Model-checking with the TimeLine formalism

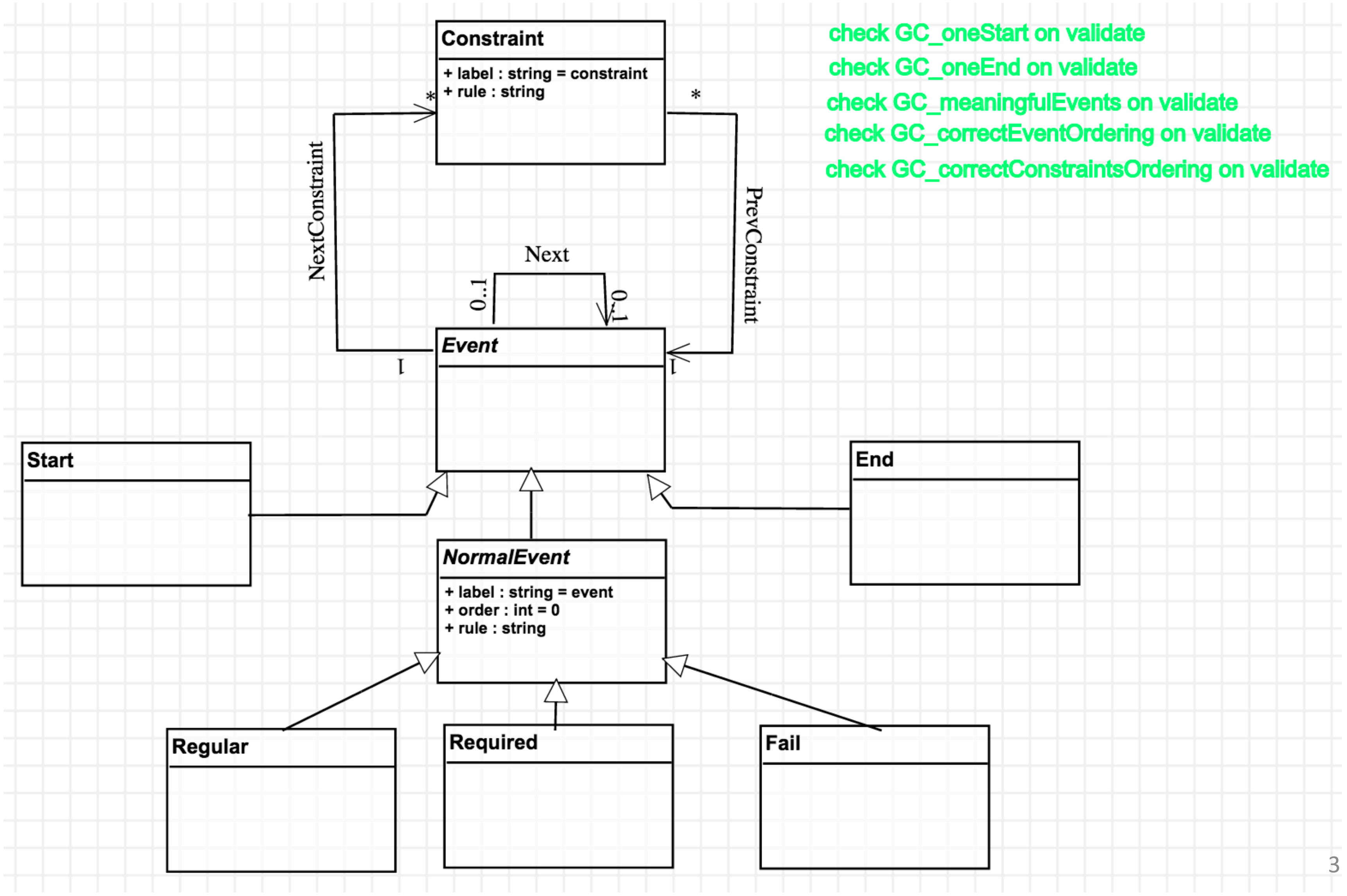
Andrea Zaccara
Andrea.Zaccara@student.uantwerpen.be

University of Antwerp

Overview

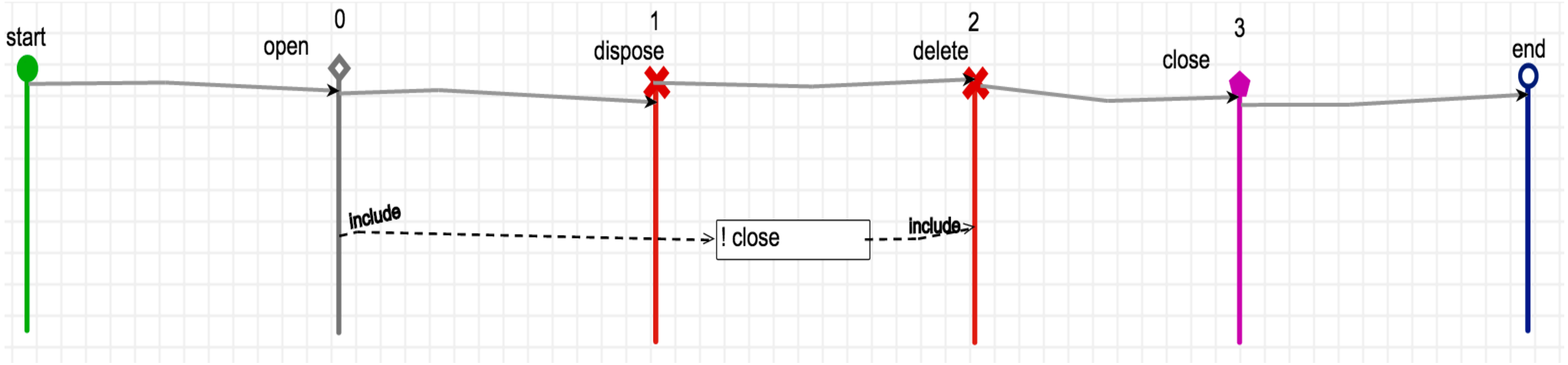
- The Timeline formalism in AToMPM
- Requirements for the chat protocol software
- Model transformation to FSA
- Code generation of a trace checker using EGL and Python

Abstract syntax in AToMPM


Constraint
+ label : string = constraint
+ rule : string
NextConstraint
PrevConstraint
*
*
1
1
Next
0..1
0..1
Event
Start
End
NormalEvent
+ label : string = event
+ order : int = 0
+ rule : string
Regular
Required
Fail
check GC_oneStart on validate
check GC_oneEnd on validate
check GC_meaningfulEvents on validate
check GC_correctEventOrdering on validate
check GC_correctConstraintsOrdering on validate

