PISE

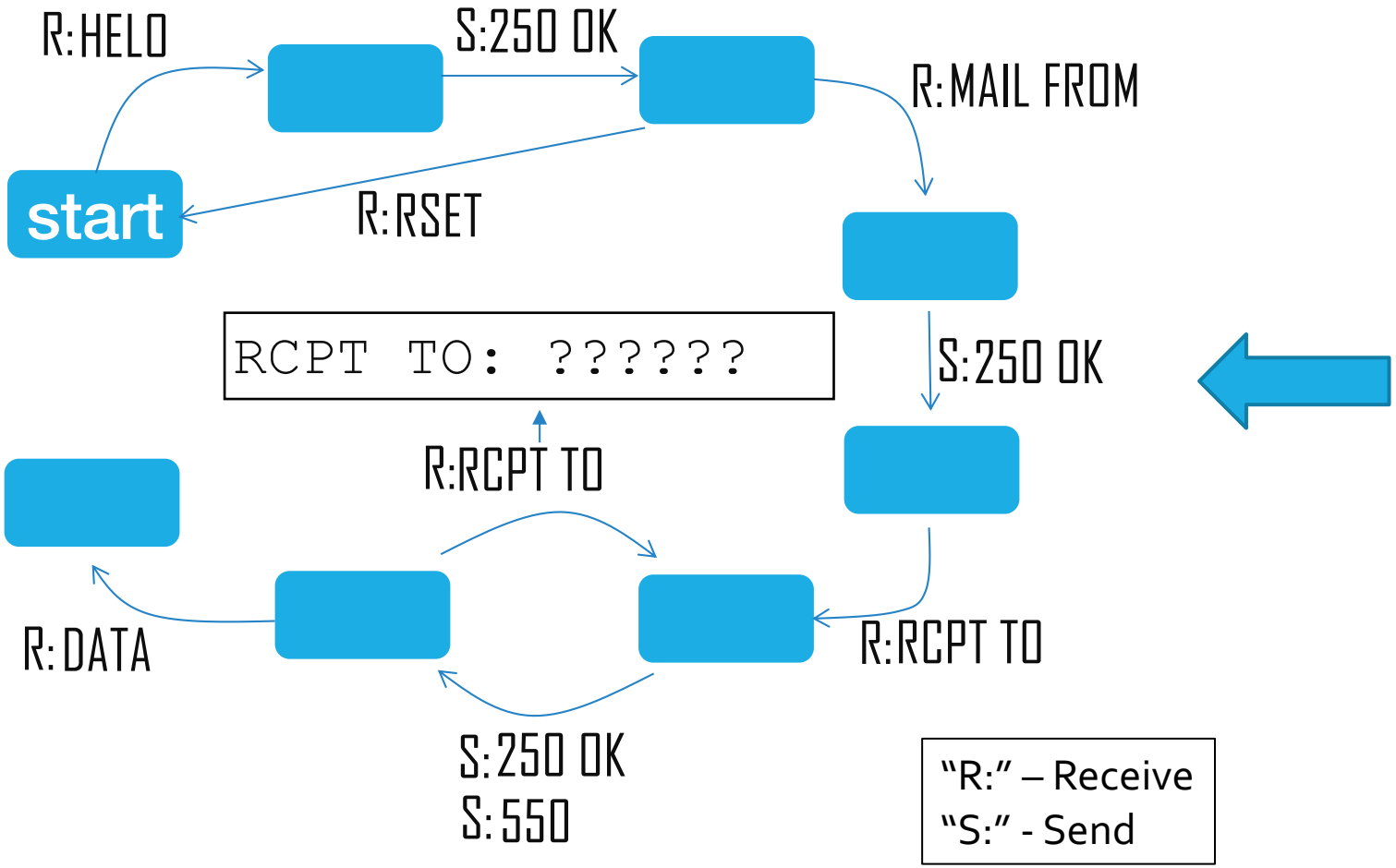
PROTOCOL INFERENCE USING SYMBOLIC EXECUTION AND AUTOMATA LEARNING

Ron Marcovich, Orna Grumberg, Gabi Nakibly

Technion – Israel Institute of Technology

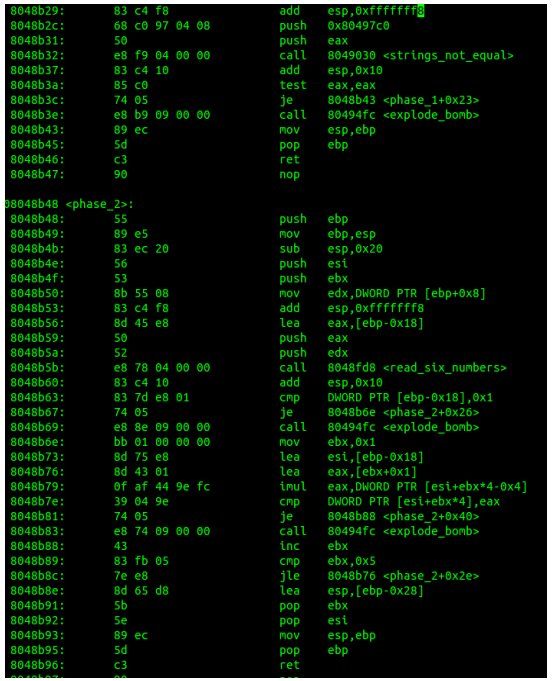
{ron.mar, orna, gnakibly}@cs.technion.ac.il

What is protocol inference?



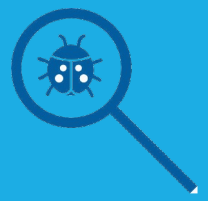
"R:" – Receive
"S:" - Send

Mail Server

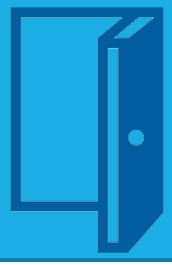


Motivations:

