Smyle: a Tool for Synthesizing Distributed Models from Scenarios by Learning

Benedikt Bollig[1] Joost-Pieter Katoen[2]
Carsten Kern[2] Martin Leucker[3]

[1] Laboratoire Spécification et Vérification

[2] Lehrstuhl für Informatik 2

[3] Institut für Informatik

CONCUR 2008, August 19^{th}

Outline

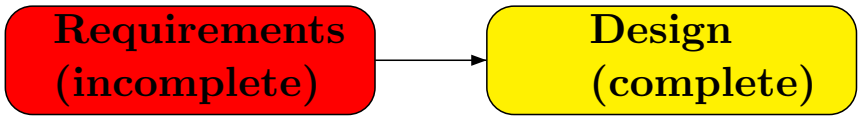
Presentation outline

Motivation

Requirements (incomplete)

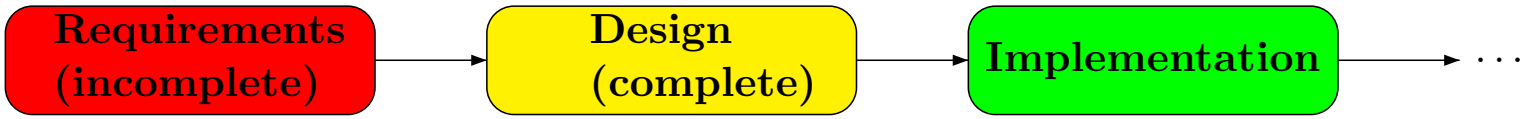
- initial phase: requirement elicitation
  - contradicting or incomplete system description
  - common description language: sequence diagrams
- goal: conforming design model
- closing gap between
  - requirement specification (usually incomplete) and
  - design model (complete description of system)

Motivation



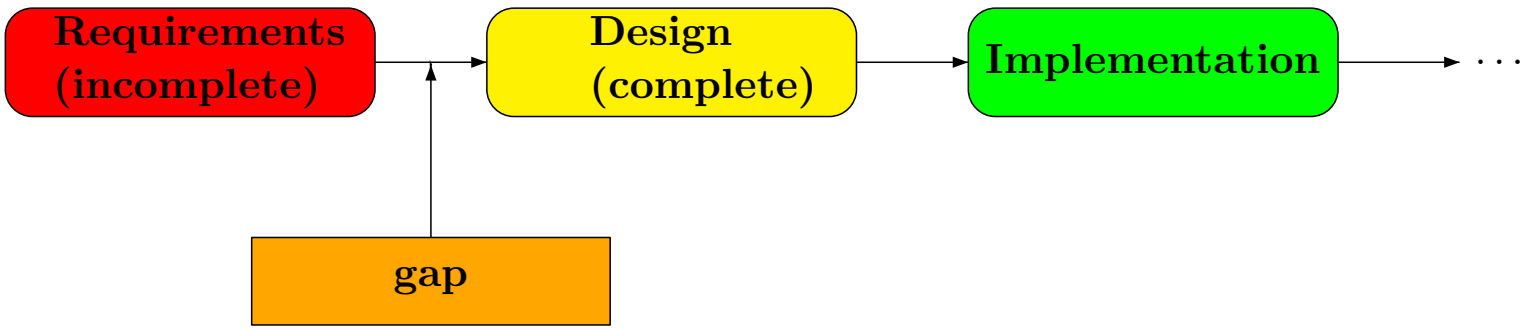
- initial phase: requirement elicitation
  - contradicting or incomplete system description
  - common description language: sequence diagrams
- goal: conforming design model
- closing gap between
  - requirement specification (usually incomplete) and
  - design model (complete description of system)

Motivation

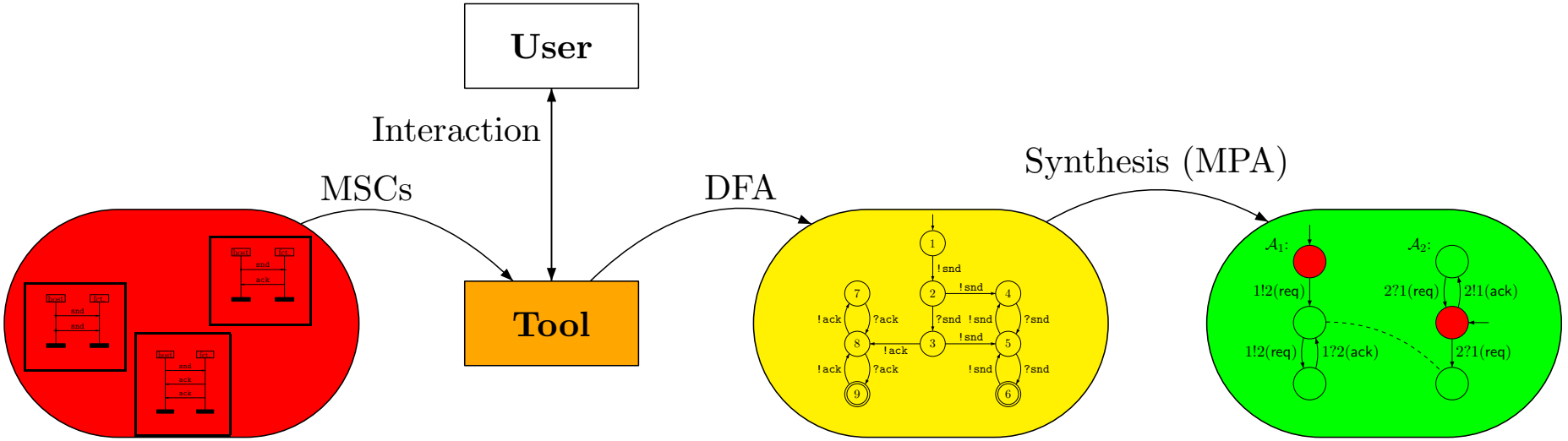


- initial phase: requirement elicitation
  - contradicting or incomplete system description
  - common description language: sequence diagrams
- goal: conforming design model
- closing gap between
  - requirement specification (usually incomplete) and
  - design model (complete description of system)

Motivation



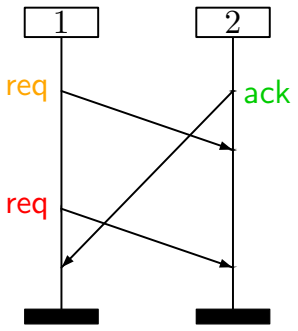
- initial phase: requirement elicitation
  - contradicting or incomplete system description
  - common description language: sequence diagrams
- goal: conforming design model
- closing gap between
  - requirement specification (usually incomplete) and
  - design model (complete description of system)



Our Approach

- use learning algorithms to synthesize models for communication protocols
- **Input:** set of Message Sequence Charts
  - standardized: ITU Z.120
  - included in UML as sequence diagrams
- **Output:** MPA fulfilling the specification
  - model is close to implementation

Message Sequence Chart (MSC)



An MSC ...