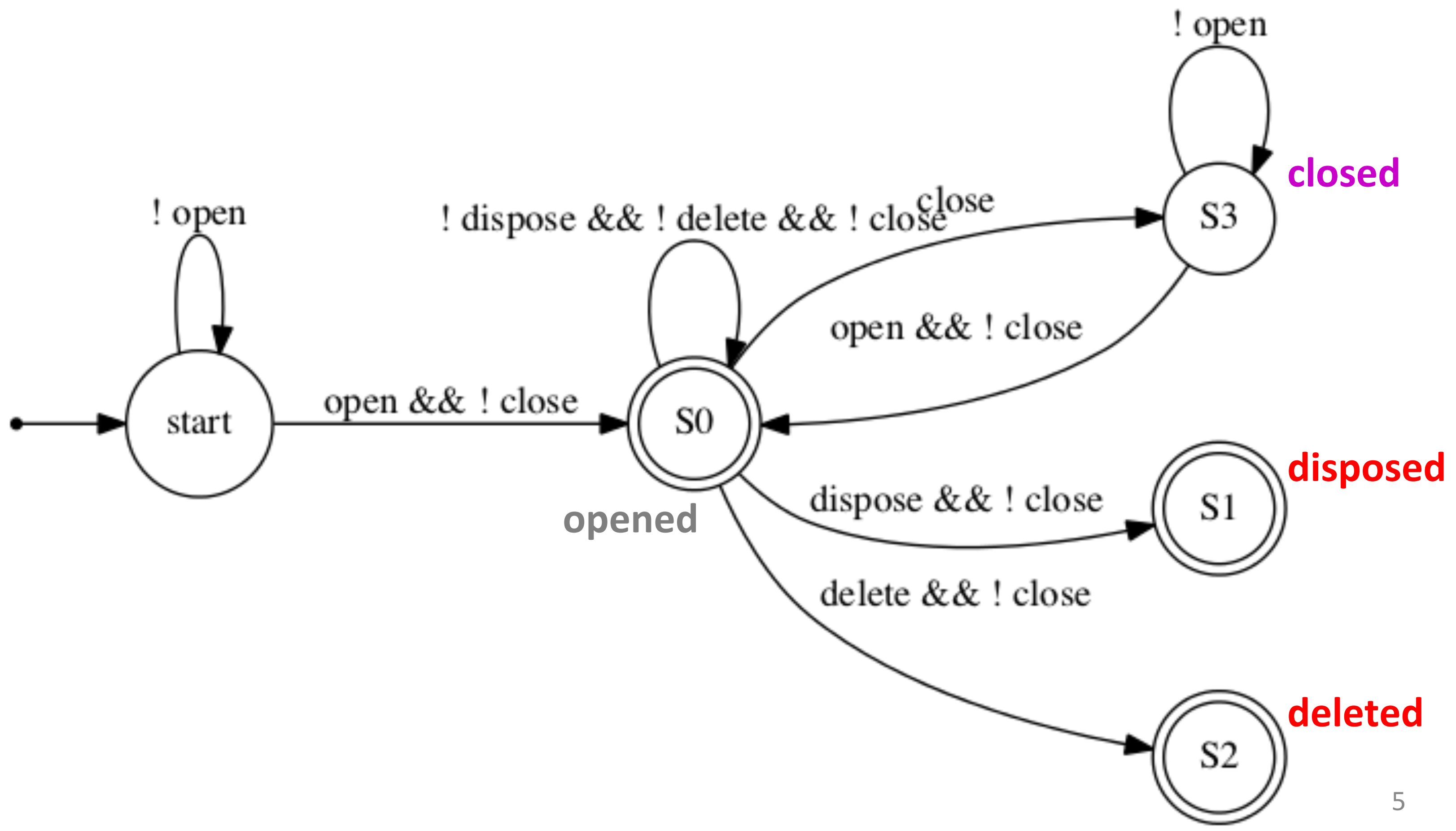
A simple requirement

- Regular event: After a file is opened,
- Fail event: it must not be disposed
- Fail event: or deleted, if it was not closed before
- Required event: and it must be closed before the end of the program.



Mapping to FSA

- Model transformations generate an augmented finite state automaton from TimeLine specification



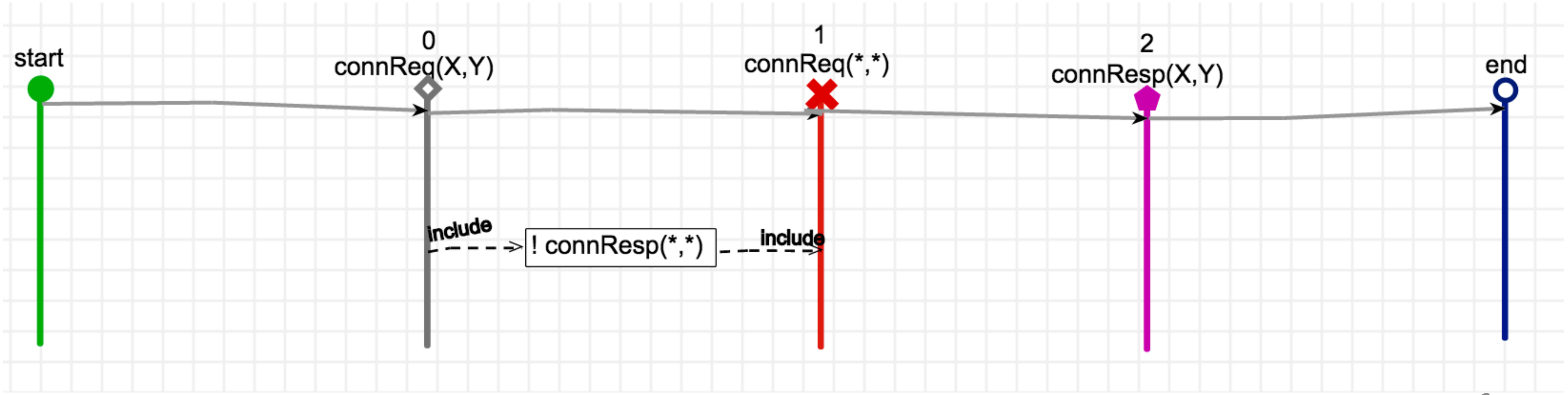
Chat protocol requirements (1)