Finding Bugs



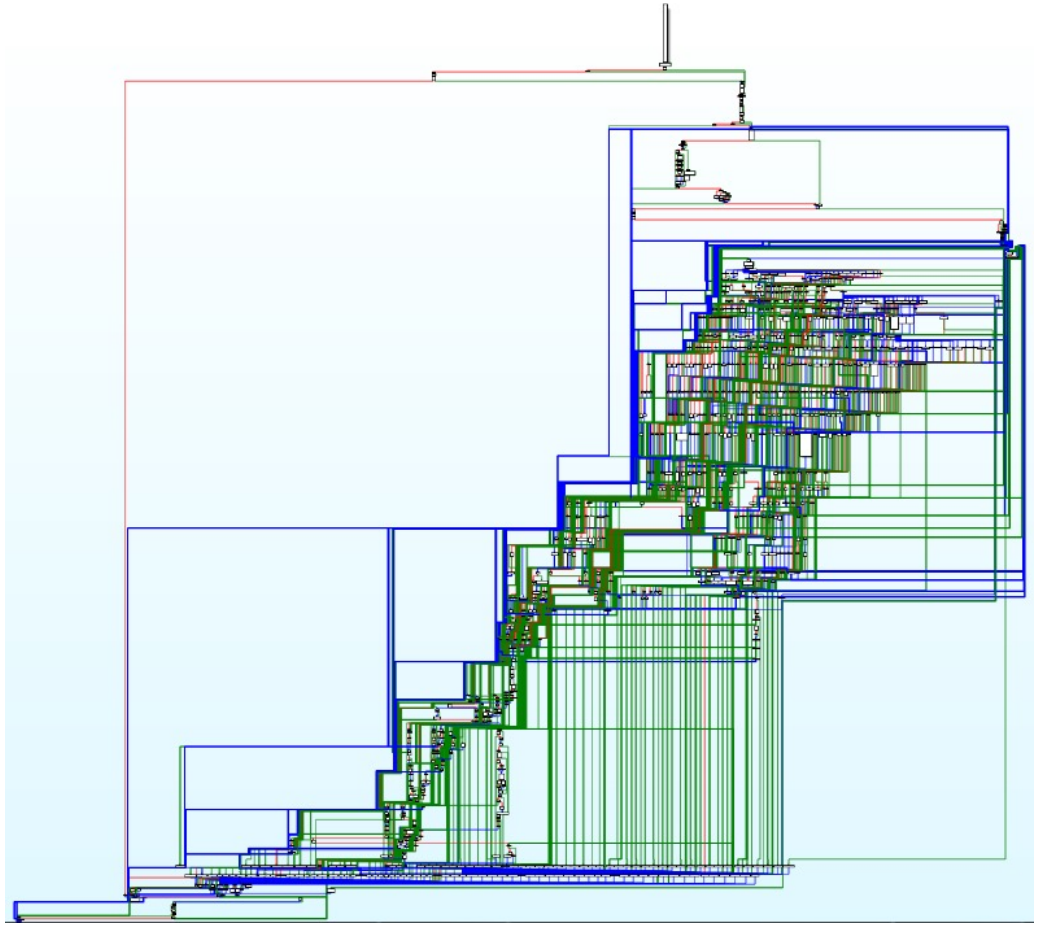
Finding Backdoors



Analyzing Malware

Manual Protocol Inference is Hard!

- It can take days or even weeks!



Research Goal

- Automatically infer the protocol
- Our input:
  - Binary code of a program
- Our output:
  - State machine of the protocol
  - Messages formats

Assumptions

We assume:

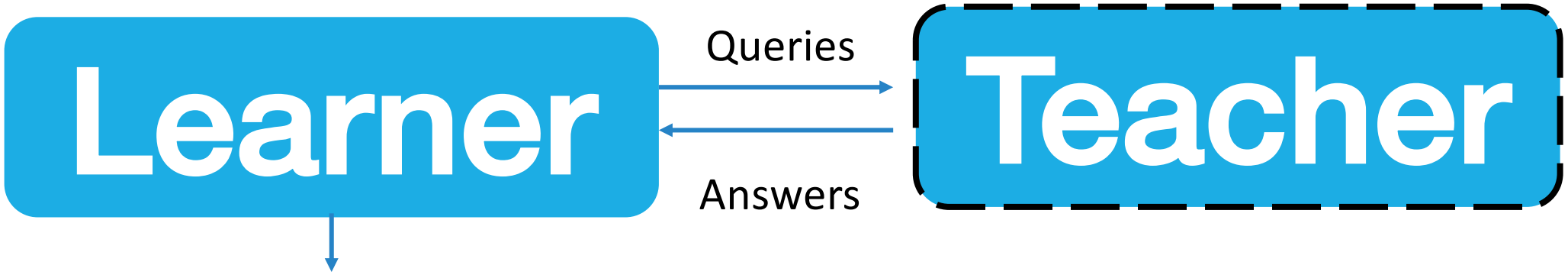
- Protocol Regularity

We do not assume:

- Past traffic captures
- Active protocol peer
- Source code
- Messages' formats

Under the Hood

Overview

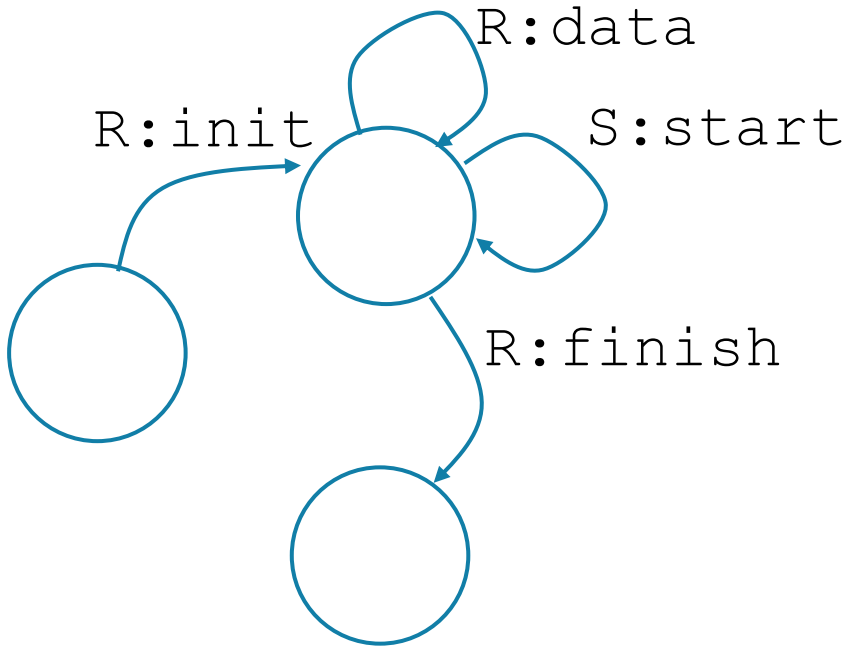


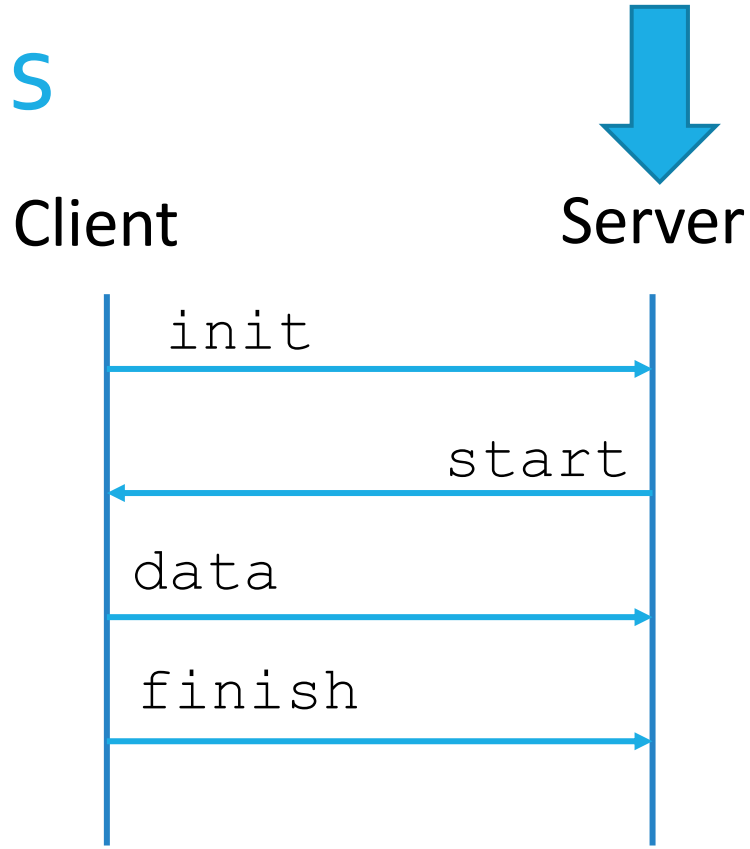
L* Algorithm

Alphabet = Message Types

L* Algorithm for protocols

- {R:init, S:start} ✓
- {R:init, R:init} ✗

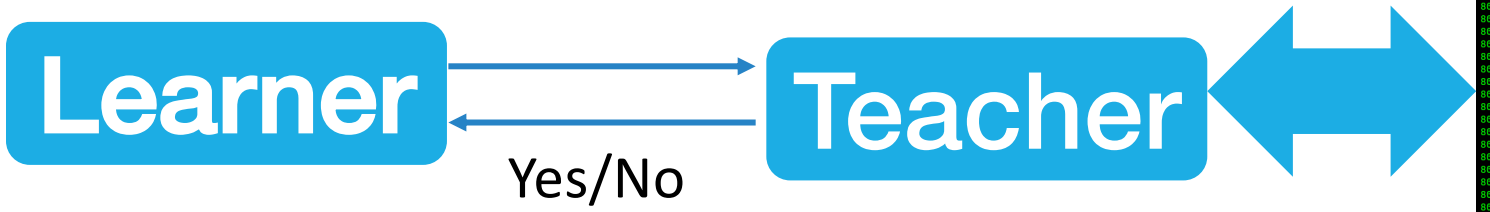




Answering Membership queries

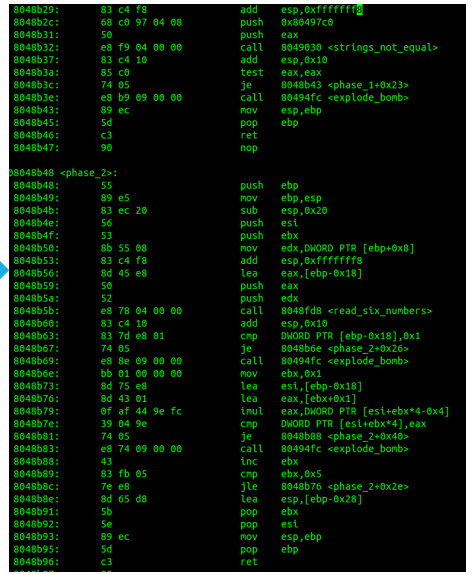
- Let's assume for now that we know the message types

Is this sequence of message types valid for the protocol?



