Generating Models of Black-Box Components using Abstraction

Bengt Jonsson

Uppsala University

Joint work with

Fides Aarts[1], Falk Howar[2], Bernhard Steffen[2], Johan Uijen[1]

1: Radboud University, Nijmegen

2: TU Dortmund

Modeling in (Component-Based) Design and Verification

Models are cornerstone of system development

- Model Driven Development
- Model Based Testing
  - tests generated as (abstract) executions
  - Tools: Qtronic,TGV, GOTCHA, TorX, …
- Model Checking
  - Models of software, and of environment

Modeling Gap

- **Typically, models are not available**
- **"Modeling SUT** [system under test] is among **biggest obstacles** in Model Based Testing" [Hartmanis]

What to do if there is no model?

(the norm in practice)

How to support generation of models?

- Model Behavior of existing implementation
  - By observations gained during extensive testing
    (Source code analysis: sometimes not feasible)
- Potential Applications:
  - Regression testing
  - Migrating from manual to model-based testing
  - Modeling environment of SUT, libraries
  - Verifying properties (Black Box Checking [Peled, Yannakakis])
- Other use of such techniques
  - For requirements capture
  - "Programming by Scenarios" (PlayIn-PlayOut) [Harel etal]

Model Generation by Inference

General Scheme:

- Given a set of instances:
  - Traces, Message charts, System states
- Produce a "simplest" specification which is consistent with these instances.

Applications:

- Behavioral Models from Behaviors
- Requirement Specifications from Scenarios
- Invariants from Sets of reachable/unreachable states

Outline

- Principles of Regular inference (automata learning)
- Extension to include data manipulation in protocols
- Abstraction techniques
- Experiments on communication protocols
- Further thoughts and future work

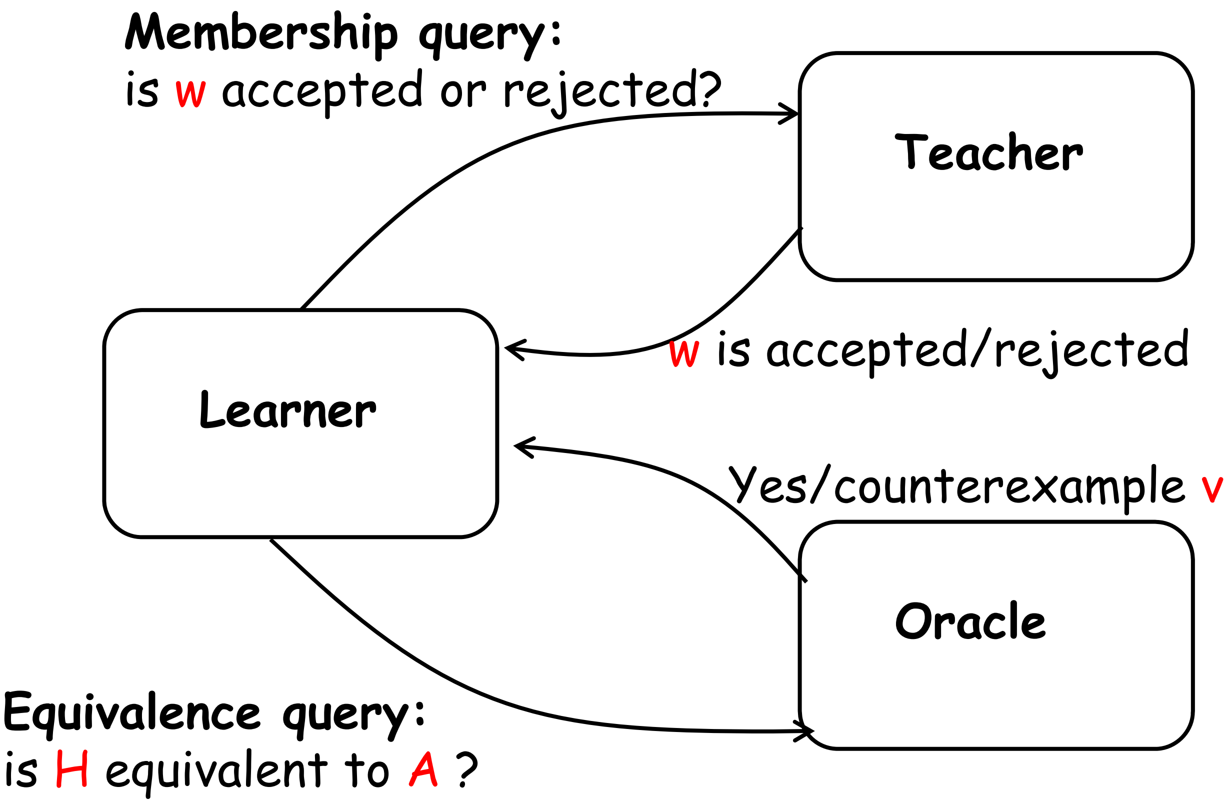
Regular Inference (Automata Learning)

- Construct Regular Language (as a DFA) from sample of accepted and rejected words.
- Developed since 1970's, . Applications in, e.g.,
  - Natural Language Processing,
  - Testing/Verification (more recently),

Regular Inference (Automata Learning)

- Construct Regular Language (as a DFA) from sample of accepted and rejected words.
- Developed since 1970's. Applications in, e.g.,
  - Natural Language Processing,
  - Testing/Verification (more recently),
- **off-line** inference:
  - sample of words fixed *a priori.*
  - Problem is to construct "good enough" DFA.
  - Constructing minimal DFA is NP-complete [Gold 78]
- **on-line** inference:
  - words chosen dynamically, on the basis of previous information.
  - Easier to construct "good enough"/minimal DFA by extending sample with "interesting" words
  - Most well known algorithm: L* [Angluin 87]

Setup for inferring A



Mealy Machines

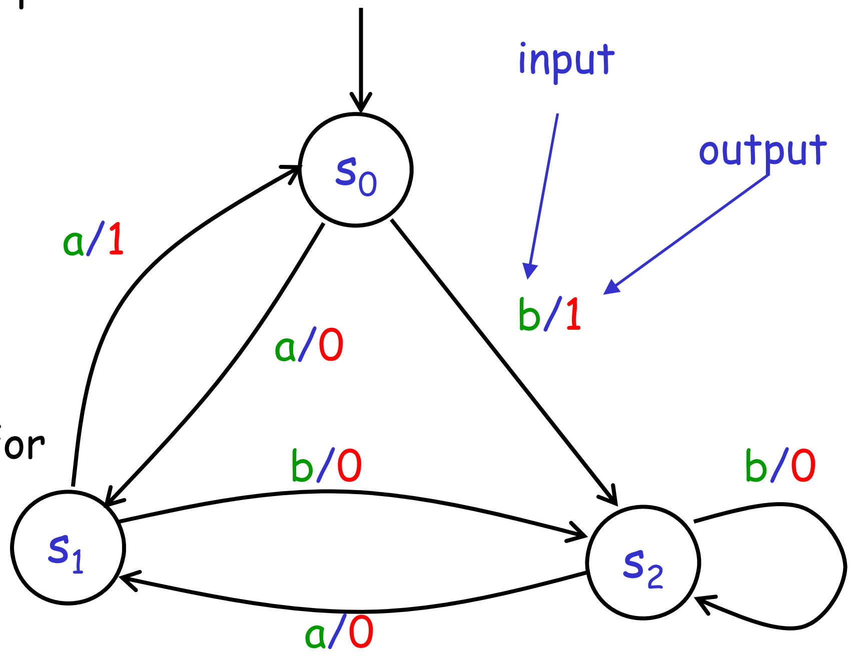
- Finite State Machines w. input & output

I	input symbols
O	output symbols
S	states
$\delta: S \times I \rightarrow S$	transition function
$\lambda: S \times I \rightarrow O$	output function

- Often used for protocol modeling, for protocol testing techniques,

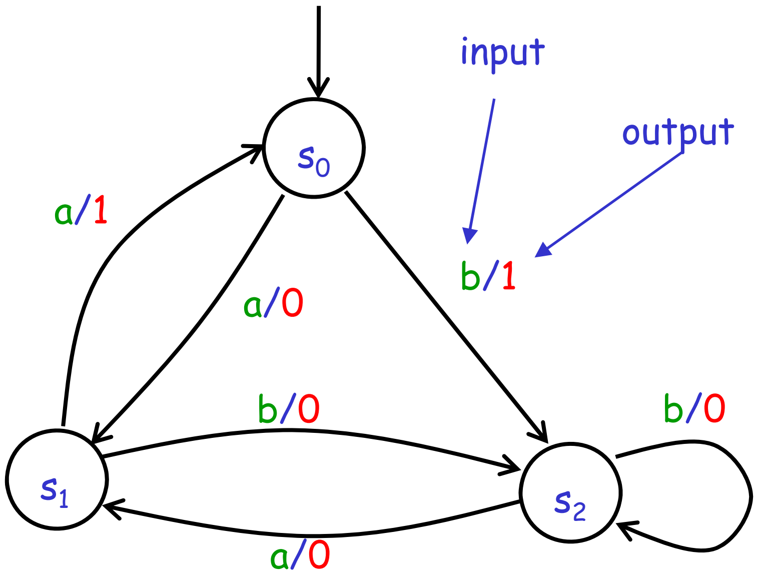
Assumptions:

- **Deterministic**
- **Completely specified**



Regular inference

- System viewed as Black box
- Membership query:
  - Supply input, observe output
- Record and Collect traces
- Construct protocol model

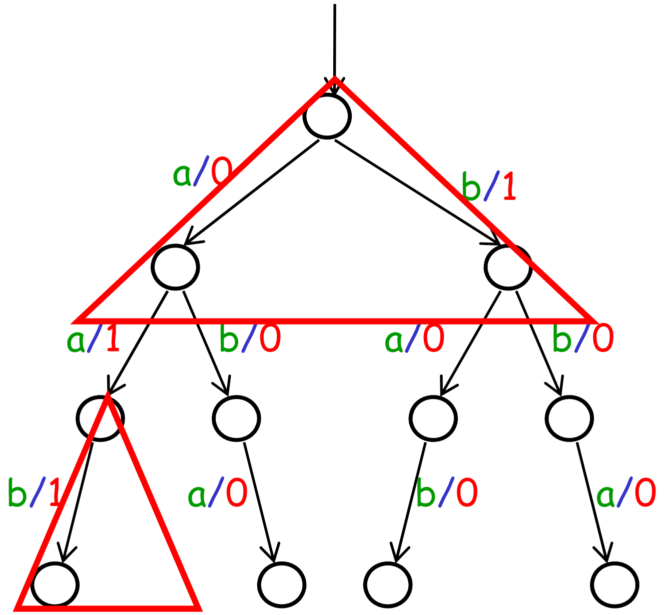


a/0 a/1 b/1 b/0 a/0
a/0 b/0 a/0 b/0
b/1 a/0 b/0 b/0 a/0
b/1 b/0 a/0 b/0

Contructing Model from Traces

a/0 a/1 b/1
a/0 b/0 a/0
b/1 a/0 b/0
b/1 b/0 a/0

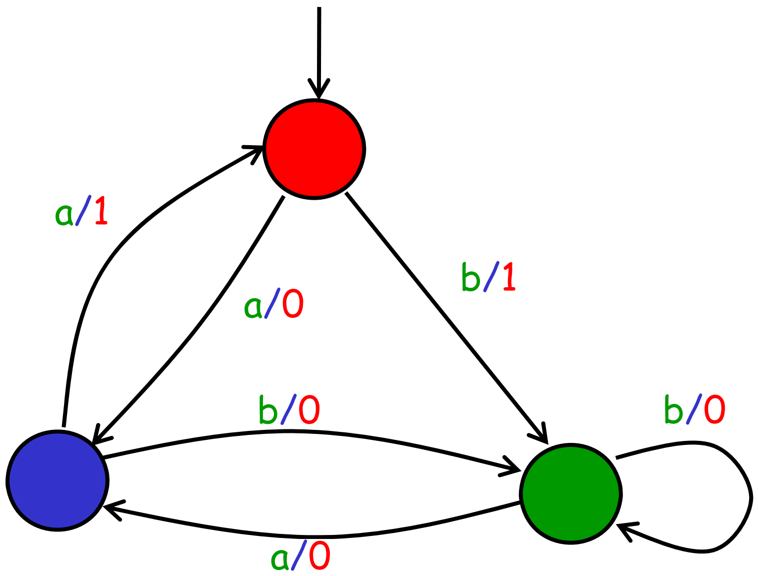
- Organize traces into tree
- Identify "equivalent" nodes

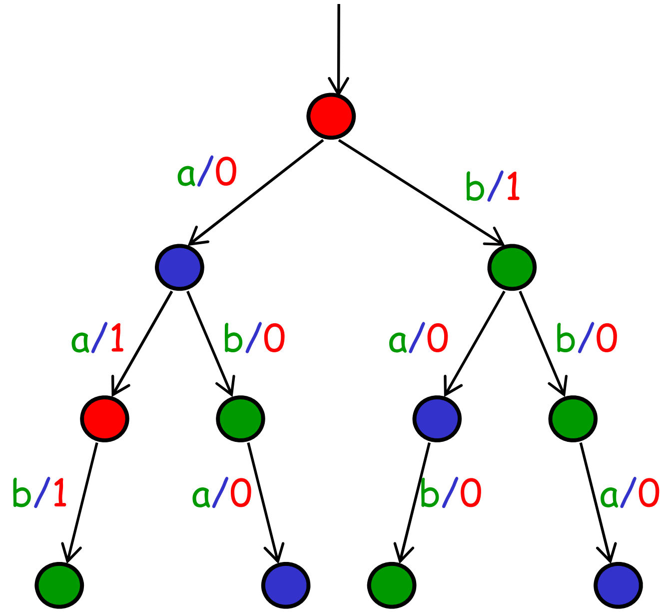


Contructing Model from Traces

a/0 a/1 b/1
a/0 b/0 a/0
b/1 a/0 b/0
b/1 b/0 a/0

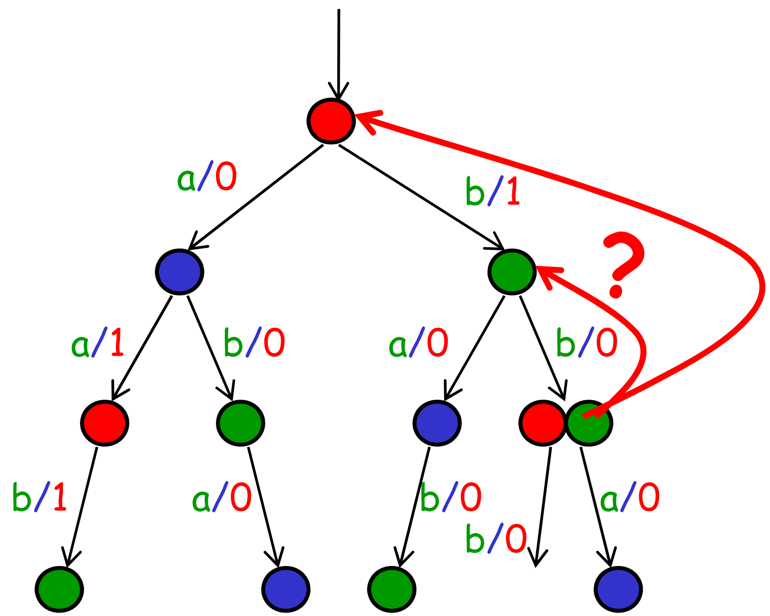
- Organize traces into tree
- Identify "equivalent" nodes
- Merge nodes
- Form automaton





Which model?

- Many ways to identify nodes
- Finding "smallest" model is NP-complete [Gold78]
- Allow to ask for more information to get more traces

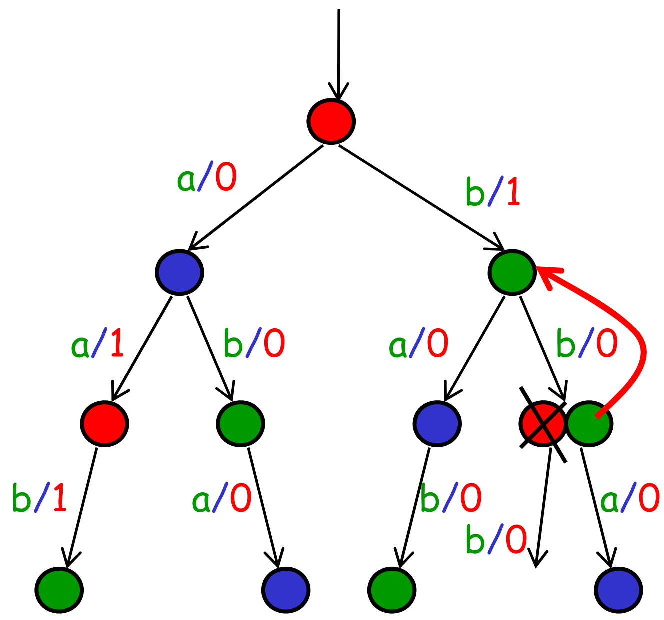


Which model?

•Many ways to identify nodes

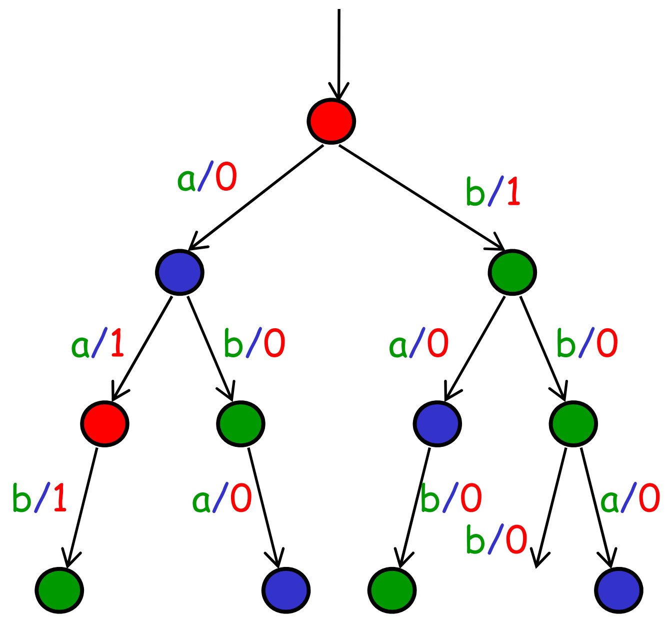
•Finding "smallest" model is NP-complete [Gold74]

•Allow to ask for more information to get more traces



Which model?