- Requirement 2: On receiving a connection request, the chat room immediately makes a decision whether to accept the client or reject it.

connReq(X,Y) -> connection request from client Y to chat room X

connReq(*,*) -> any connection request

connResp(X,Y) -> connection response from chat room X to client Y

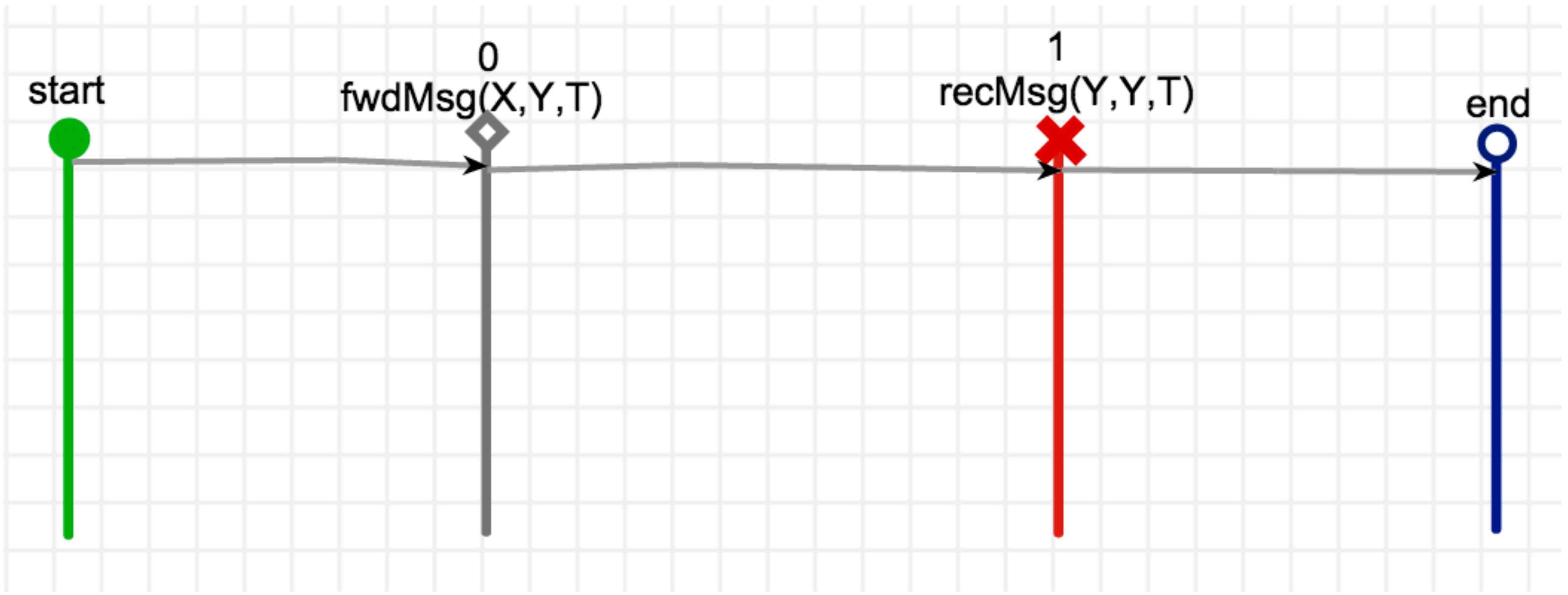


Chat protocol requirements (2)

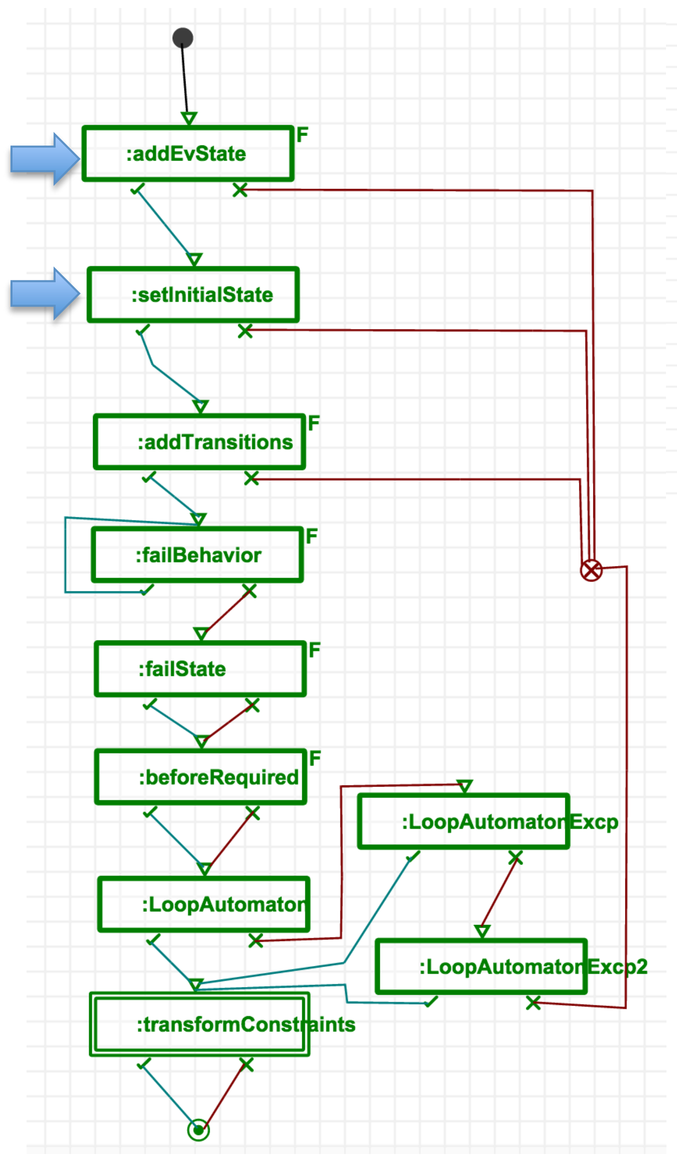
- Requirement 7: The sender cannot receive its own message after it sends it.