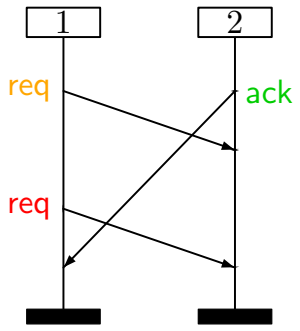
- represents a system execution
- can be regarded as a labelled partial order or
- a set of words (*linearizations*)

Some linearizations

- $1!2(\text{req})\ 1!2(\text{req})\ 2!1(\text{ack})\ 1?2(\text{ack})\ 2?1(\text{req})\ 2?1(\text{req})$
- $1!2(\text{req})\ 2!1(\text{ack})\ 1!2(\text{req})\ 1?2(\text{ack})\ 2?1(\text{req})\ 2?1(\text{req})$
- $2!1(\text{ack})\ 1!2(\text{req})\ 2?1(\text{req})\ 1!2(\text{req})\ 2?1(\text{req})\ 1?2(\text{ack})$
- ...

An MSC M is uniquely determined by its linearizations.

Message Sequence Chart (MSC)



An MSC ...

- represents a system execution
- can be regarded as a labelled partial order or
- a set of words (*linearizations*)

Some linearizations

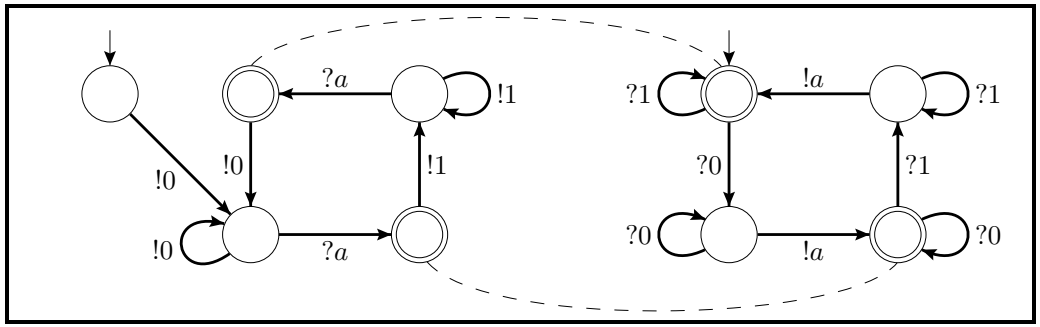
- 1!2(req) 1!2(req) 2!1(ack) 1?2(ack) 2?1(req) 2?1(req)
- 1!2(req) 2!1(ack) 1!2(req) 1?2(ack) 2?1(req) 2?1(req)
- 2!1(ack) 1!2(req) 2?1(req) 1!2(req) 2?1(req) 1?2(ack)
- ...

An MSC M is uniquely determined by its linearizations.

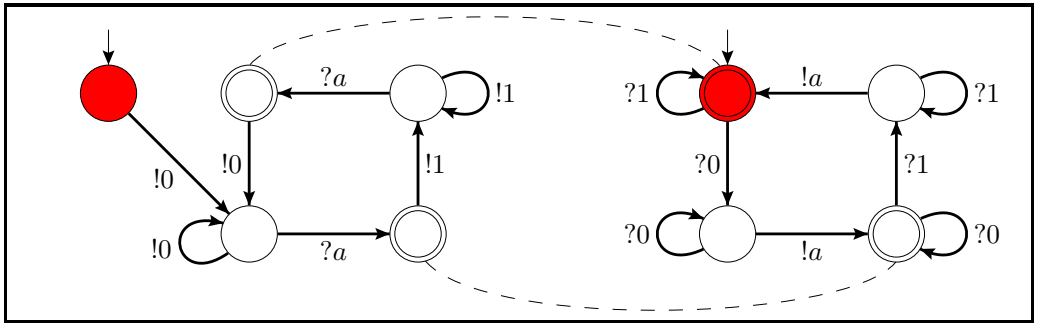
Message Passing Automata (MPA)

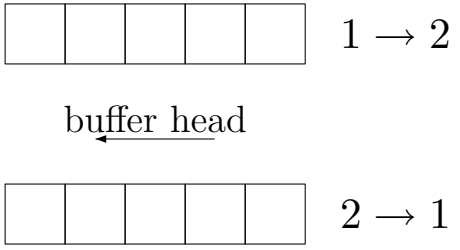
An MPA consists of:

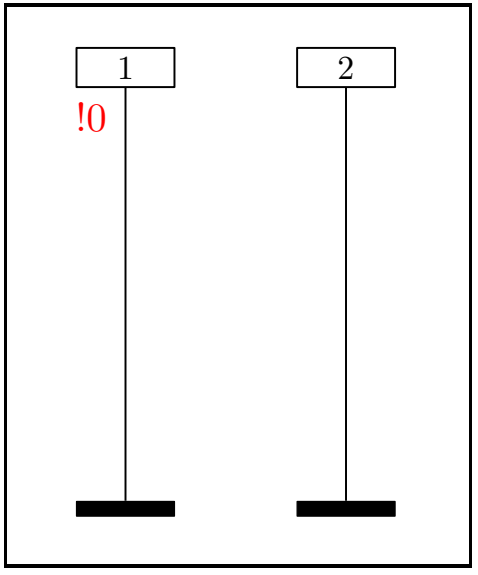
- a set of finite-state automata (*processes*) with
  - common global initial state
  - set of global final states
- communication between automata through (reliable) FIFO channels
  - $p!q(a)$ appends message a to buffer between p and q
  - $q?p(a)$ removes message a from buffer between p and q