- Many ways to identify nodes
- Finding "smallest" model is NP-complete [Gold74]
- Allow to ask for more information to get more traces
- Resolves ambiguities
- Constructing "smallest" model becomes simple [Angluin 87]



Algorithms for On-Line Inference

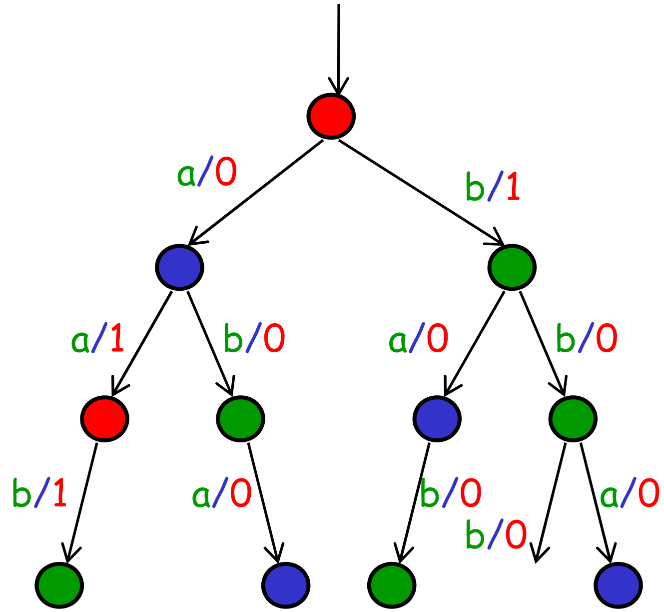
- Exist several variations, most well-known: L* [Angluin 87]
- Traces divided into prefix-suffix,
- Organized into Observation Table

Suffixes

Prefixes

	a	b
ε	0	1
a	1	0
b	0	0
aa		1
ab	0	
ba		0
bb	0	0

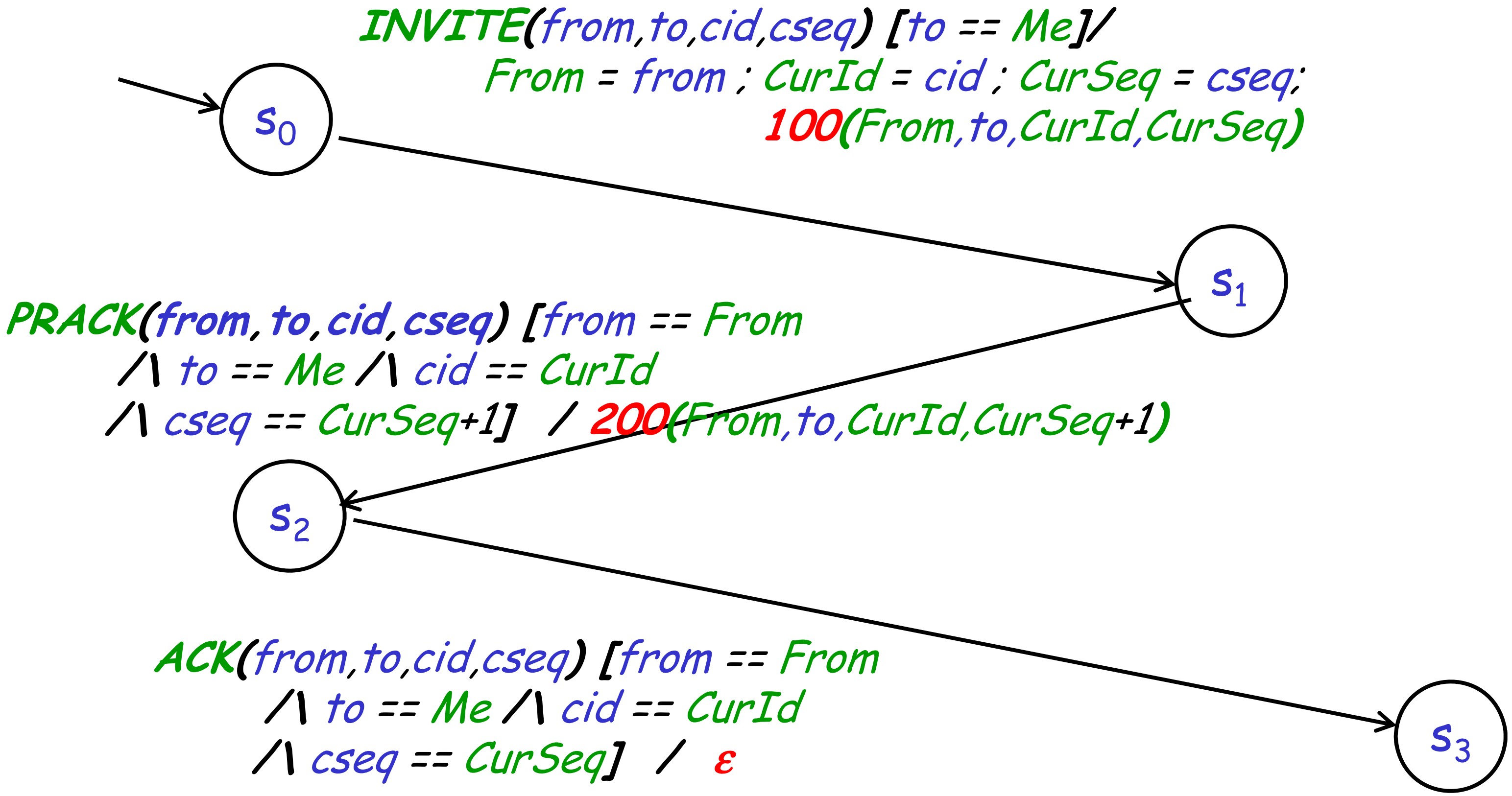
output



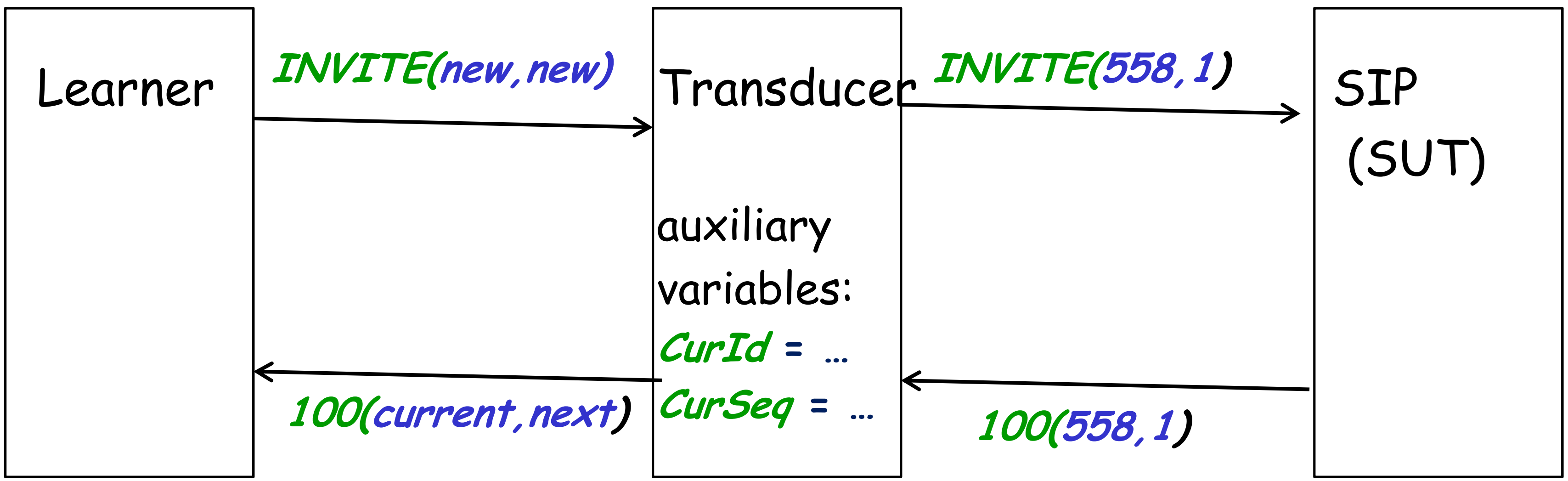
What about Protocols w. Data?