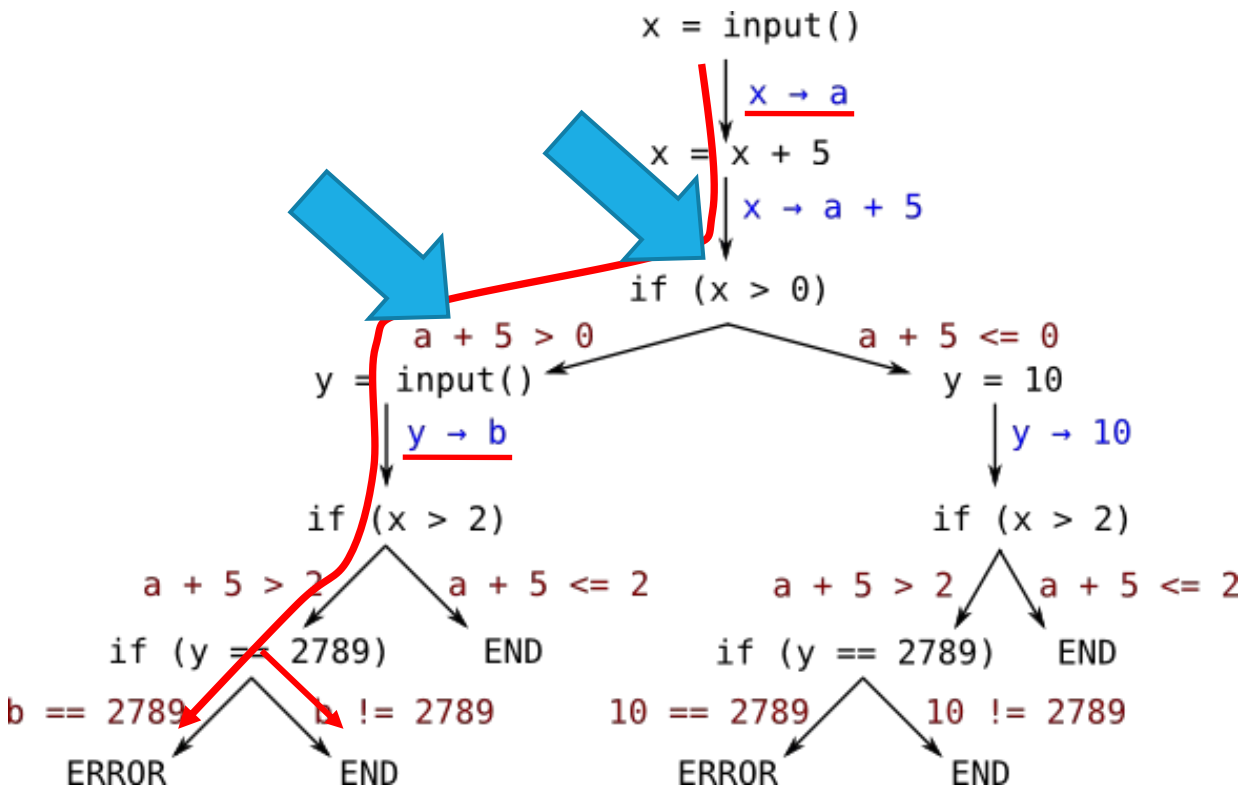
L* Algorithm

Symbolic Execution



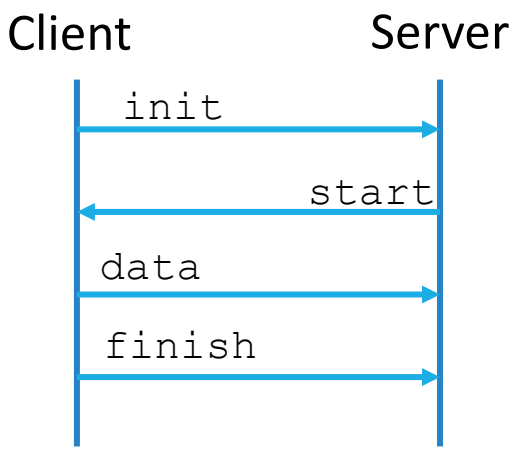
Symbolic Execution

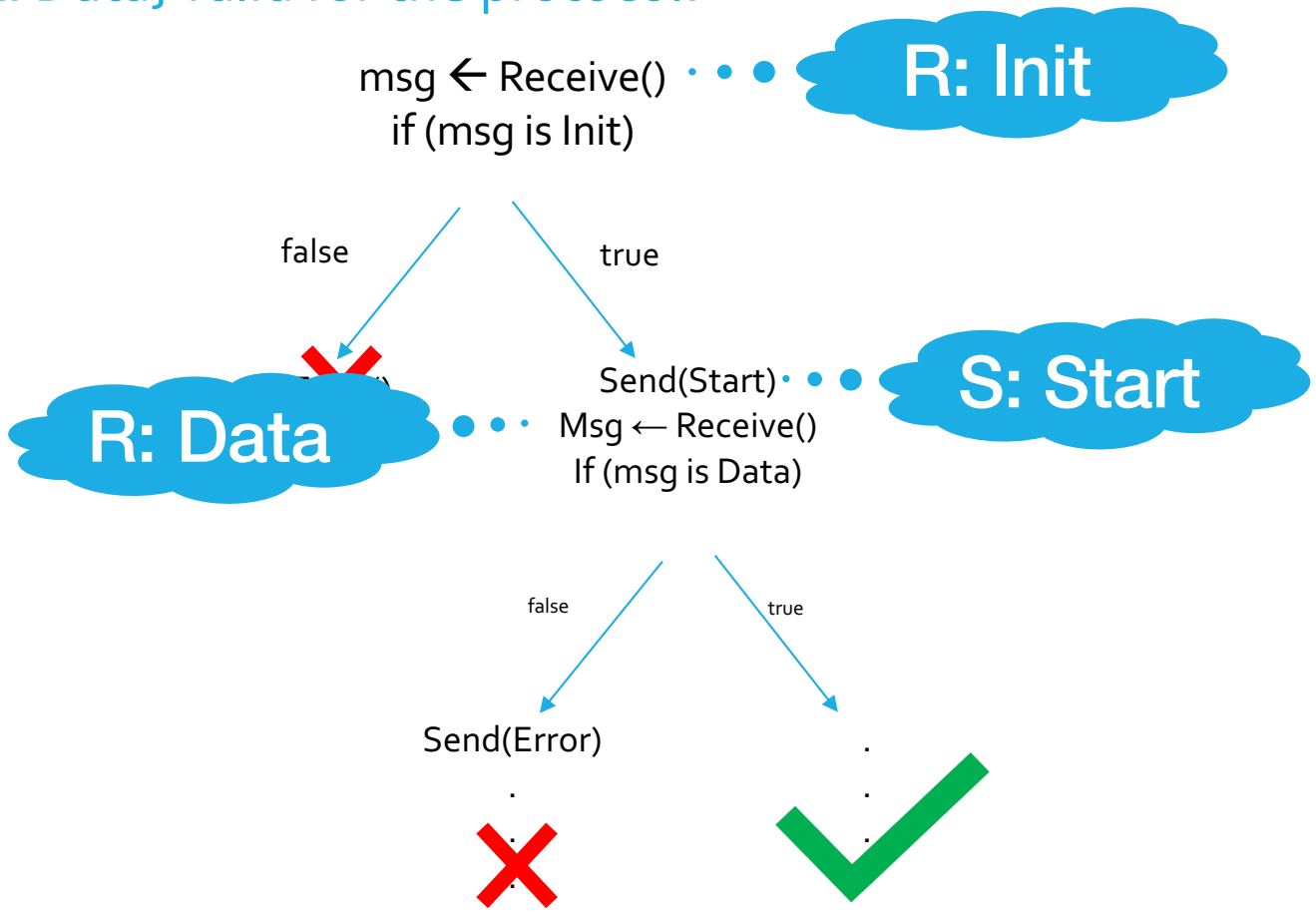
a > 3 ,b = 2789



Answering Membership queries

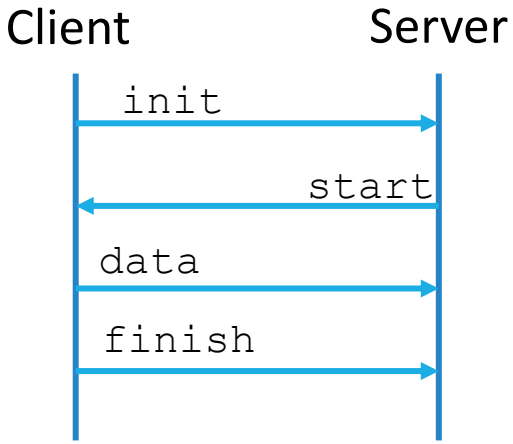
- Is {R: Init, S: Start, R: Data} valid for the protocol?

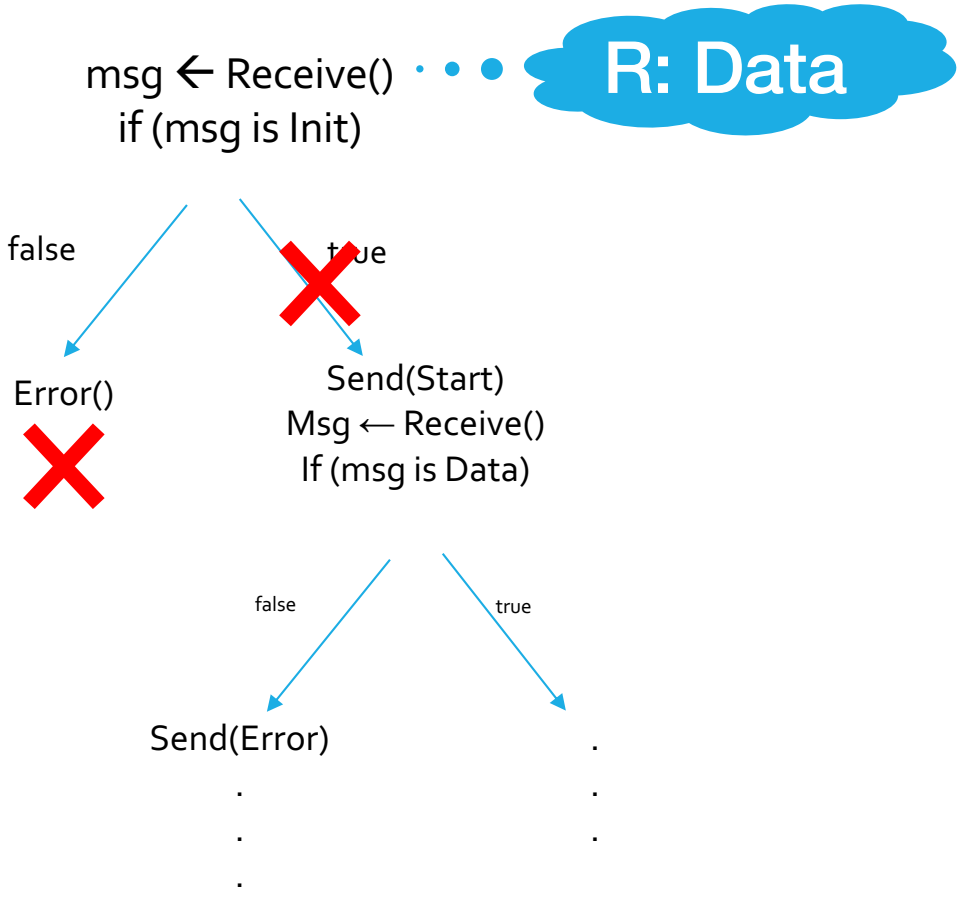




Answering Membership queries

- Is {R: Data} valid for the protocol?