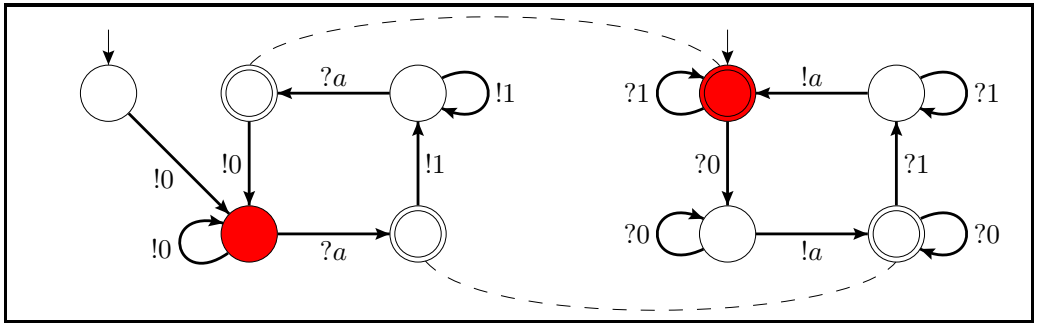
MPA: An Example

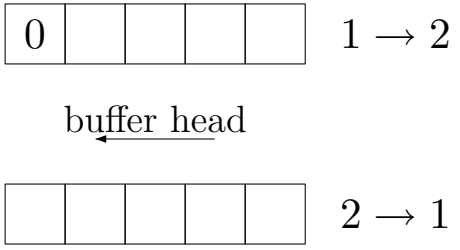


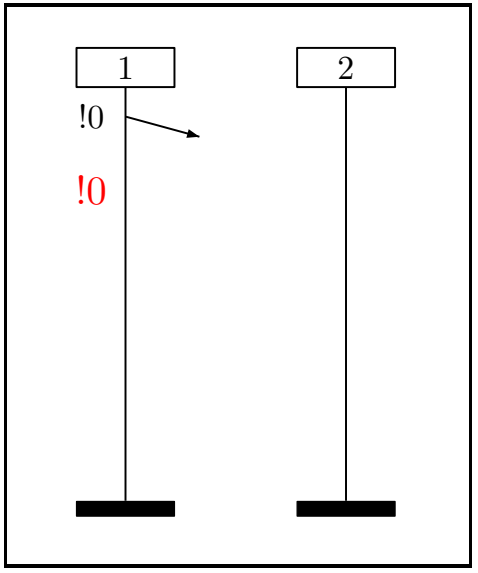




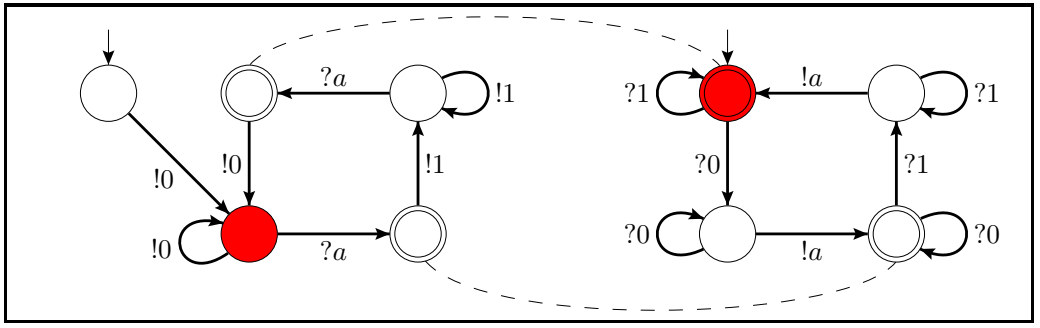
MPA: An Example

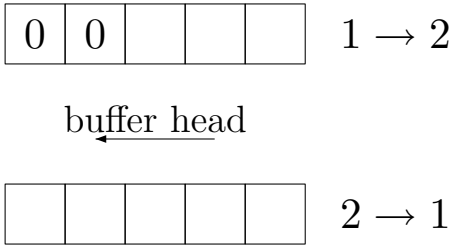


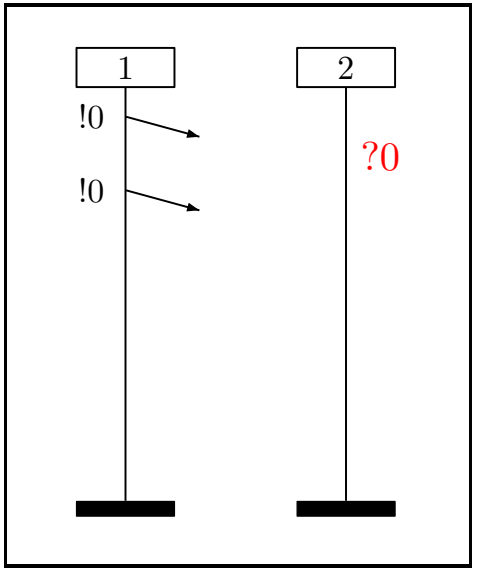




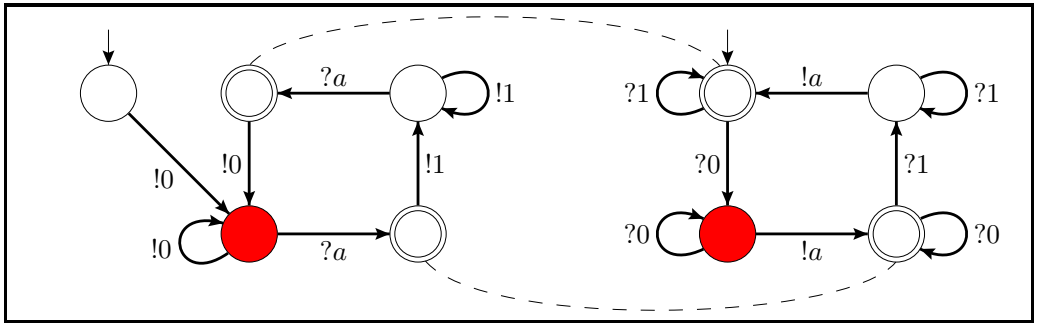
MPA: An Example

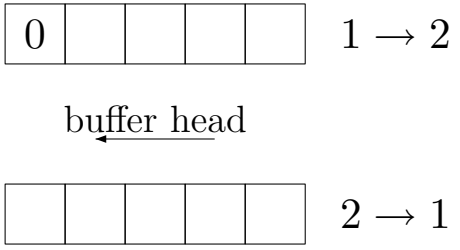