SIP Protocol (part of Server)

Variables: *From, CurId, CurSeq*
Constants: *Me*



Adapting to Automata Learning



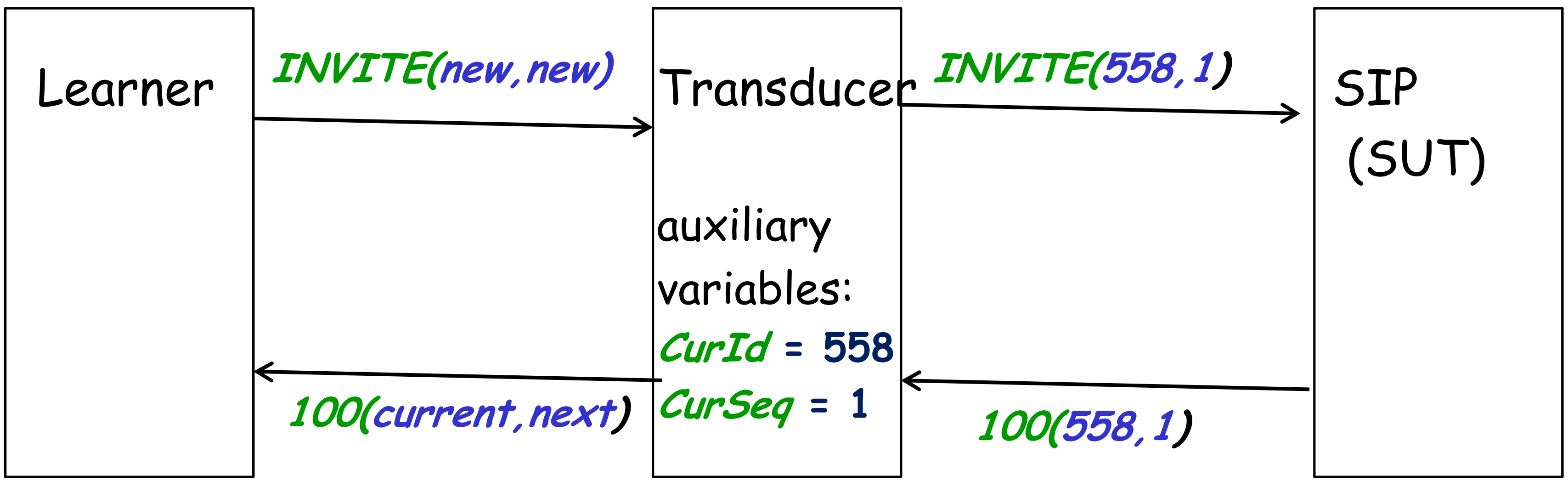
Mapping parameters of input messages

	new	current	other	
cid	CurId == “undef”	= CurId	!= CurId	
	new	**current**	**next**	**other**
cseq	CurSeq == “undef”	== CurSeq	== CurSeq+1	<other>

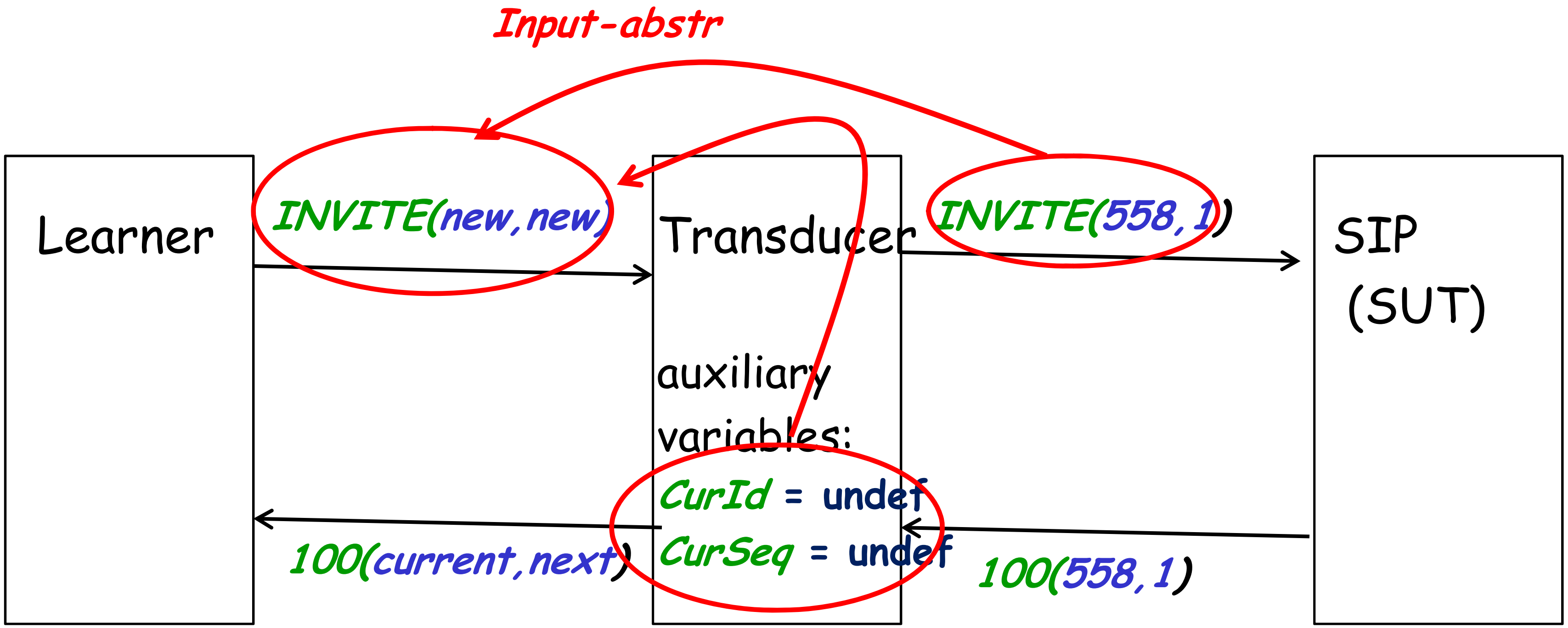
Maintaining auxiliary variables

	new	current	other	
CurId	= cid	<unchanged>	<unchanged>	
	new	**current**	**next**	**other**
CurSeq	= cseq	<unchanged>	<unchanged>	<unchanged>