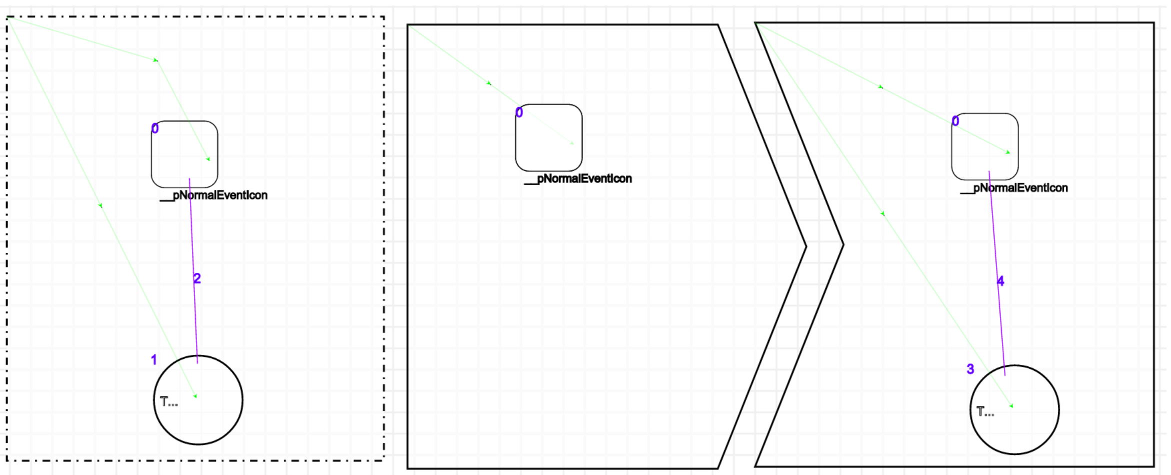
fwdMsg(X,Y,T) -> message T from client Y received by chat room X

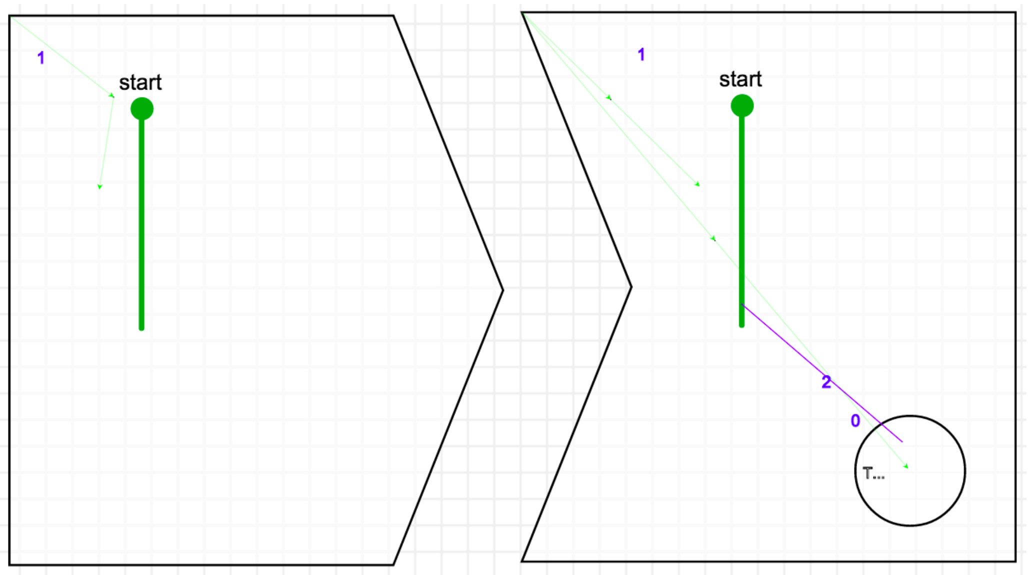
recMsg(Y,Y,T) -> message T from client Y received by client Y



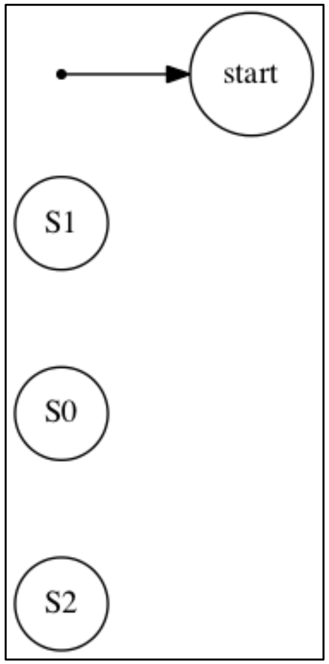
Model transformation to FSA (1)