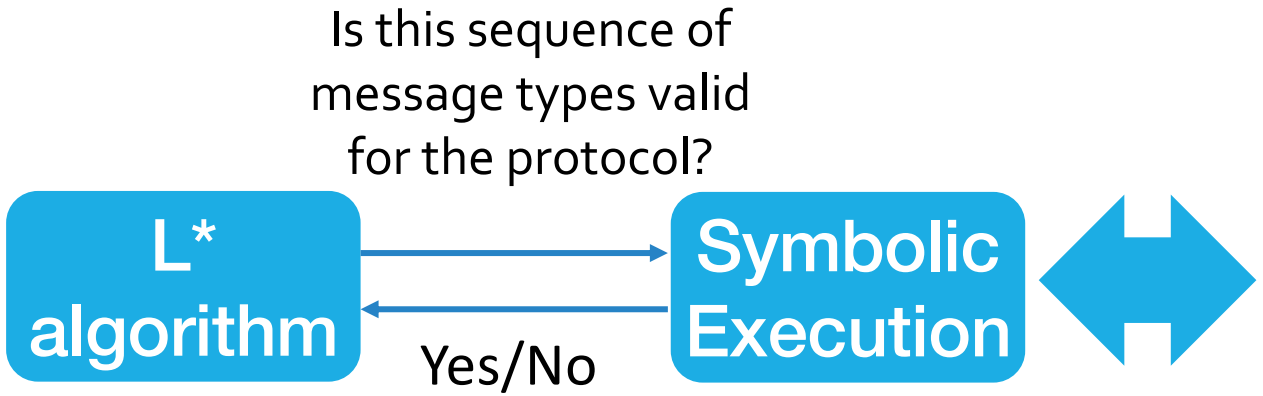


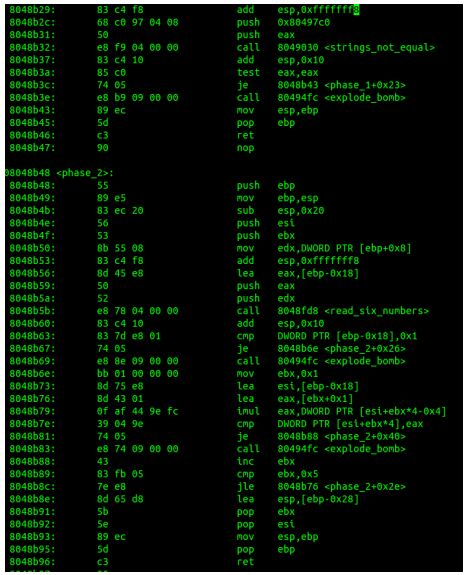


Discovering message types

- As said, we do not know in advance the protocol's message types.
- We update membership queries to discover it little by little.