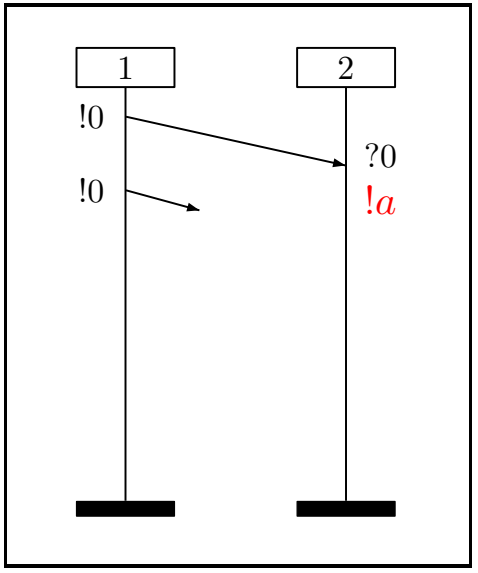




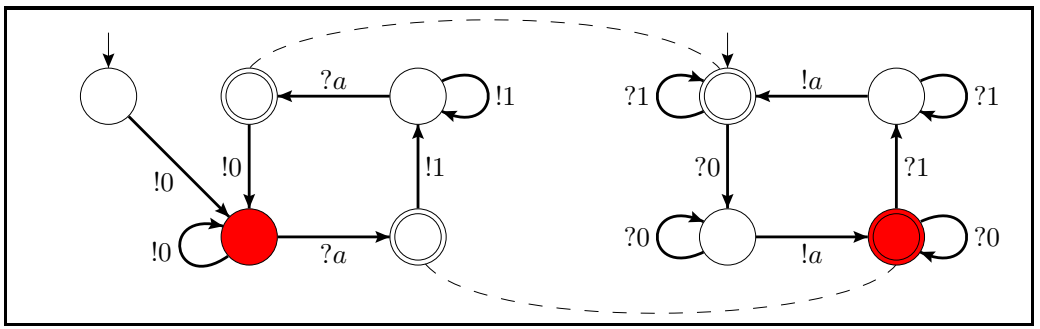
MPA: An Example

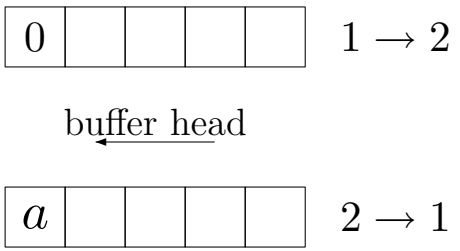


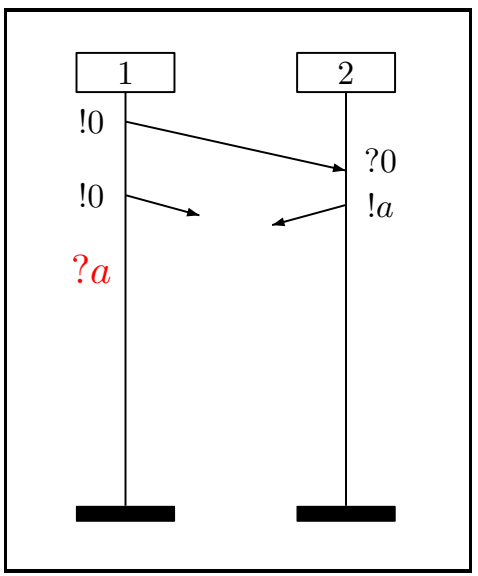




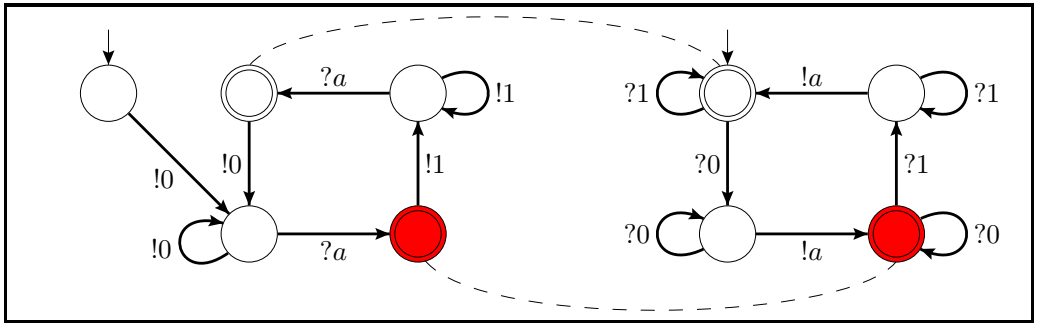
MPA: An Example

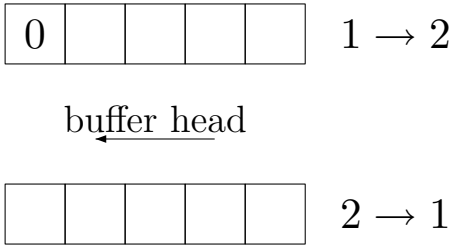


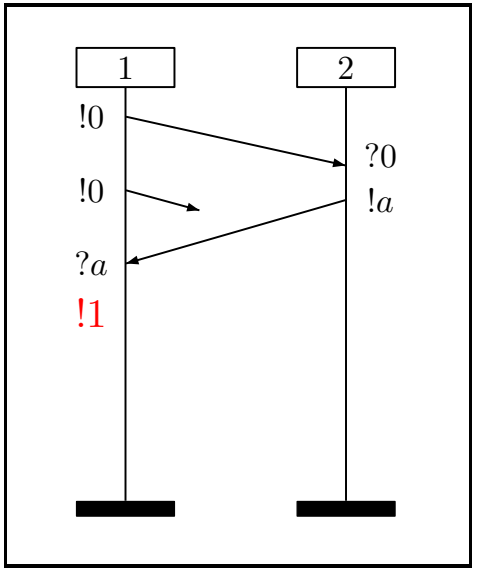




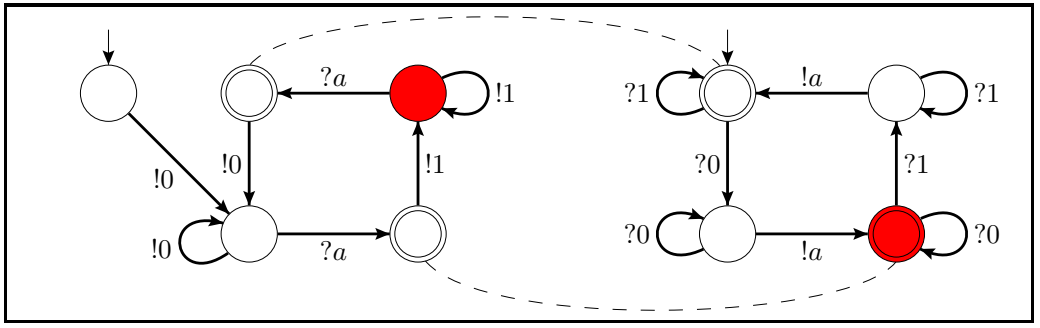
MPA: An Example