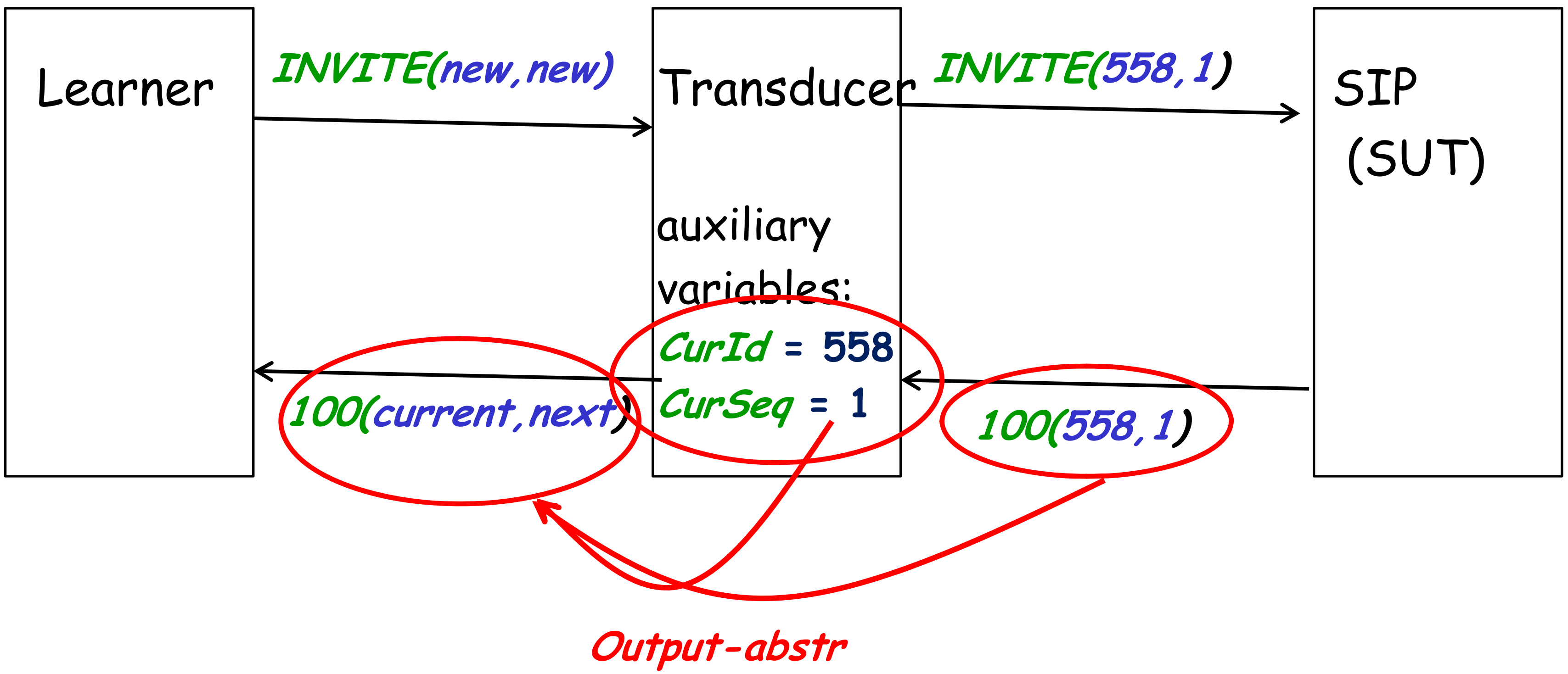
Inference by Abstraction



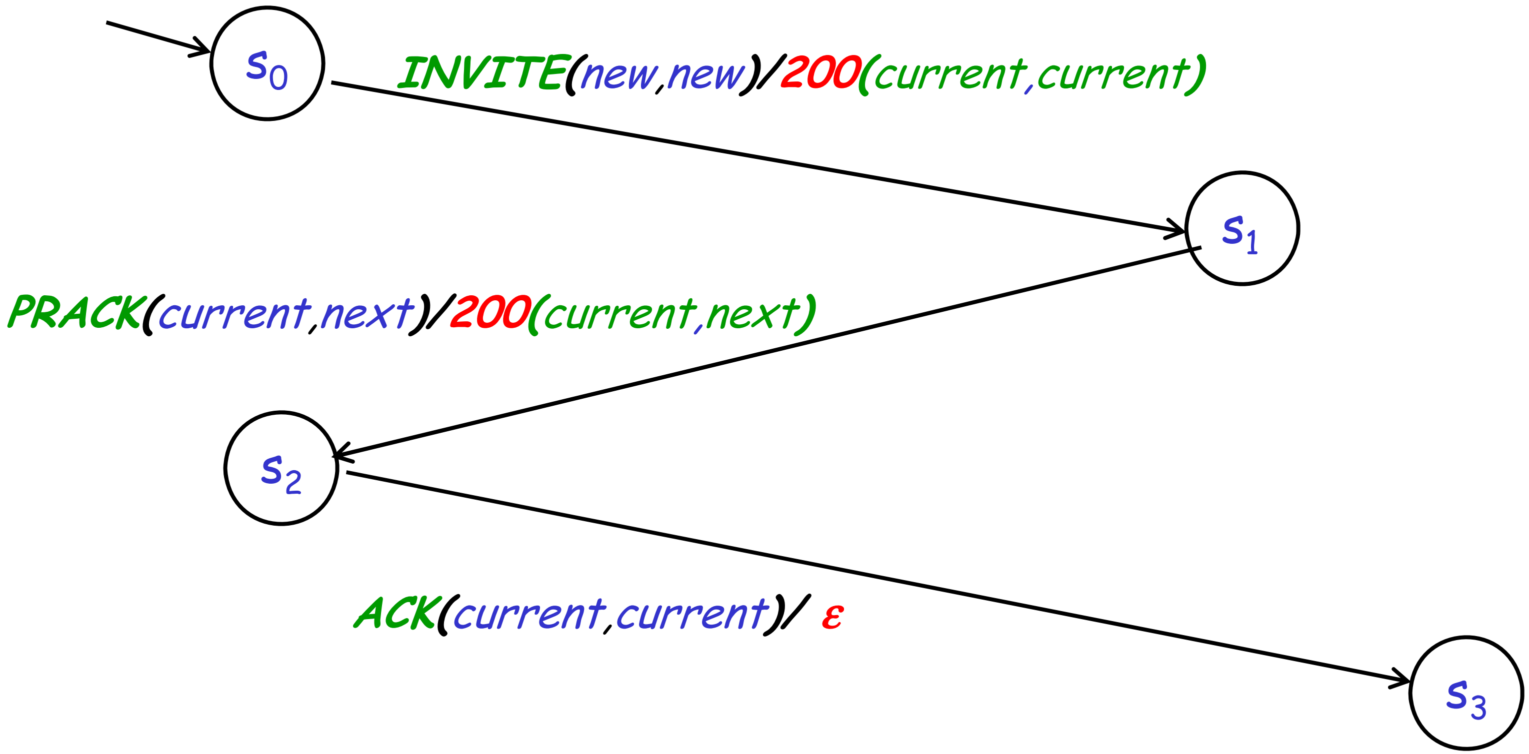
Abstraction Mappings



Abstraction Mappings

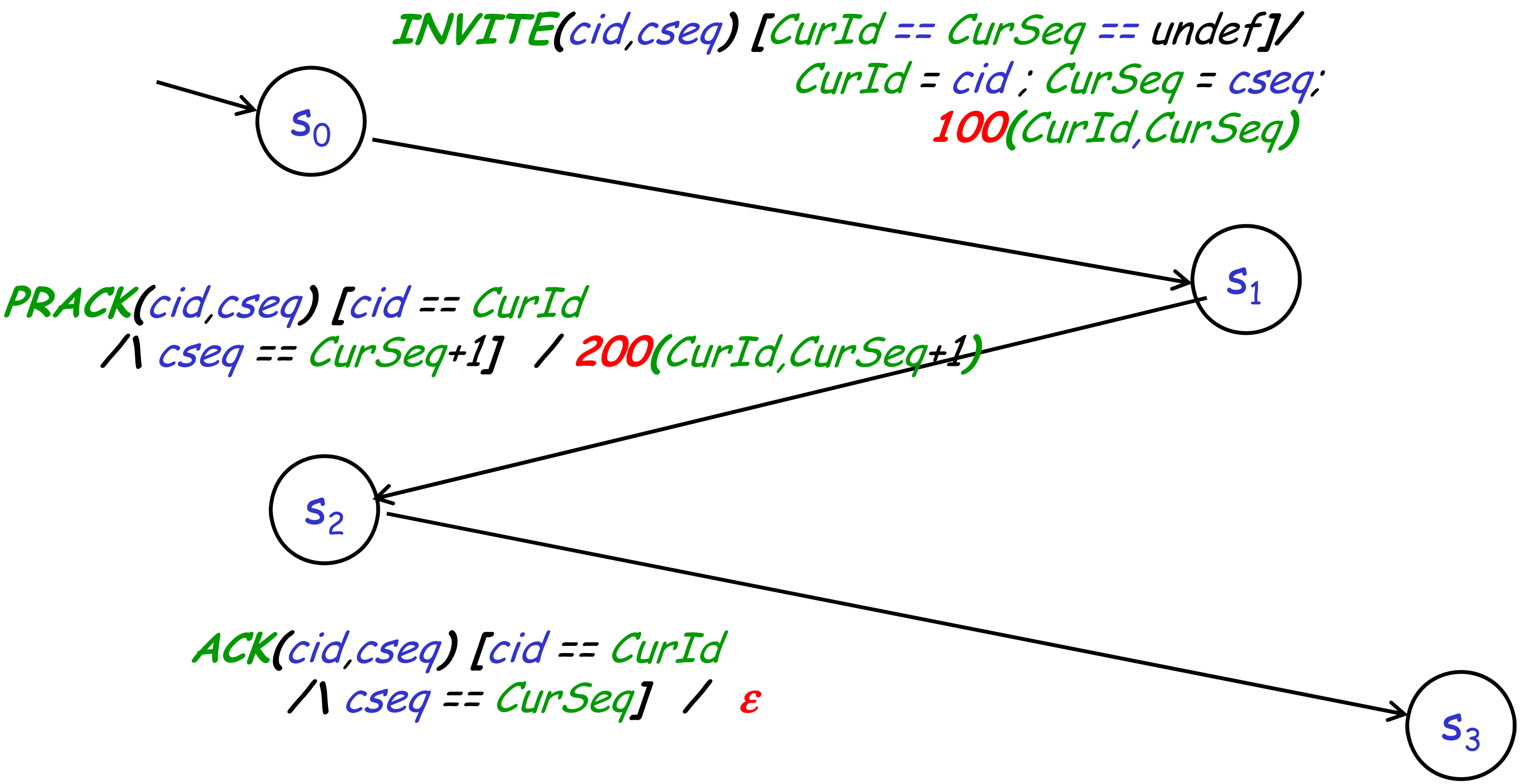


Model inferred by Learner (part)



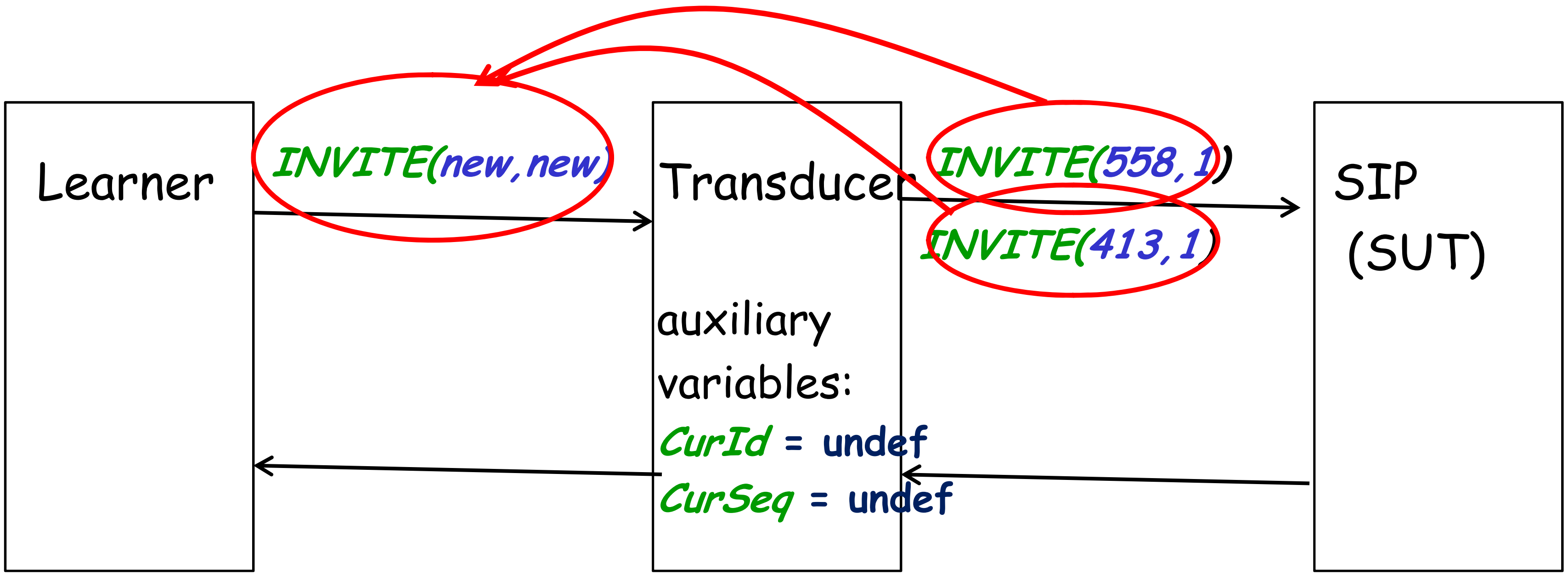
What the SUT must have done:

Variables: *CurId, CurSeq*



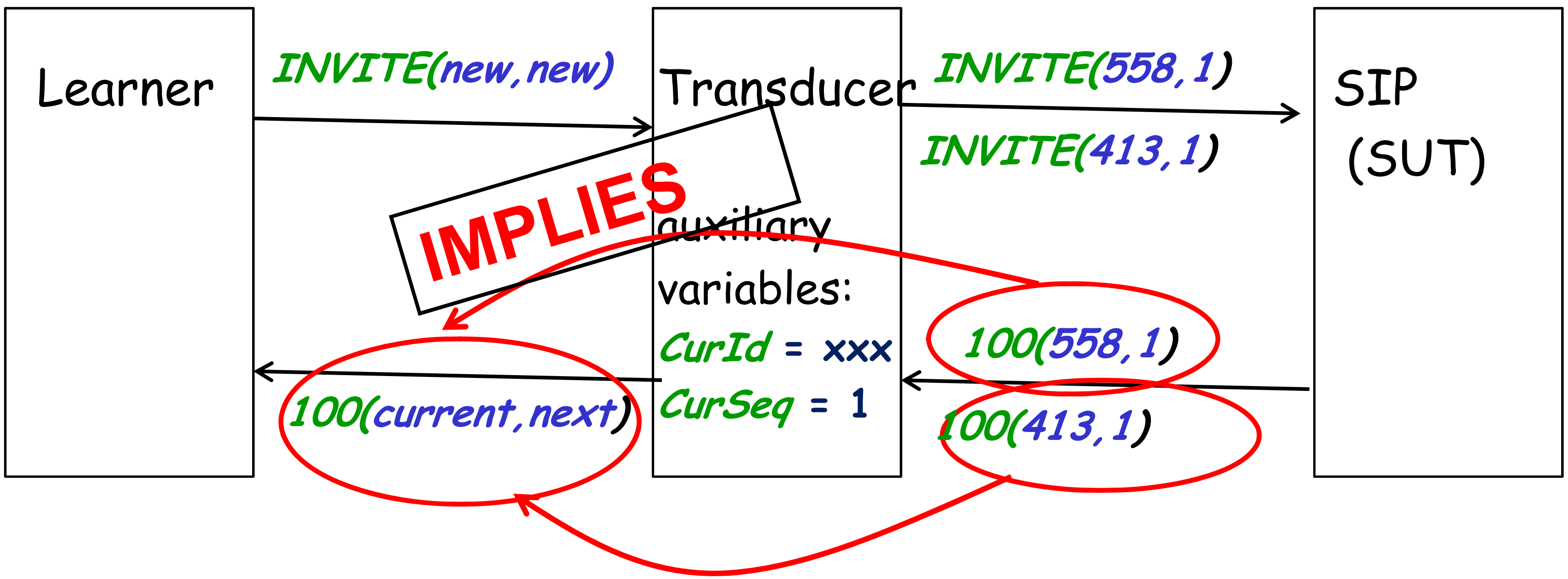
Healthiness condition:

Sufficiently Distinguishing input abstraction

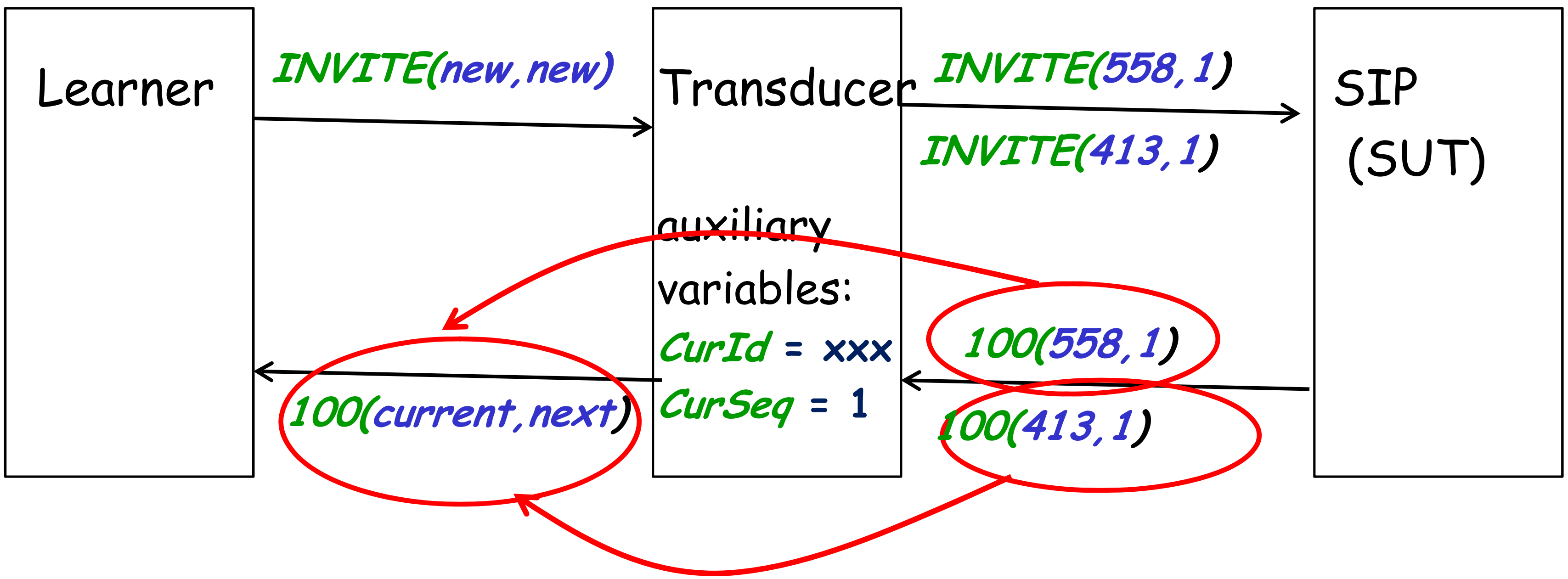


Healthiness condition:

Sufficiently Distinguishing input abstraction



Healthiness condition:



This does NOT guarantee that Learner will infer Finite Machine

Experiments

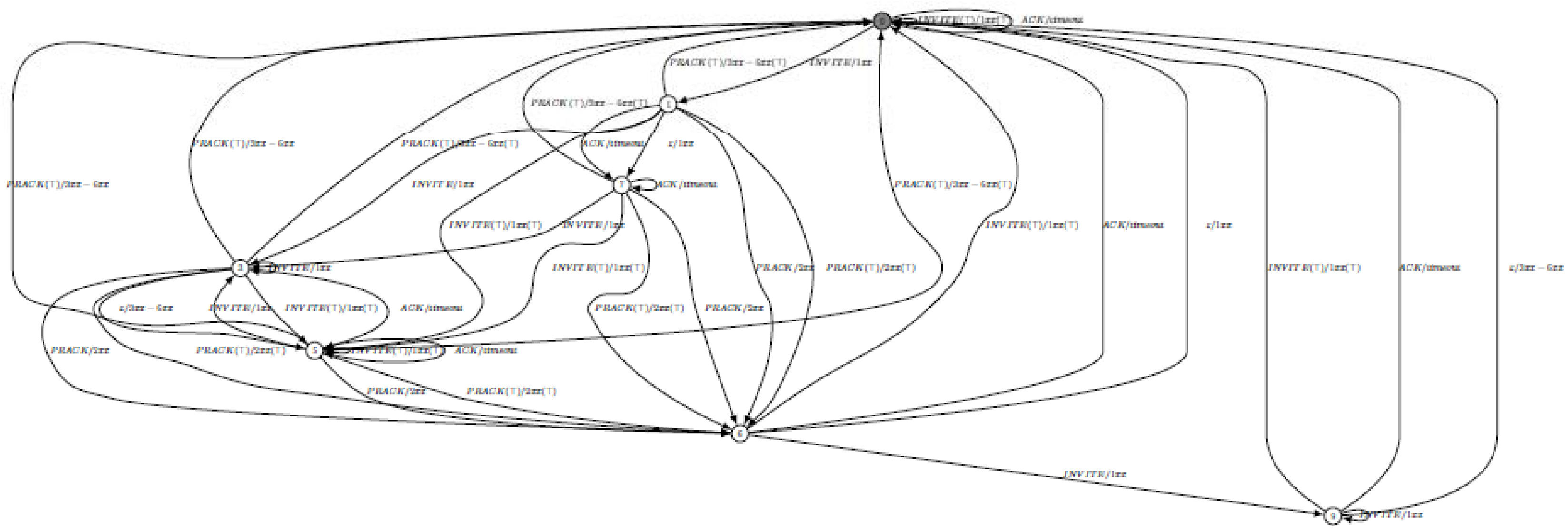
- Learner: the LearnLib tool (developed at TU Dortmund)
  - Efficient implementation of L*
  - Several equivalence oracles, e.g., controllable-size random test suite.
- SUT: ns-2 protocol simulator
  - Provides implementations of many standard protocols
  - Rather convenient C++ interface (no packet analyzer necessary)
- Transducer
  - Bridges asynchronous interface of LearnLib w. synchronous interface of ns-2
  - Implements instantiation of input symbols, and abstraction of output symbols