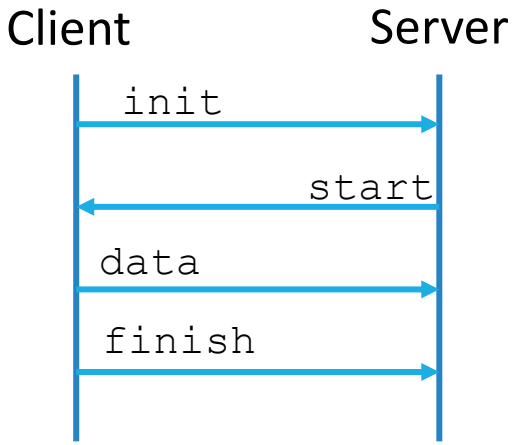
Extend L* to handle new message types

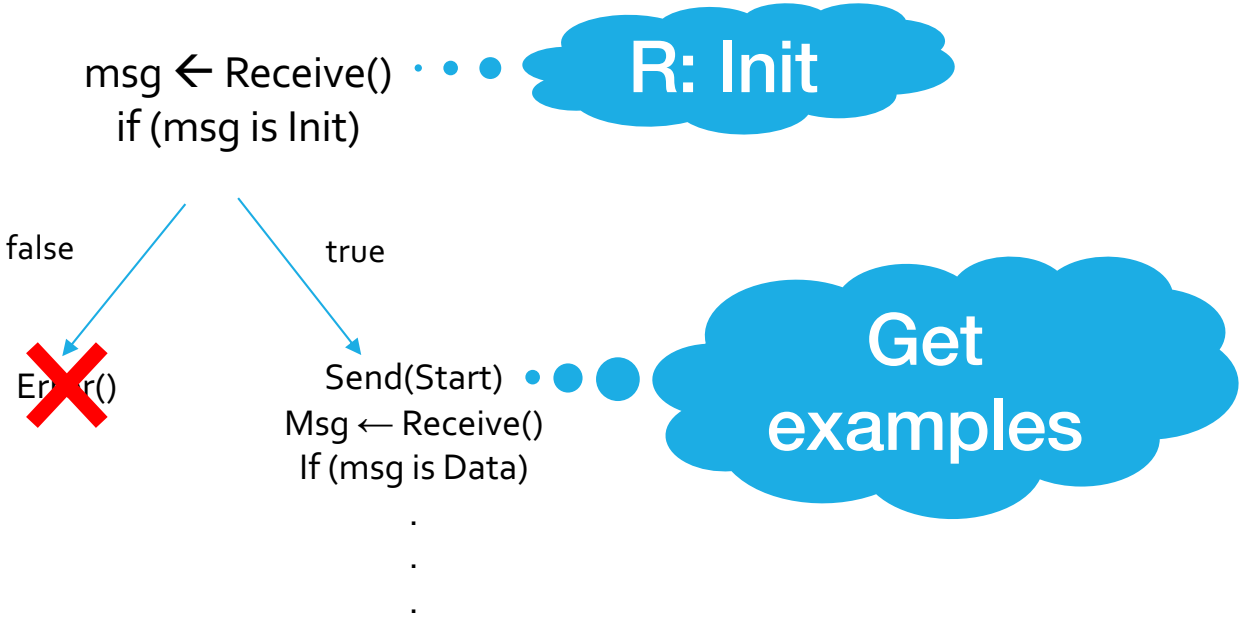




Probing for following message types

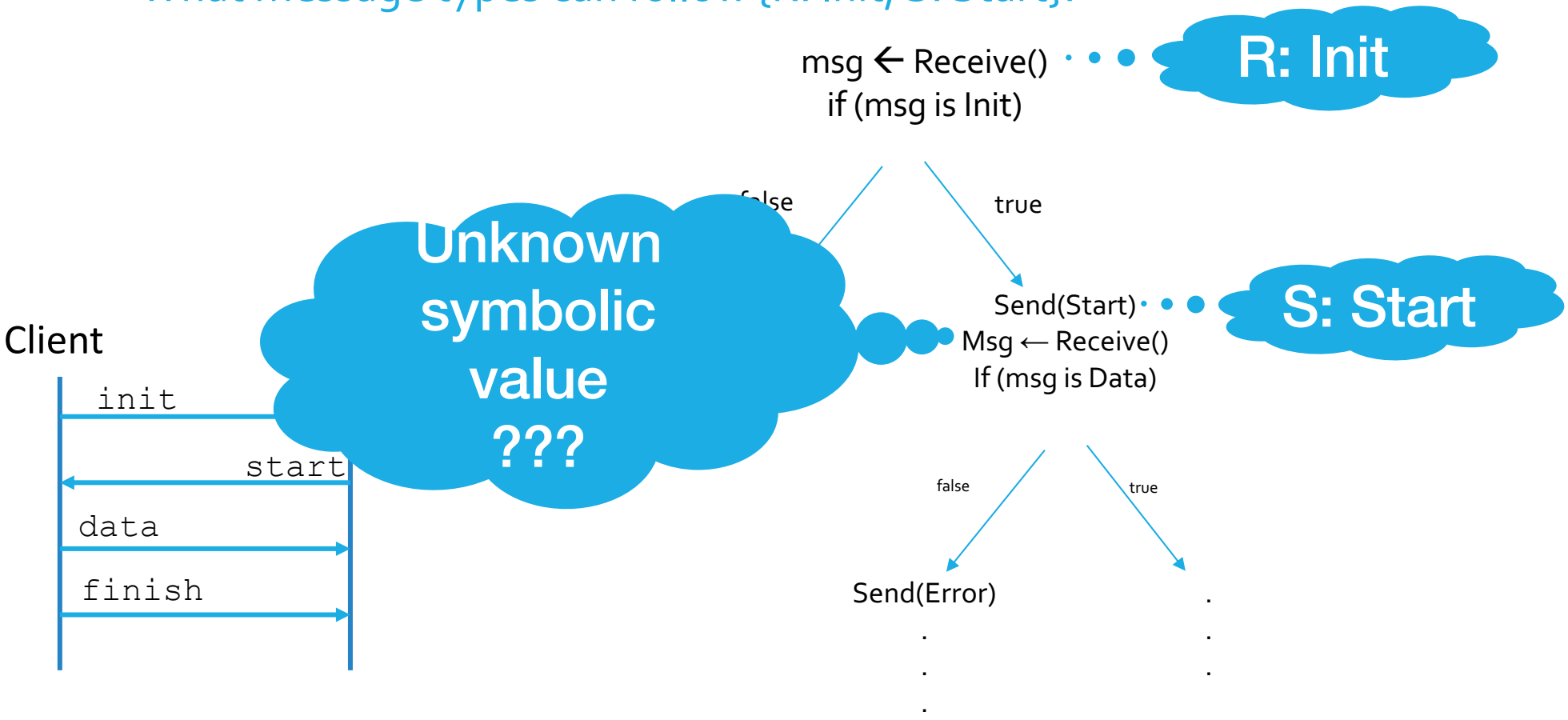
- What message types can follow {R: Init}?





Probing for following message types

- What message types can follow {R: Init, S: Start}?



Probing for following message types


```
msg ← receive()
if (msg begins with ‘data’) {
      // Constraint: msg begins with ‘Data’
} else {
      // I can’t parse this message, error
}
```

Concrete messages -> Message type

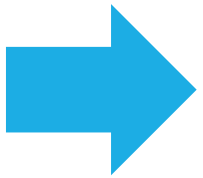
Example Messages

Find features of message type

RCPT TO: email1@blabla.com

RCPT TO: user2@lalala.bbb

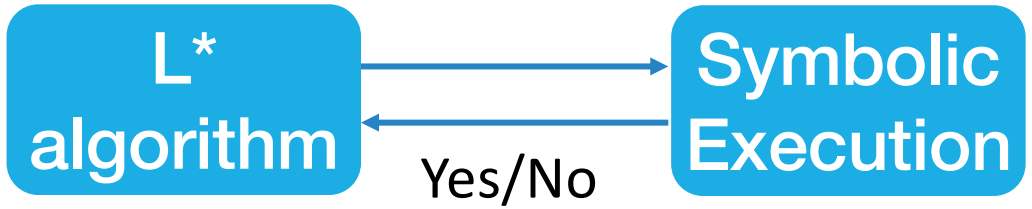
RCPT TO: person3@nana.ccc



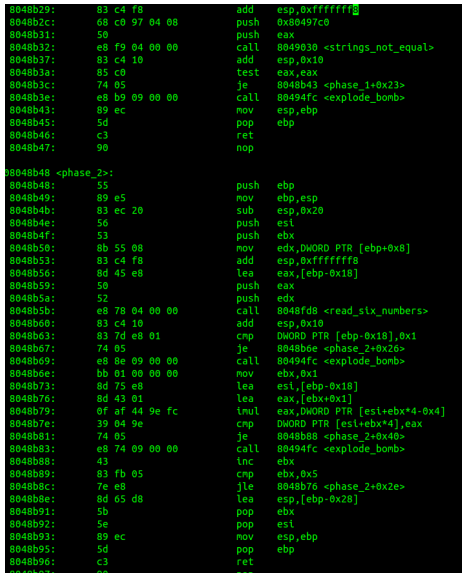
RCPT TO: ?????

Tying it all together

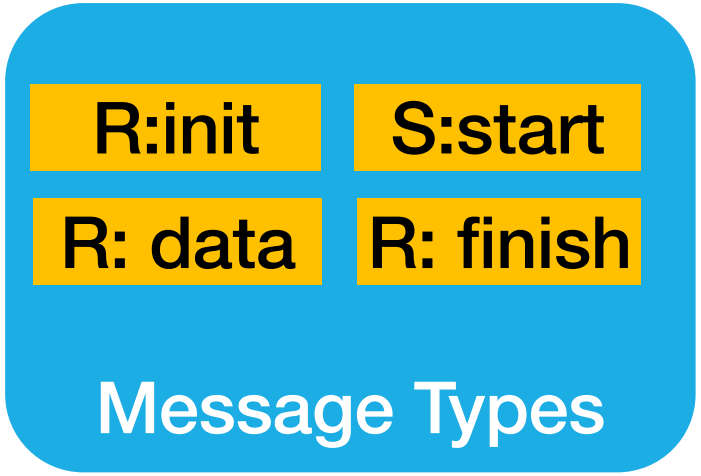
Is this sequence of message types valid for the protocol?