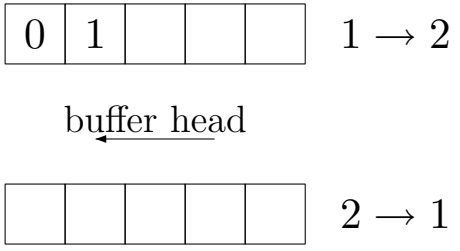


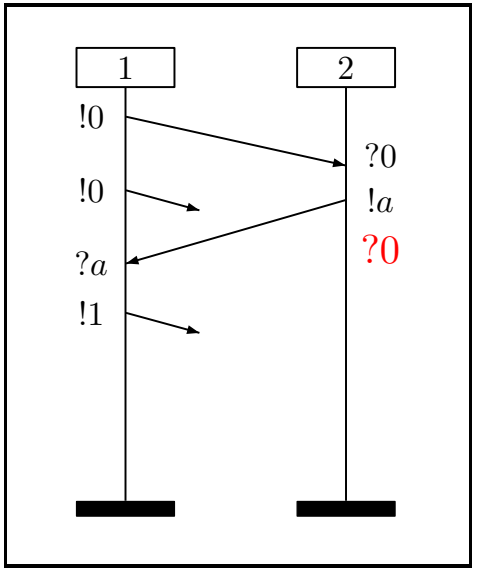




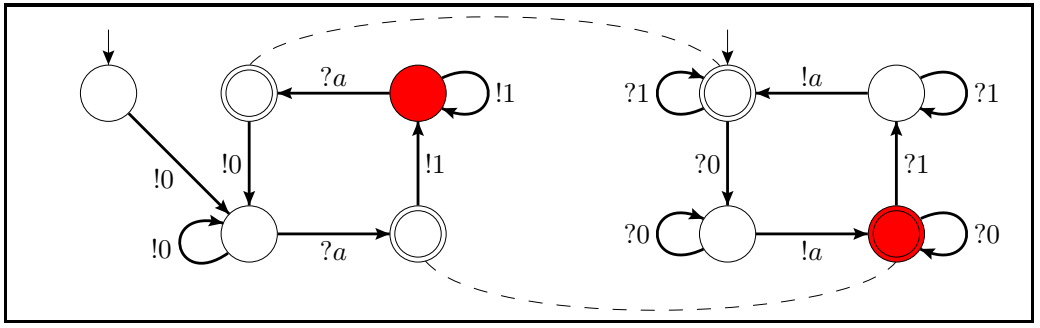
MPA: An Example

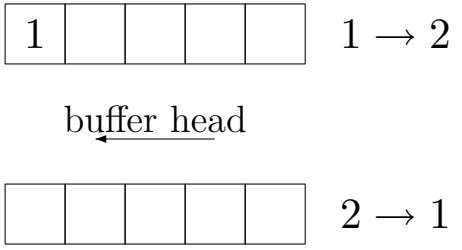


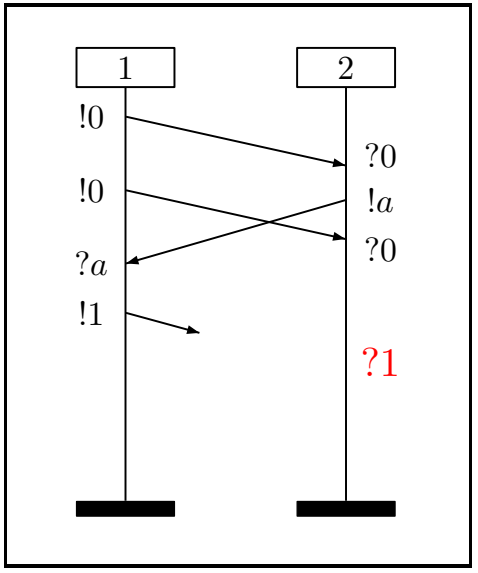




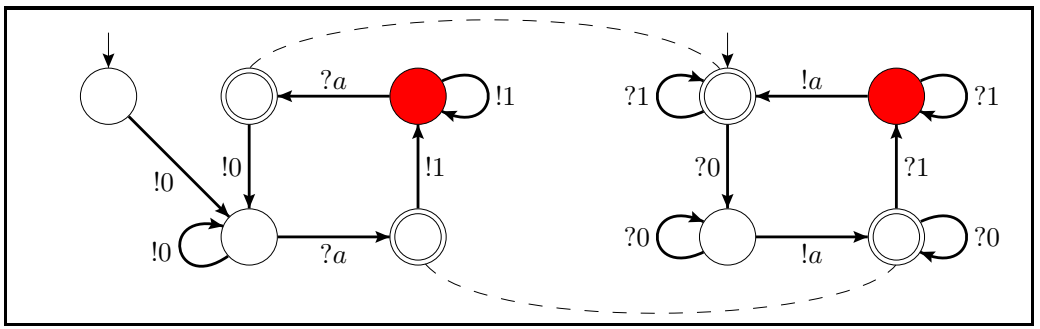
MPA: An Example

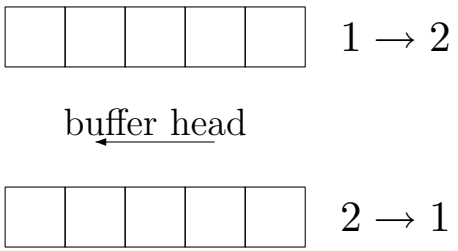


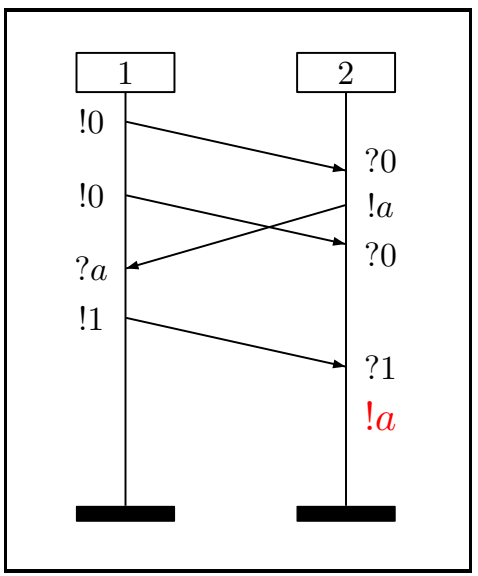




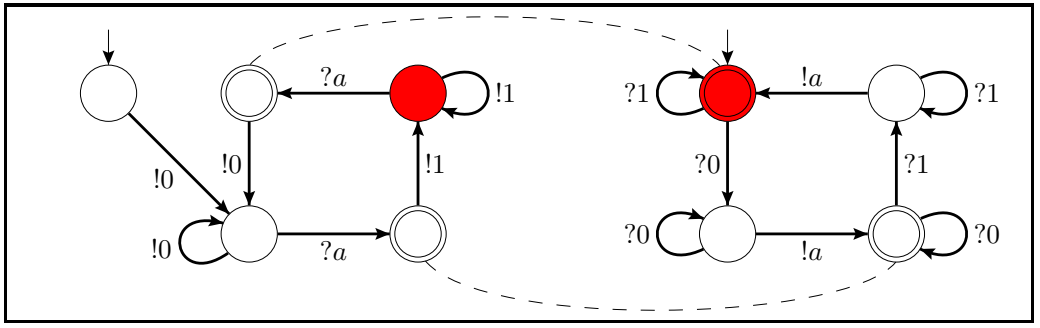