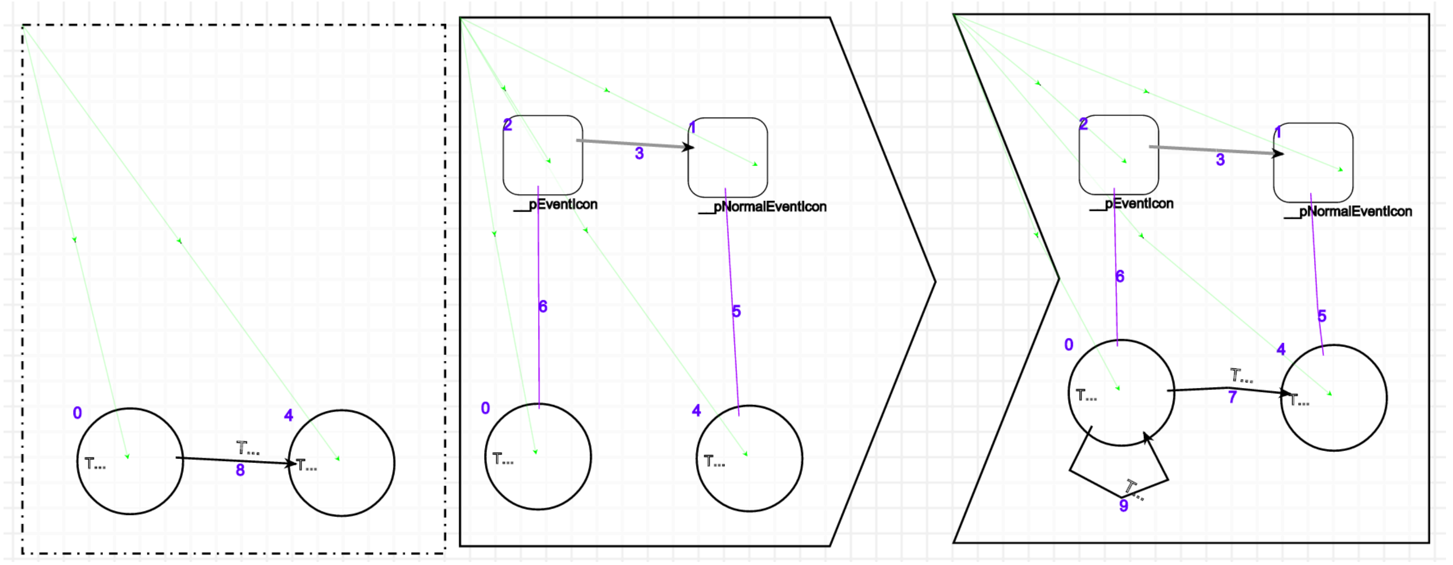


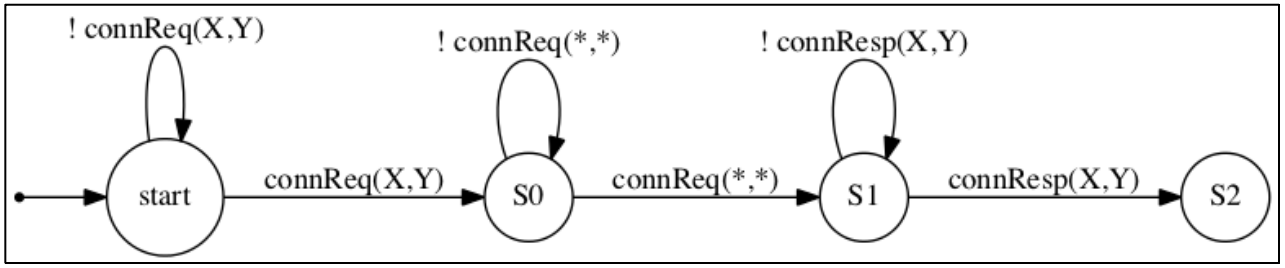


Model transformation to FSA (2)