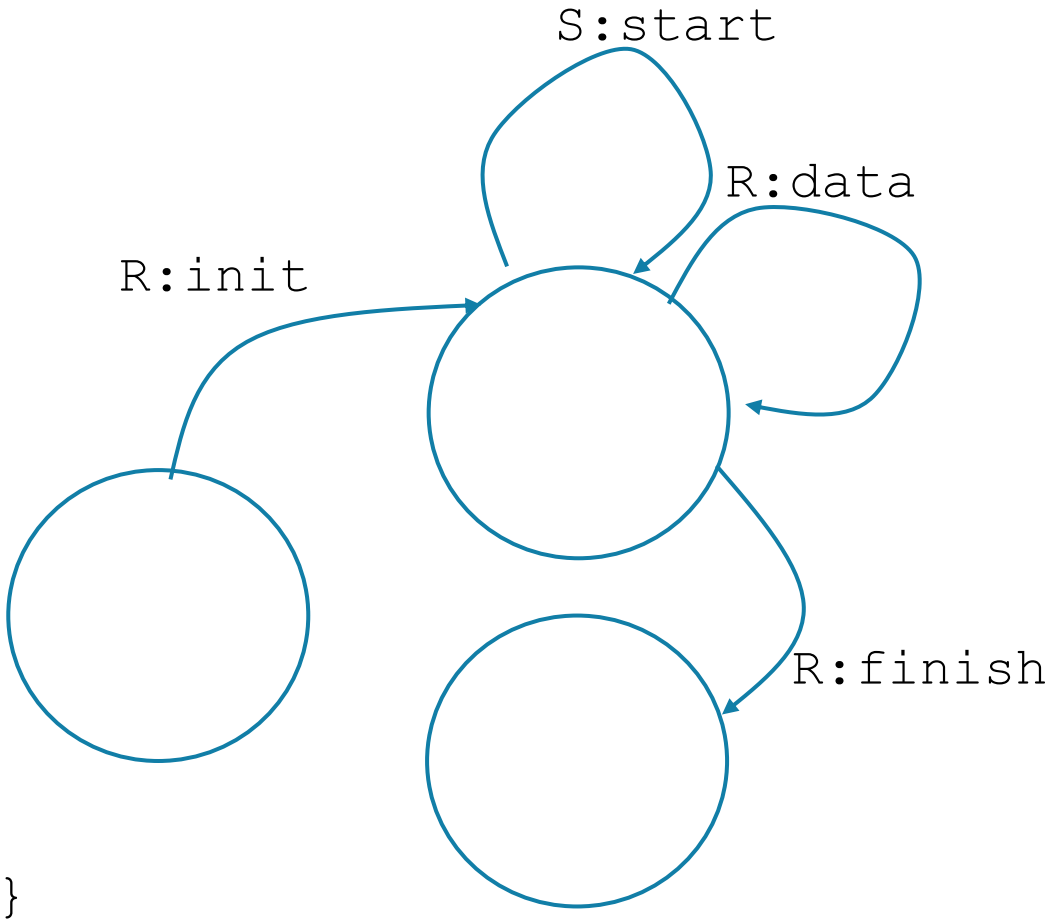


If yes, here are message types that can follow the sequence.

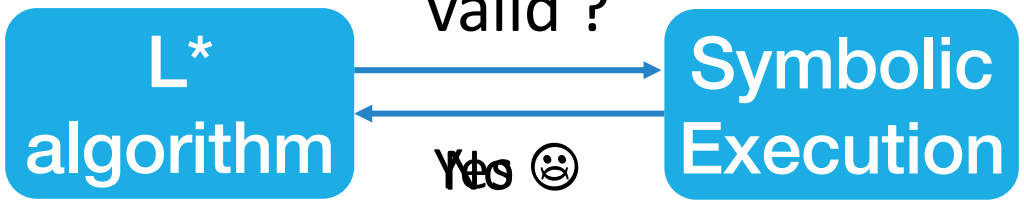


An illustrative example





M={R:init,S:start,R:data,R:finish}

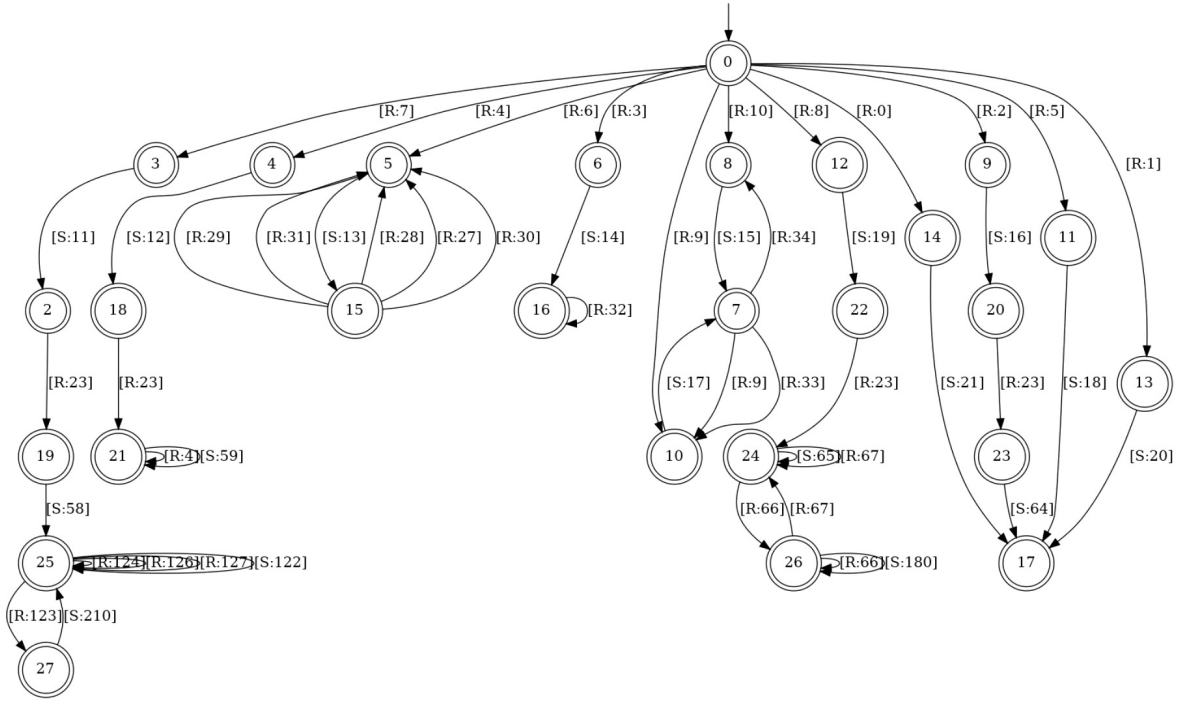


m_{next} = {S:start, R:data, R:finish}

Equivalence Query

- Approximated as in the original L* work, with a test suite
- Probing is also in use for the test suite
  - To discover missing message types