MPA: An Example

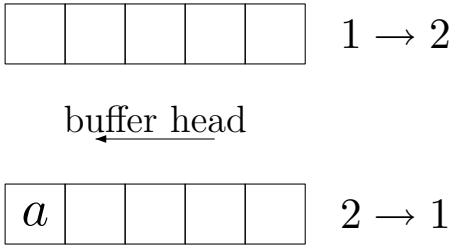


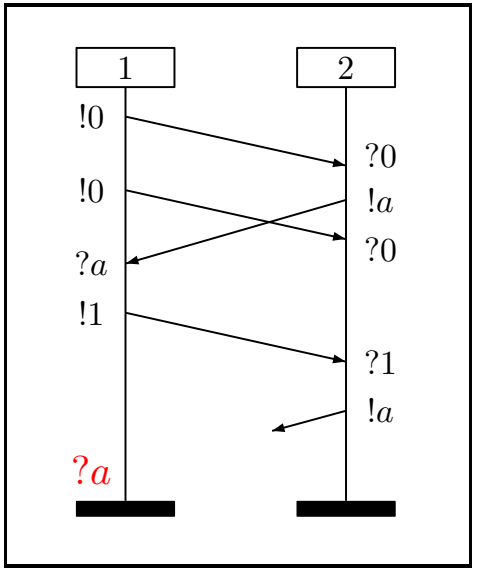




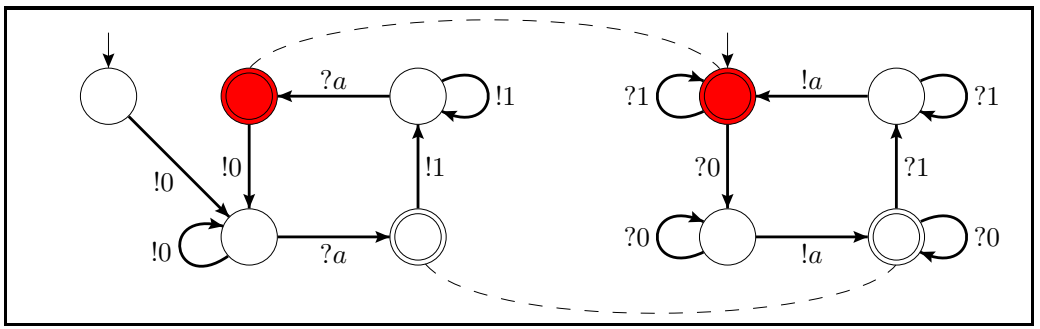
MPA: An Example

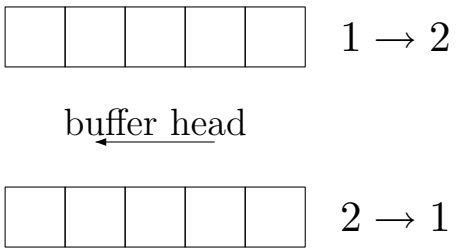


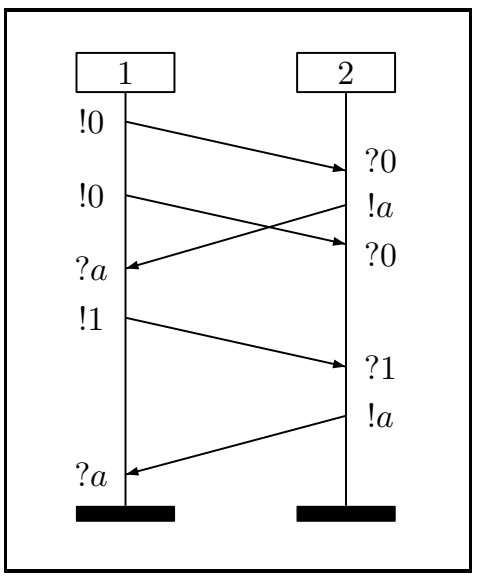




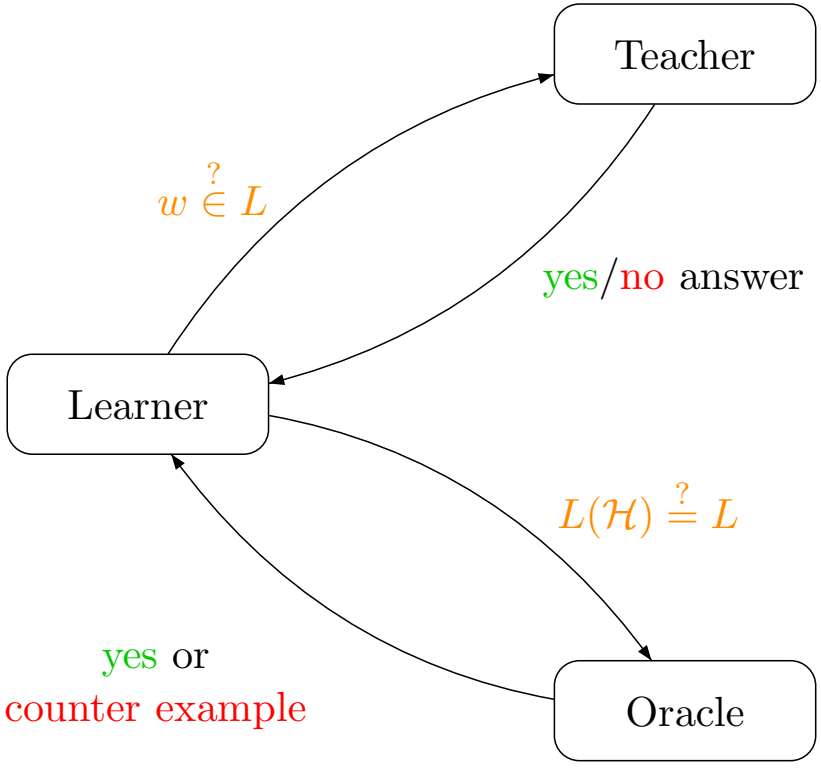
MPA: An Example





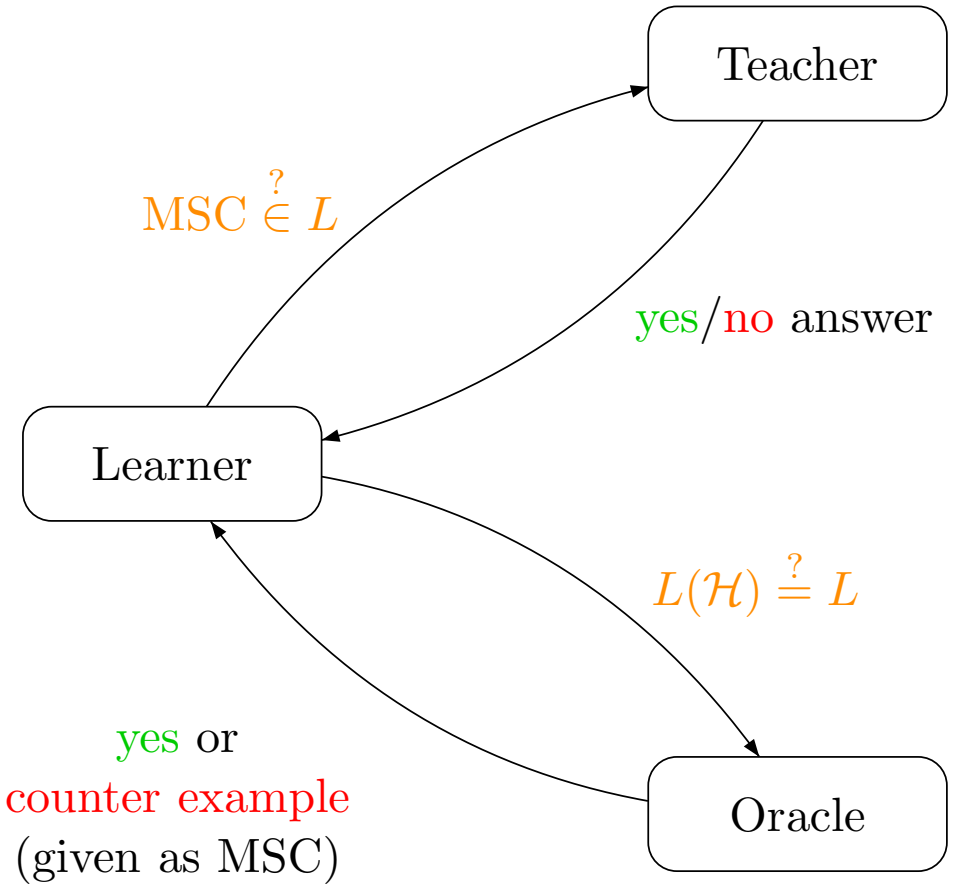


Angluin's algorithm (L^*)