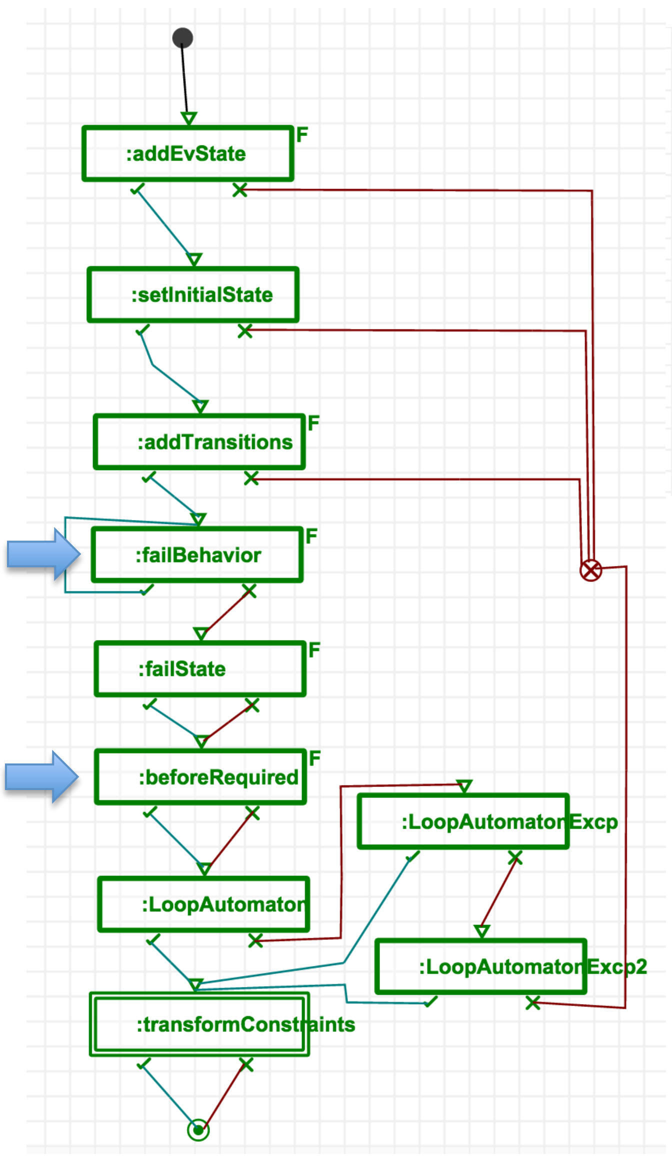


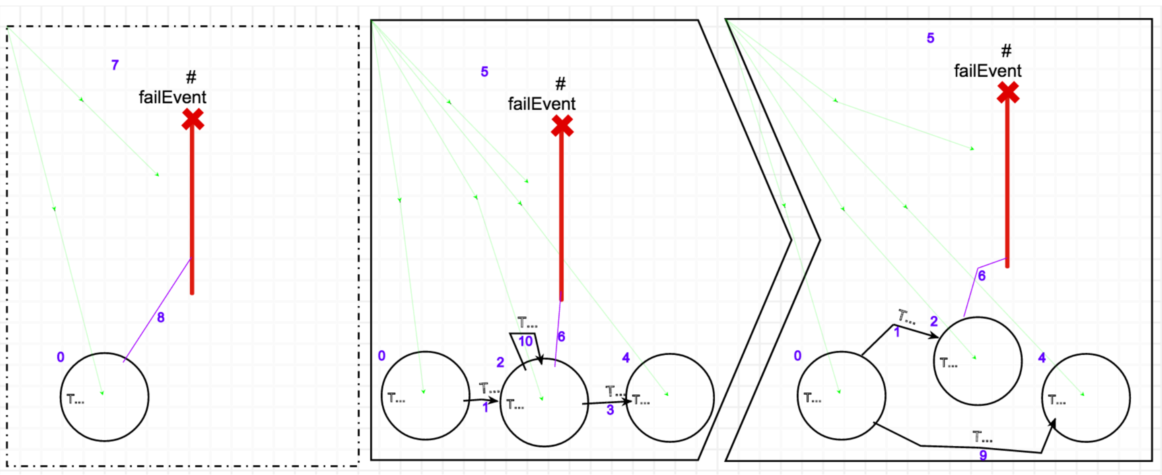
Add transitions

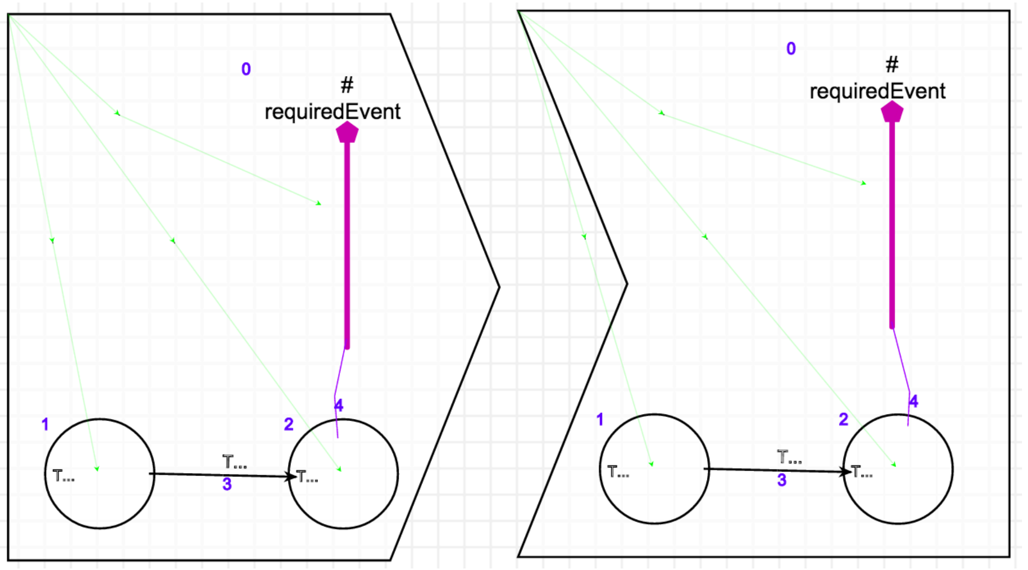




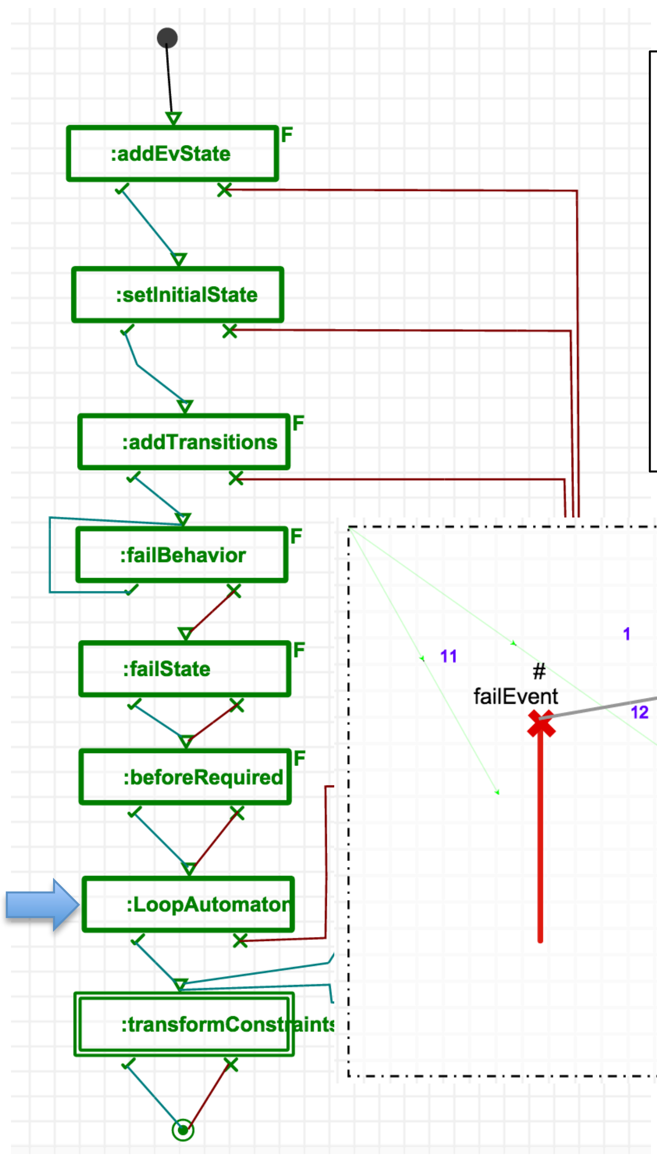
Model transformation to FSA (3)

