=NO CONN – >
G (out=NO CONN | out=CH MAX | out=CH NONE))

SUL no longer responds

Formalizing SSH Specifications

We consider an transport layer state machine secure if there is:

no path from the initial state to the point where the authentication service is invoked without exchanging and employing cryptographic keys.

- basic properties
- security properties
- rekey properties
- functional properties

G (hasReqAuth – >
O ((inp=NEWKEYS & out=NO RESP) &
O ((inp=KEX30 & out=KEX31_NEWKEYS) &
O (out=KEXINIT))))

Formalizing SSH Specifications

SSH_MSG_CHANNEL_CLOSE
Upon receiving this message, a party MUST send back an SSH_MSG_CHANNEL_CLOSE unless it has already sent this message for the channel.
(RFC 4254, p 9)

- basic properties
- security properties
- rekey properties
- functional properties

G (hasOpenedChannel – >
((inp=CH CLOSE) – > (out=CH CLOSE))
W (connLost | kexStarted | out=CH CLOSE))

Formalizing SSH Specifications

SSH_MSG_CHANNEL_CLOSE
Upon receiving this message, a party MUST send back an SSH_MSG_CHANNEL_CLOSE unless it has already sent this message for the channel.
(RFC 4254, p 9)

- basic properties
- security properties
- rekey properties
- functional properties

G (hasOpenedChannel – >
((inp=CH CLOSE) – > (out=CH CLOSE))
W (connLost | kexStarted | out=CH CLOSE))

→ in red, predicates not expressed in RFC statement, yet deducted from context
→ formalization forces clarification

Formalizing SSH Specifications

SSH_MSG_USERAUTH_SUCCESS MUST be sent only once.
(RFC 4252 p. 5)

G (out=UA SUCCESS − > X G out != UA SUCCESS)

- basic properties
- security properties
- rekey properties
- functional properties

SSH_MSG_USERAUTH_SUCCESS has been sent, any further authentication requests received after that SHOULD be silently ignored.
(RFC 4252 p. 5)

G (out=UA SUCCESS − >
X ((authReq − > out=NO RESP) W (connLost | kexStarted)))

Formalizing SSH Specifications

key exchange does not affect the protocols that lie above the SSH transport layer.
(RFC 4253 p. 24)

- basic properties
- security properties
- rekey properties
- functional properties

- state based property:
  - ➔ cannot be efficiently formulated by LTL
  - ➔ checked using script

Model Checking Results

	Property	Key word	OpenSSH	Bitvise	DropBear
Security	Trans.		✓	✓	✓
	Auth.		✓	✓	✓
Rekey	Pre-auth.		X	✓	✓
	Auth.		✓	X	✓
Funct.	Prop. 6	MUST	✓	✓	✓
	Prop. 7	MUST	✓	✓	✓
	Prop. 8	MUST	X*	X	✓
	Prop. 9	MUST	✓	✓	✓
	Prop. 10	MUST	✓	✓	✓
	Prop. 11	SHOULD	X*	X*	✓
	Prop. 12	MUST	✓	✓	X

← UA_SUCC once

← NO_RESP after UA_SUCC

(*X sends UNIMPL)

← CH_CLOSE after CH_CLOSE

Conclusions and Future Work

Conformance checking of the SSH protocol,
using model learning & model checking

- inferred models for 3 SSH server implementations
- run extensive testing on models
- formalized and checked models against security properties, as well as server RFC MUST/SHOULD requirements
- found inconsistencies with limited security impact

Future work:

- formalize mapper so it is clear what it does (and a concretization can be made)
- make mapper abstraction less impactful ➔ reduce num. of mapper induced states
- learn SSH client, model check assembly client/server
- replace classical learner by a register automata learner, extract parameters and infer their related behavior