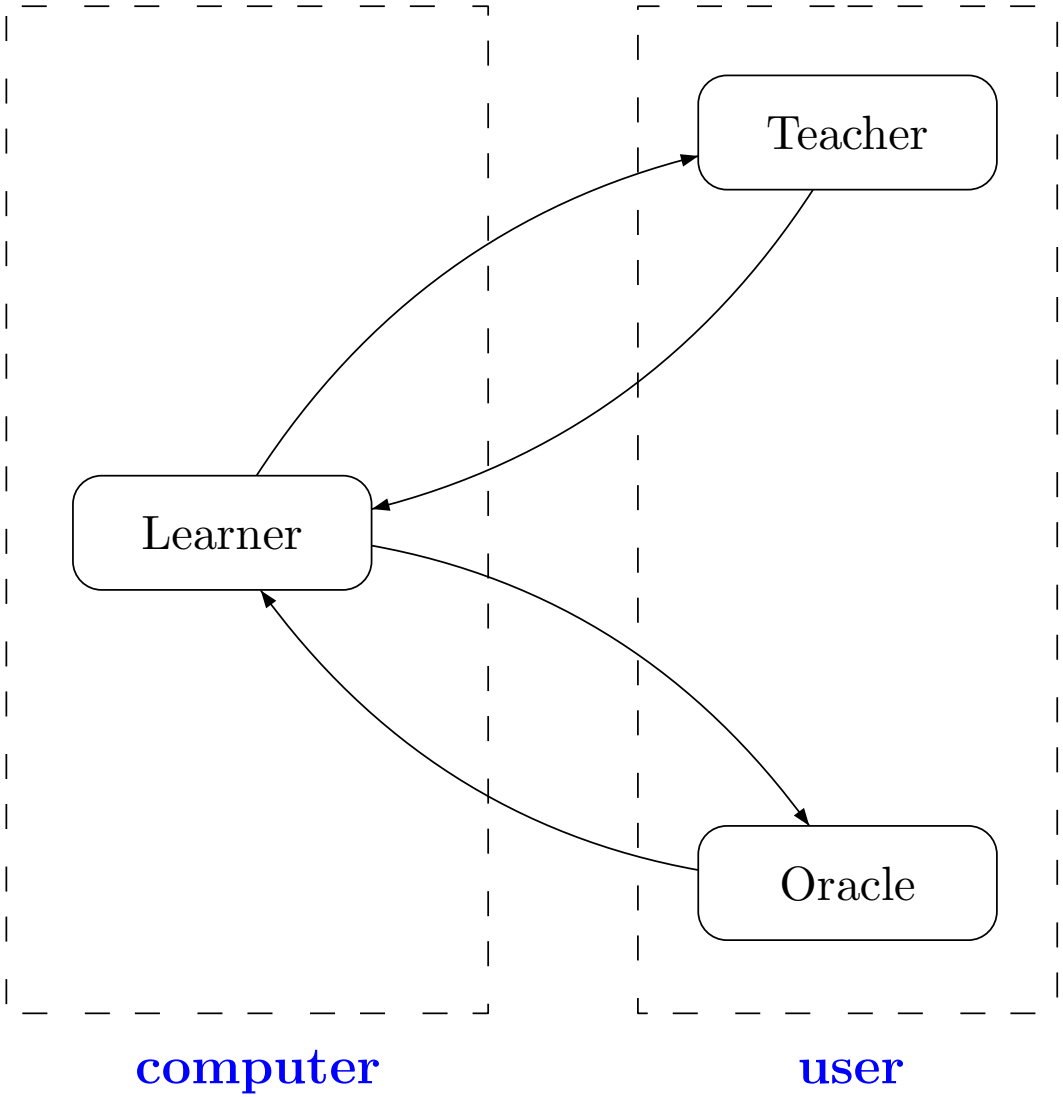


- L : language to learn
- $\mathcal{H}$: hypothetical (learned) automaton

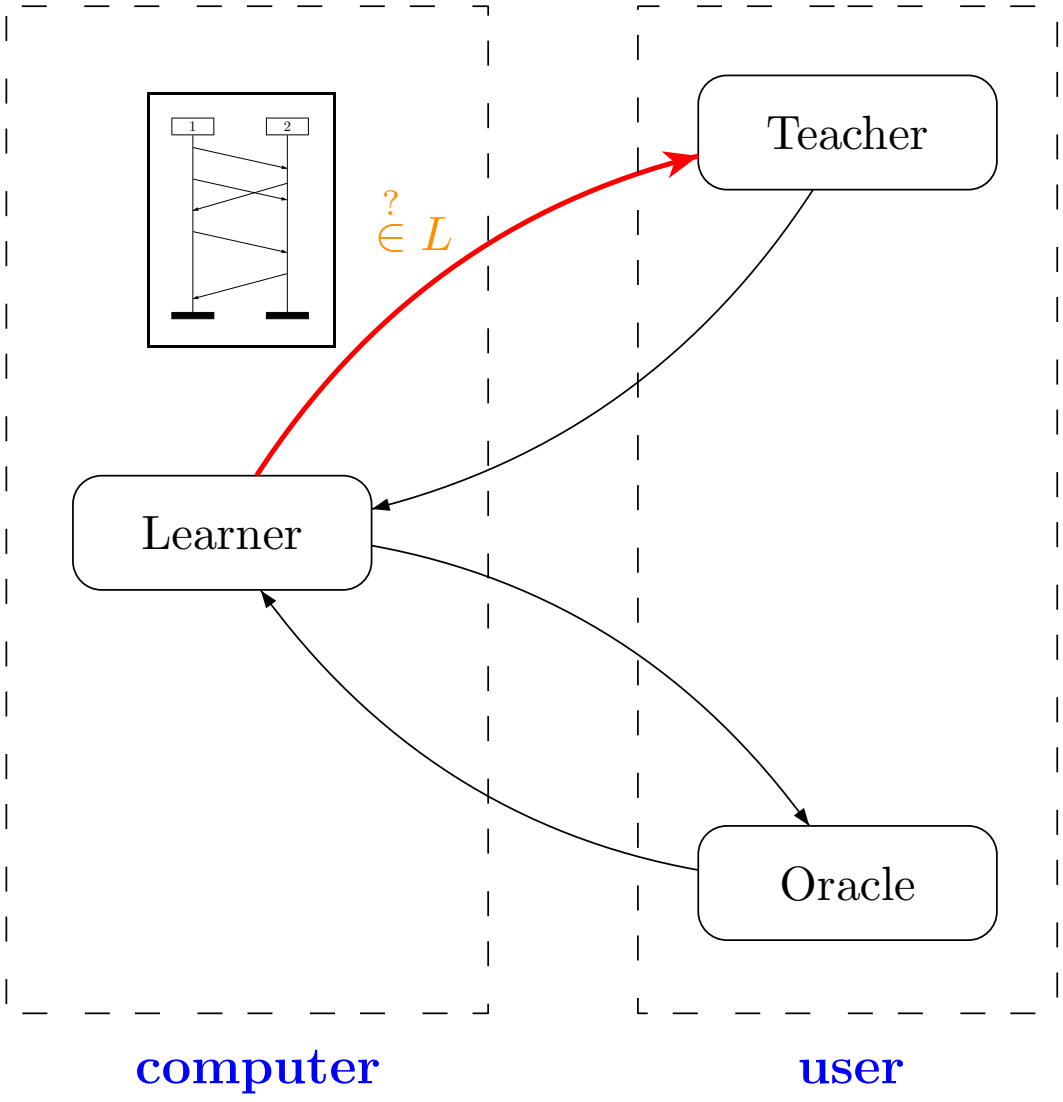