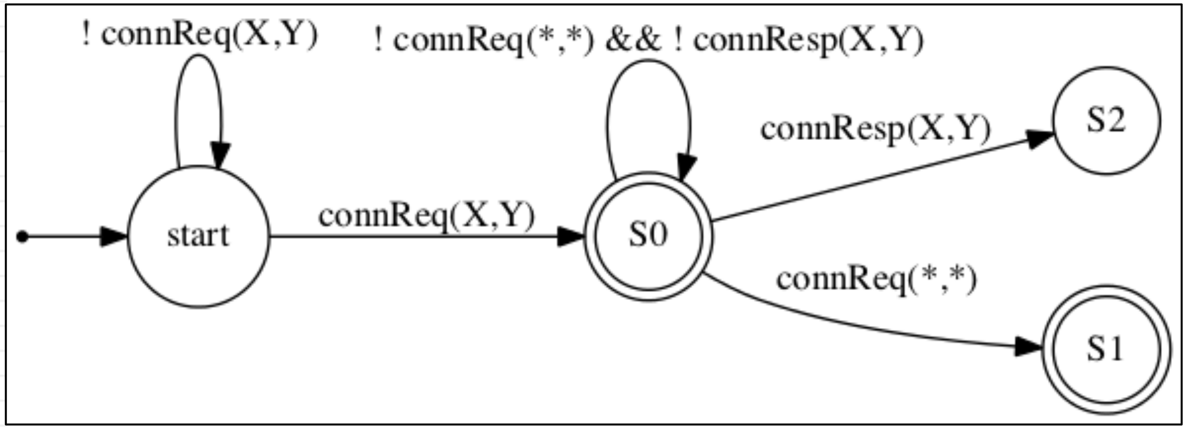


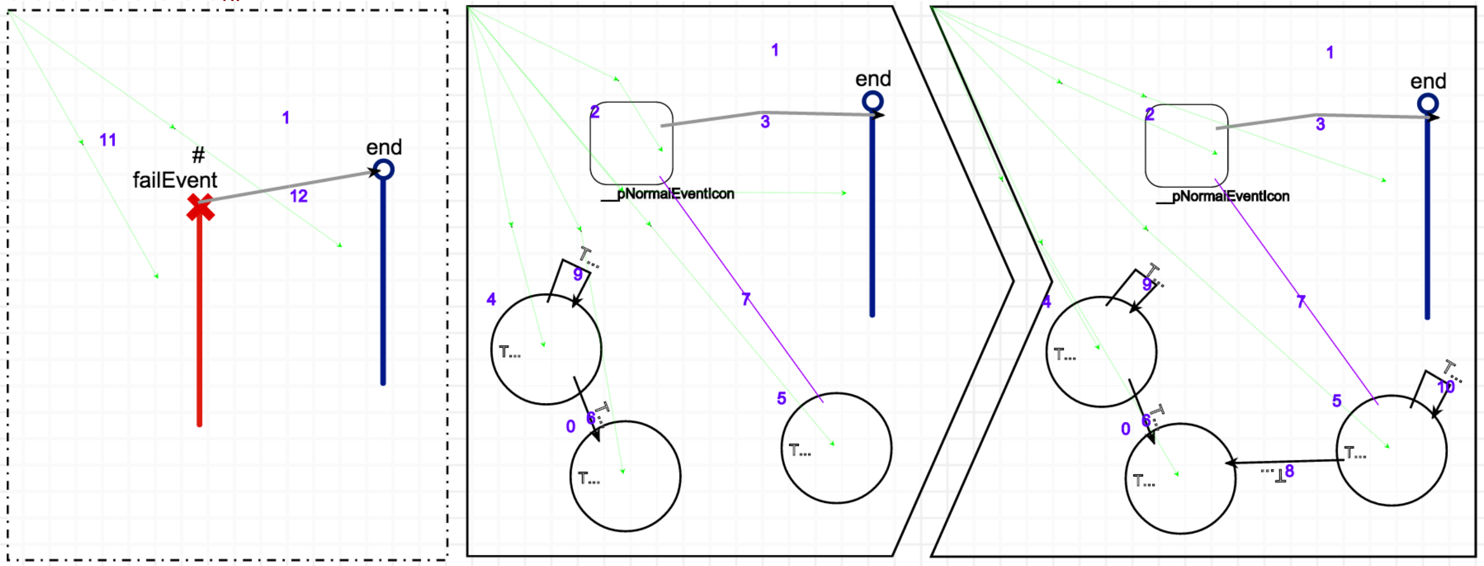




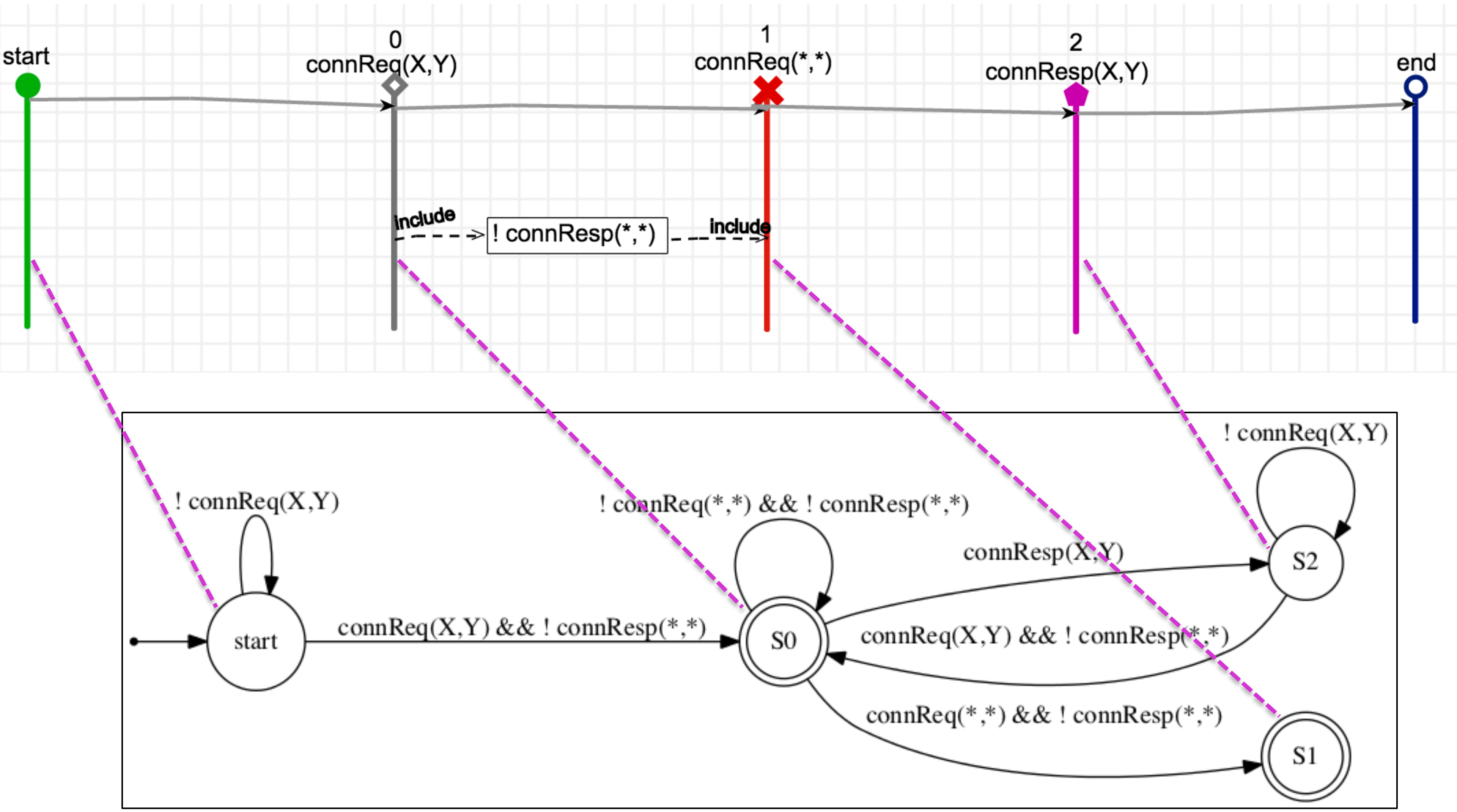
Model transformation to FSA (4)