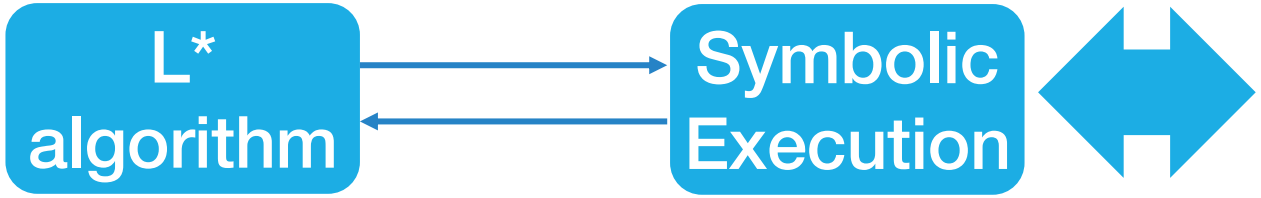
Example - method's output

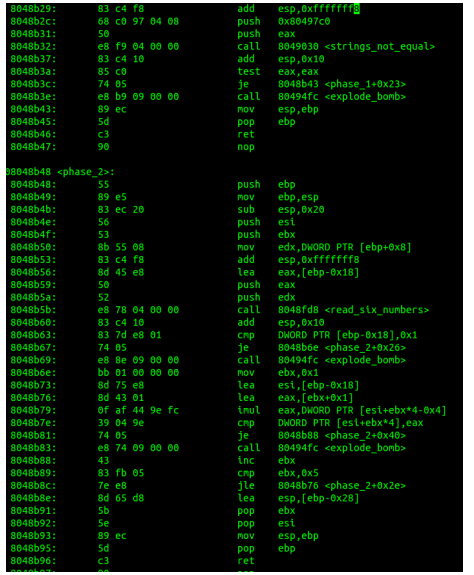


MSG ID	Name	Prefix	MSG ID	Name	Prefix
[R:0]	SERVER_EXIT	0xcd	[R:1]	CMD_BYE	0xcc
[R:2]	CMD_TALK	0x34	[R:3]	CMD_REGEDIT	0x33
[R:4]	CMD_AUDIO	0x22	[R:5]	CMD_SHELL	0x28
[R:6]	CMD_SERVICES	0x32	[R:7]	CMD_SCREEN_SPY	0x10
[R:8]	CMD_CAM	0x1a	[R:145]	CMD_SCREEN_BLOCK_INPUT	0x15
[R:10]	CMD_SYSTEM	0x23	[S:277]	TOKEN_CLIPBOARD_TEXT	0x76
[S:12]	TOKEN_BITMAPINFO	0x73	[S:13]	TOKEN_AUDIO_START	0x79
[S:14]	TOKEN_SERVERLIST	0x81	[R:140]	CMD_SCREEN_SET_CLIPBOARD	0x19
[S:16]	TOKEN_WSLIST	0x7e	[S:17]	TOKEN_TALK_START	0x84
[S:19]	TOKEN_SHELL_START	0x80	[S:20]	TOKEN_CAM_BITMAPINFO	0x77
[S:21]	CMD_BYE	0xcc	[R:32]	CMD_SVCCFG/START	0x83 0x01
[R:24]	CMD_NEXT	0x1e	[R:30]	CMD_SVCCFG/DEMAND_START	0x83 0x04
[R:29]	CMD_SERVICELIST	0x82	[R:31]	CMD_SVCCFG/AUTO	0x83 0x03
[S:22]	SERVER_EXIT	0xcd	[R:33]	CMD_SVCCFG/STOP	0x83 0x02
[R:34]	CMD_REG_FIND	0xc9	[R:36]	CMD_WINDOW_CLOSE	0x00
[R:37]	CMD_PSLIST	0x24	[S:67]	TOKEN_FIRSTSCREEN	0x74
[S:68]	TOKEN_AUDIO_DATA	0x7a	[S:74]	TOKEN_CAM_DIB	0x78 0x00
[S:73]	TOKEN_TALKCMPLT	0x85	[R:75]	CMD_CAM_ENABLECOMPRESS	0x1b
[S:112]	TOKEN_PSLIST	0x7d	[R:76]	CMD_CAM_DISABLECOMPRESS	0x1c
[S:137]	TOKEN_NEXTSCREEN	0x75	[R:138]	CMD_SCREEN_GET_CLIPBOARD	0x18
[S:15]	TOKEN_REGEDIT	0xc8	[R:144]	CMD_SCREEN_CONTROL	0x14
[R:9]	CMD_WSLIST	0x25	[S:199]	TOKEN_CAM_DIB/COMPRESS	0x78 0x01
[R:11]	CMD_LIST_DRIVE	0x01			

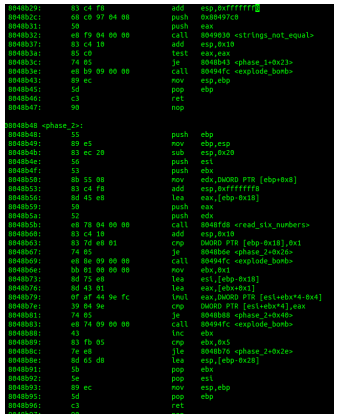
Caveats

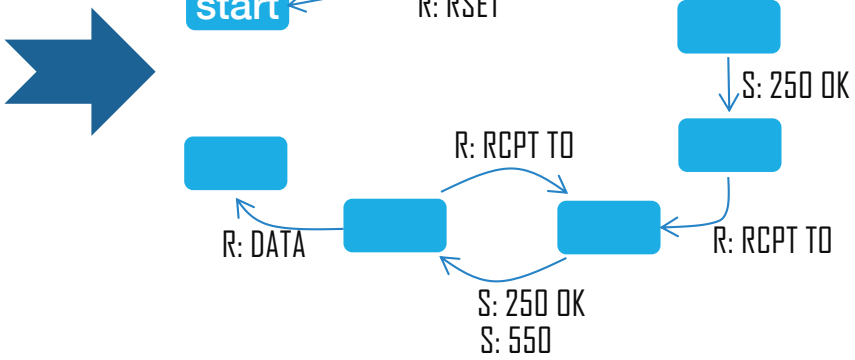
- PISE is as good or as bad as the symbolic tool it uses.
- Currently, PISE uses angr. 
  - Trouble supporting multiple threads.
  - Does not fully support windows API

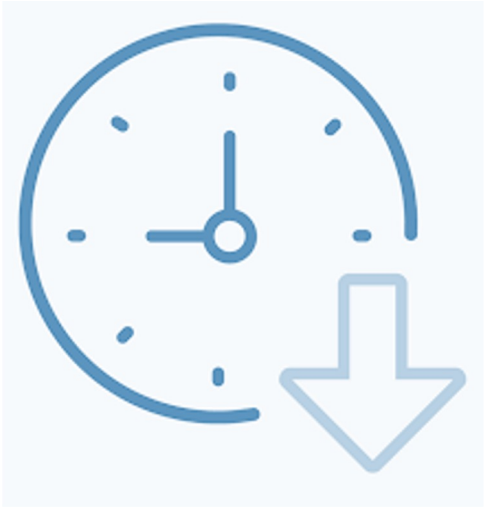




Summary




R: HELO
S: 250 OK
R: MAIL FROM
start
R: RSET
S: 250 OK
R: RCPT TO
R: DATA
R: RCPT TO
S: 250 OK
S: 550



QUESTIONS