Session Initiation Protocol (SIP)

- Creating and Managing Multimedia protocol sessions
- SUT is ns-2 implementation of SIP Server
- Input messages have 7 parameters
- Each parameter abstracted to 2 or 3 values
- Inference: about 2 million membership queries
- Model w. 7 states and 41 transitions

SIP

- Model of behavior of SIP in ns-2
- SIP in ns-2 seems not to distinguish connected and unconnected state

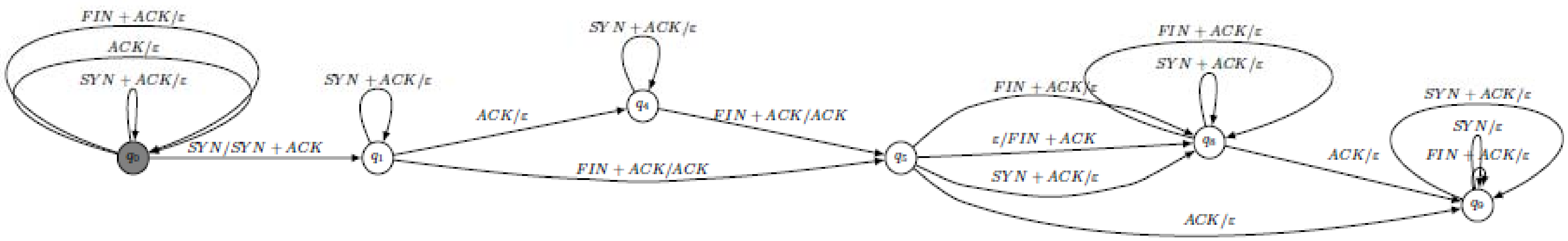


Transport Control Protocol (TCP)

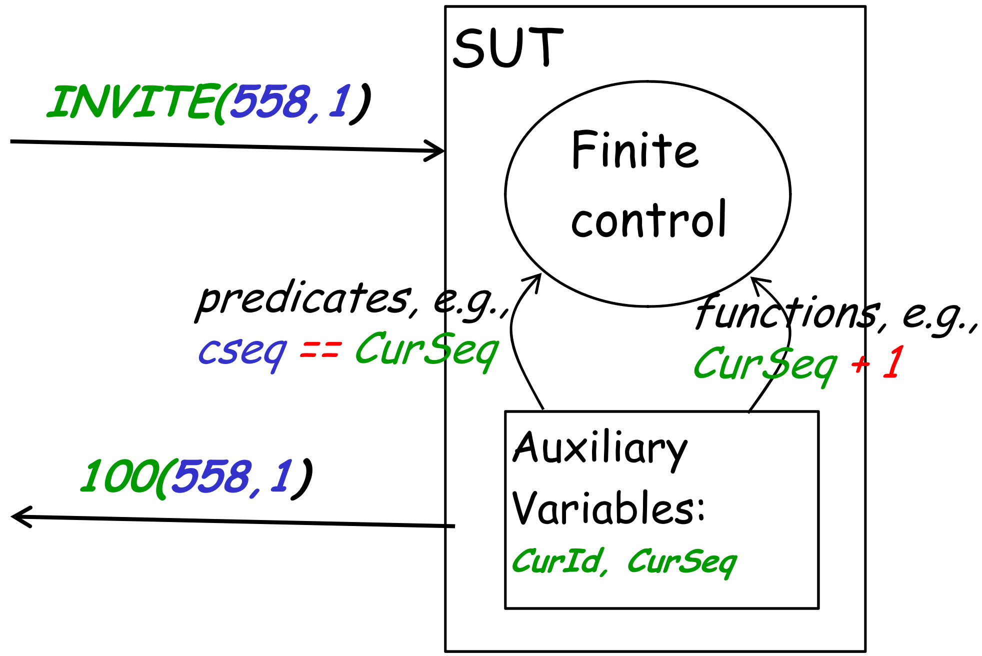
- Only connection establishment and termination
- SUT is ns-2 implementation of TCP
- Consider 2 sequence number parameters
- Each parameter abstracted to 2 or 3 values
- Model w. 33 states and 203 transitions

TCP

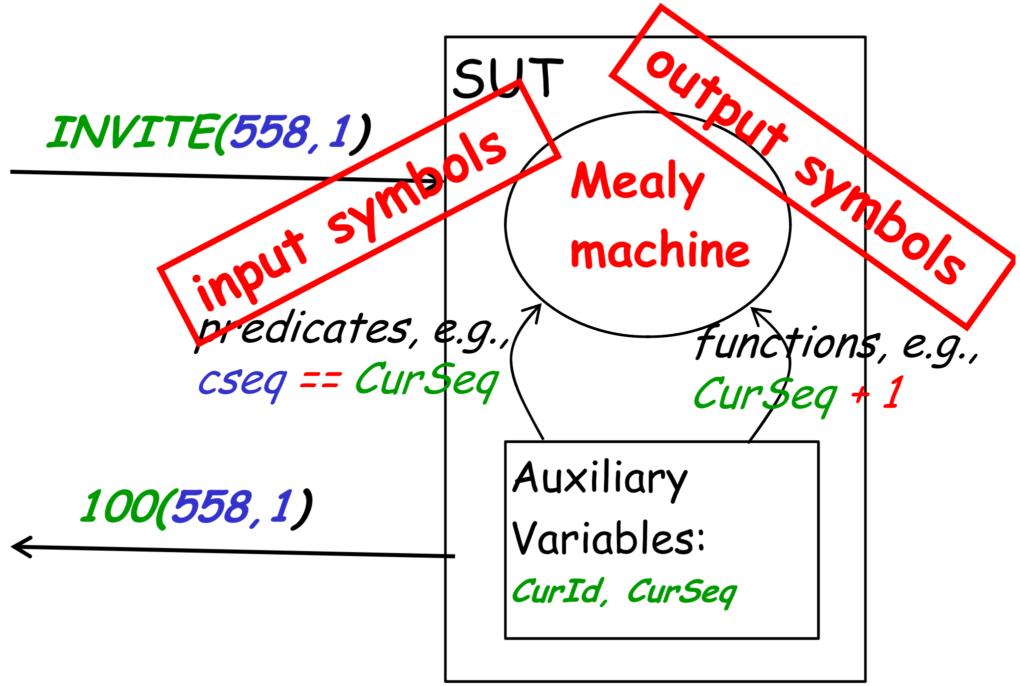
- Model of behavior of TCP in ns-2
- Only transitions with "accepted" values of input parameters are shown.
- Values of parameters not displayed



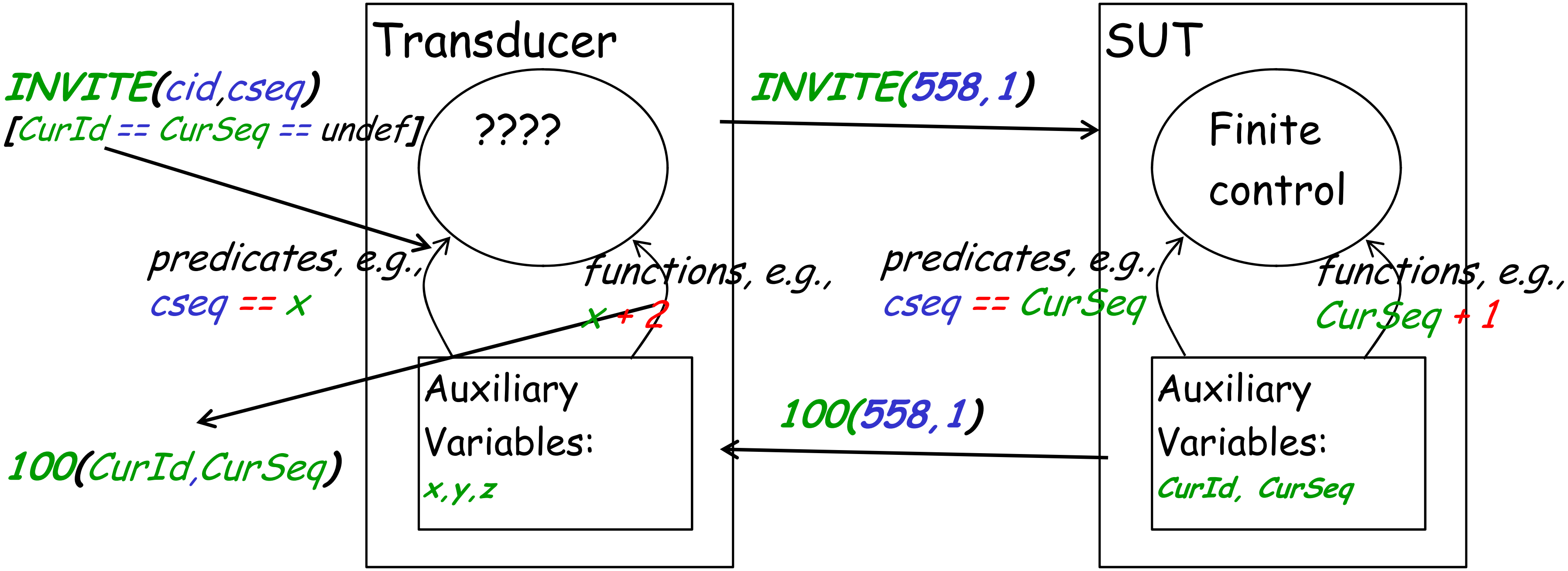
Conditions for Success:



Conditions for Success:



Conditions for Success:



If auxiliary variables in Transducer are more expressive than in SUT
i.e., all predicates and functions can be imitated,
Then finite control of Transducer can be a finite Mealy machine

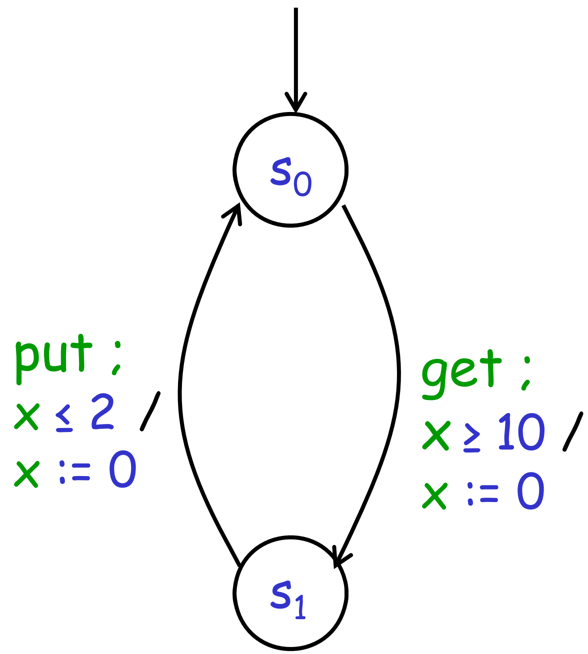
How find appropriate auxiliary variables, predicates, and functions?

Towards Automated Algorithm

- Infer input alphabet by successive refinement
- Library of commonly occurring alphabets,
- Adapt regular inference algorithm to dynamically changing alphabet

Timed Automata

- Based on standard automata
- Clocks give upper and lower bounds on distance in time between occurrences of symbols.
- Temporal properties of Timed Automata (reachability, LTL, ...) can be model-checked
- Implemented in tools (UPPAAL, IF/Kronos)

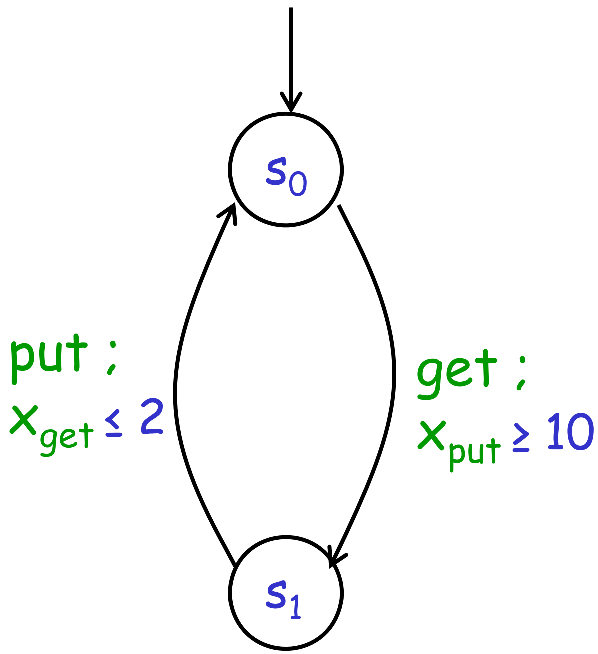


Inference of Event-Recording Automata [w. Olga Grinchtein]

- Timed Automata can not be determinized in general
- Event-Recording Automata (ERA): Each clock associated with particular symbol.
- ERA can be determinized

Assumption:

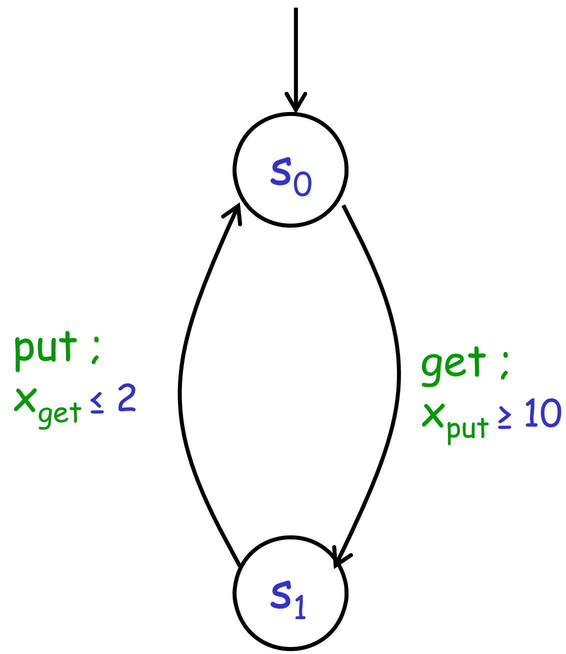
Inference algorithm can precisely control and record timing of symbols.



Inference of ERAs

Problems:

- Determine guards
- Can be seen as inferring the input alphabet
- Done by refinement from observations of nondeterminism



Refinement of guards

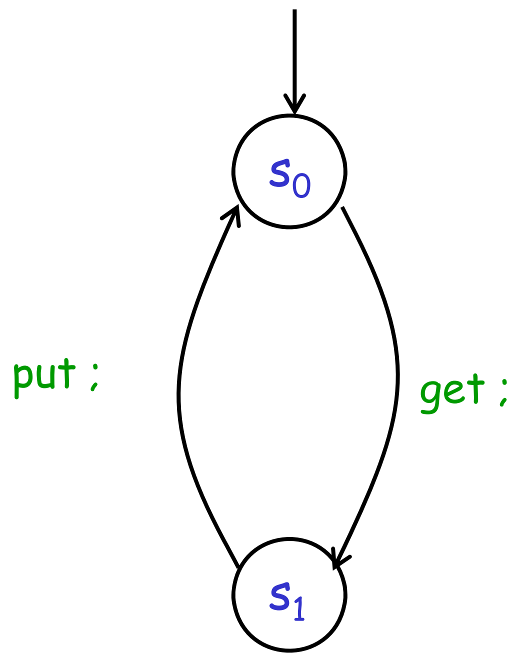
Start from untimed alphabet

Guards refined from nondeterminism

- get @0 put @2 accepted
- get @3 put @7 rejected

Determine the reason for difference by investigating other traces

- (binary) search procedure
- Finds "explaining pair", e.g.,
  - get @2 put @4 accepted
  - get @2 put @4.5 rejected



Refinement of guards

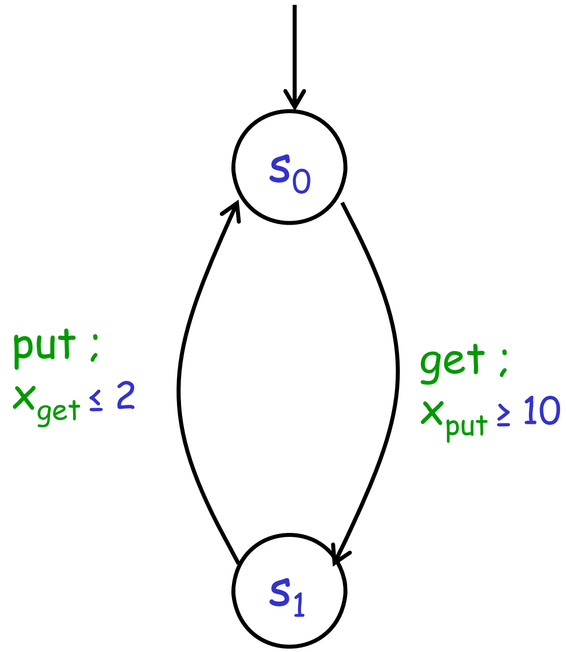
Start from untimed alphabet

Guards refined from nondeterminism

- get @0 put @2 accepted
- get @3 put @7 rejected

Determine the reason for difference by investigating other traces

- (binary) search procedure
- Finds "explaining pair", e.g.,
  - get @2 put @4 accepted
  - get @2 put @4.5 rejected
- Suggests guard $x_{get} \leq 2$ on put transition



Conclusions

- State machine models of communication protocols can be inferred
  - Using a priori knowledge about primitives for data manipulation
- The primitives can be inferred, *given constraints on their form*

Future work

- Library of common data structures
- Automatic generation of transducers
- Automated inference of input and output symbols
- Adapted Learning algorithm and implementation
- Incorporating nondeterminism
- Systematic coverage of possible concretizations of abstract symbols (= test input selection)