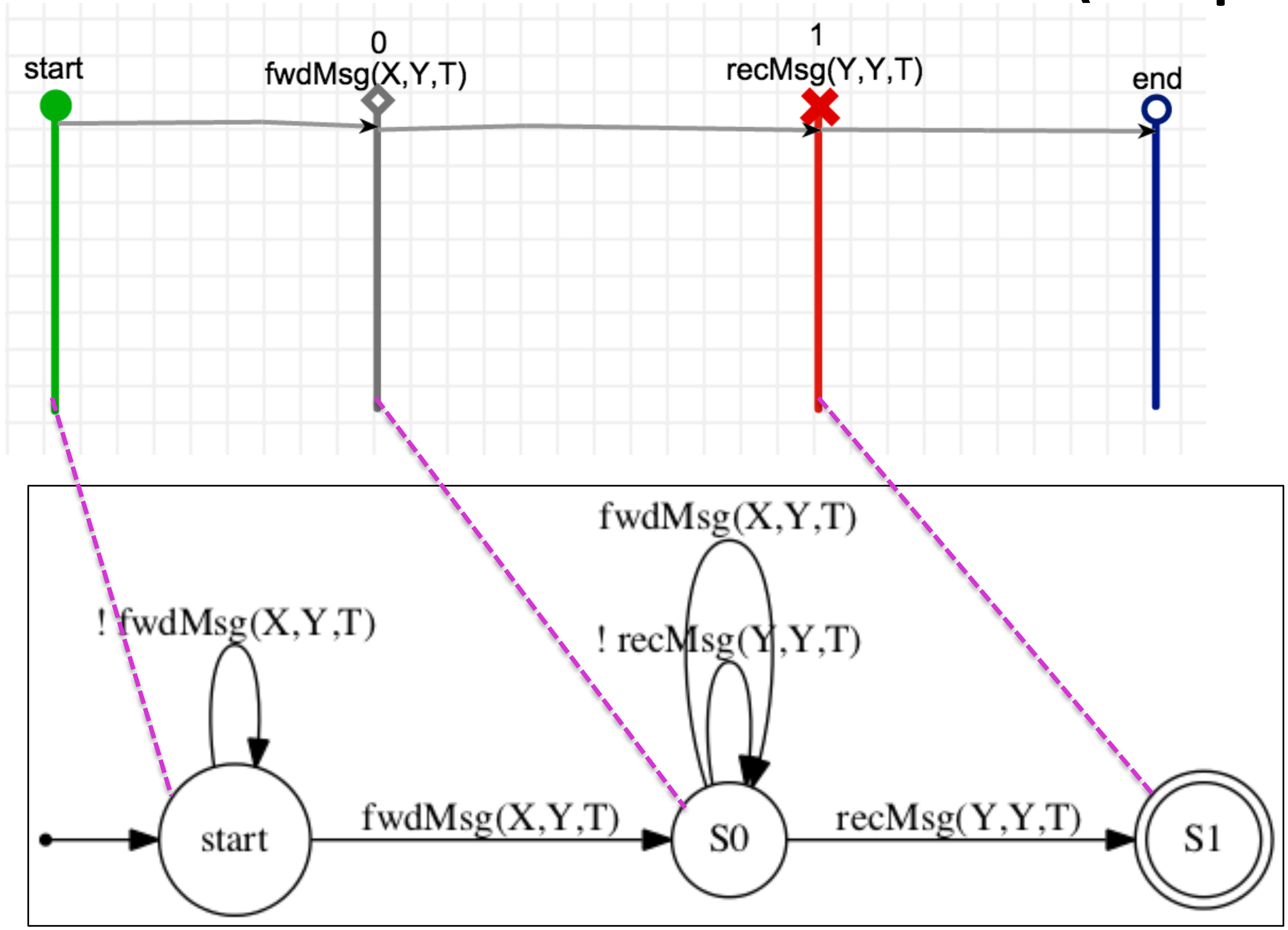




Result of the transformation (req 2)



Result of the transformation (req 7)

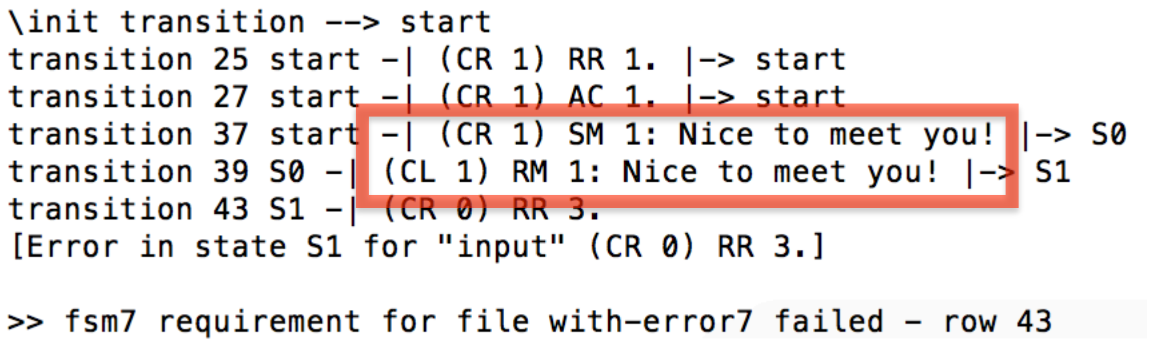


Rule definition

- The rules for matching an event use a basic grammar for simple regular expression
  - '! '   for negation
  - ‘%..%’ stores values (e.g., %X% -> save value in X)
  - ‘?..?’ checks stored value (e.g., ?X?)
  - ‘?*?’ matches any string sequence
  - ‘[str1|str2]’ matches for both string str1 and str2

Code generation and checking

- Done in two phases:
  1. Generation of a data structure in Python for the automaton, using the metaDepth exported model and a script in EGL
  2. Generation of a pre-processed parser using a custom Python script



```
\init transition --> start
transition 25 start -| (CR 1) RR 1. |-> start
transition 27 start -| (CR 1) AC 1. |-> start
transition 37 start -| (CR 1) SM 1: Nice to meet you! |-> S0
transition 39 S0 -| (CL 1) RM 1: Nice to meet you! |-> S1
transition 43 S1 -| (CR 0) RR 3.
[Error in state S1 for "input" (CR 0) RR 3.]

>> fsm7 requirement for file with-error7 failed - row 43
```

Conclusions and Future works

- The TimeLine meta-model in AToMPM enables faster and easier definition of requirements
- Can already give useful information on software validation
- Regular expression support can be expanded with more functionality
- Rules definition can be made more modular and easier to define

Questions?