Angluin's algorithm: extended



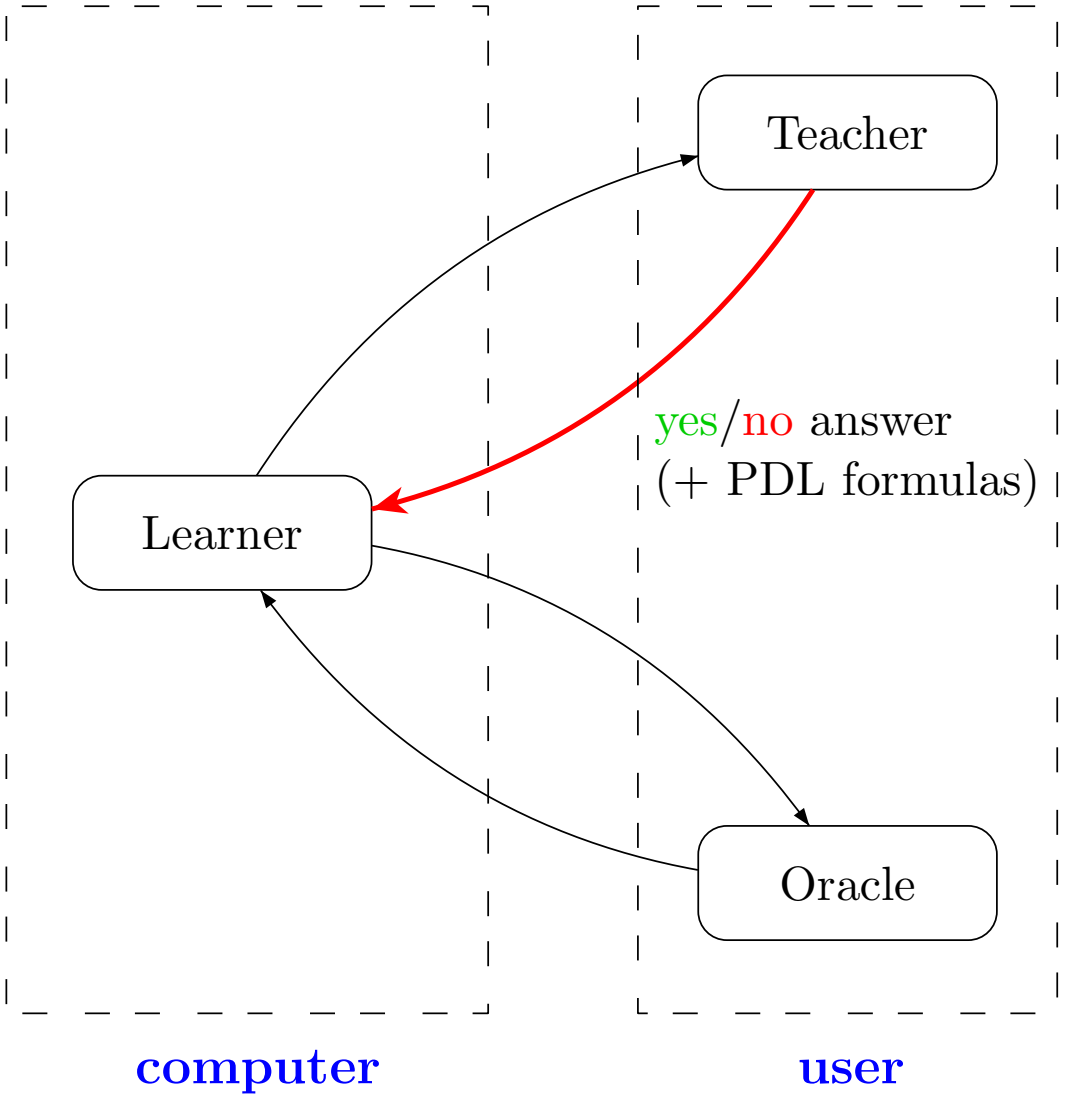
Angluin's algorithm: extended



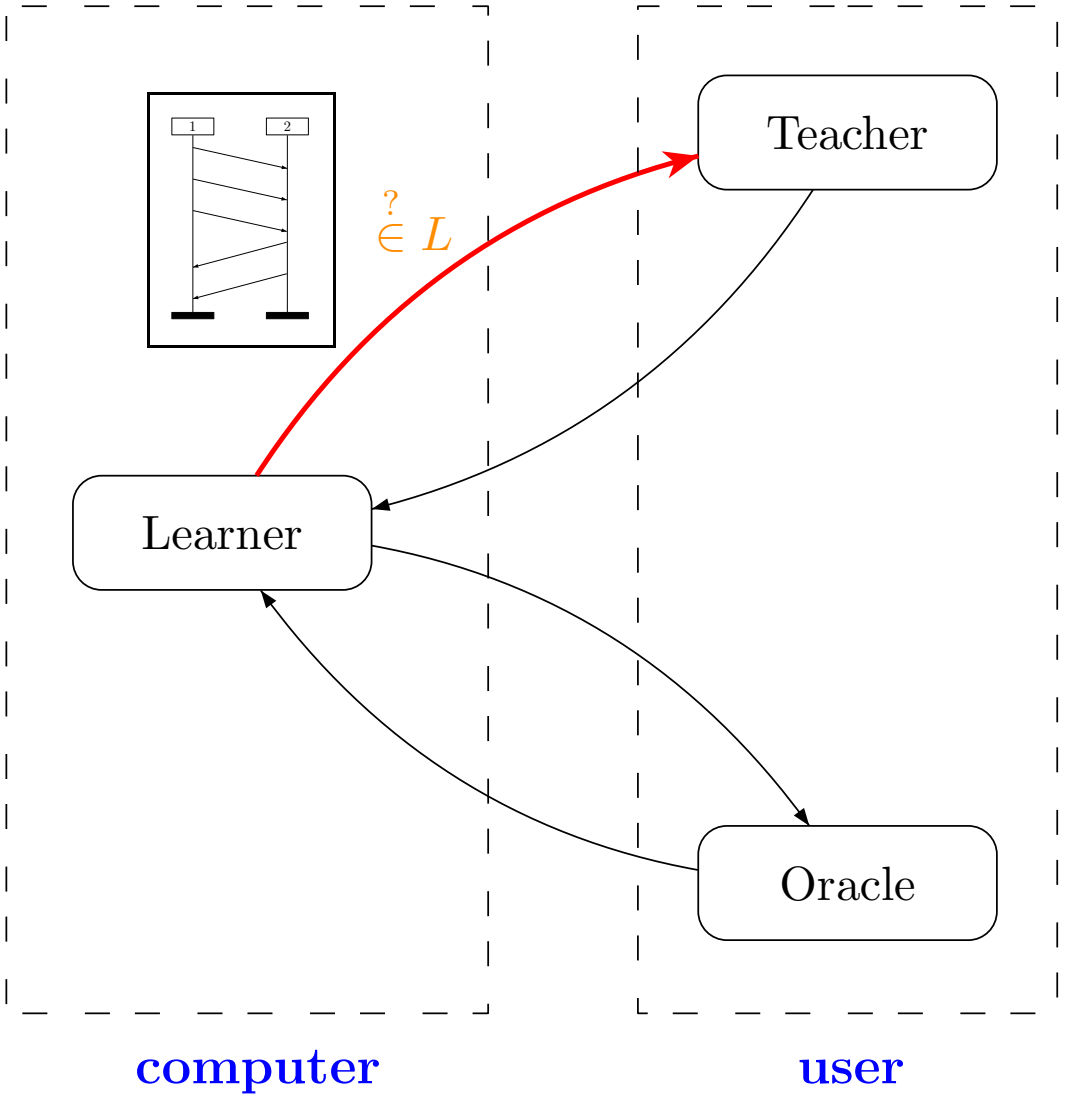
Angluin's algorithm: extended



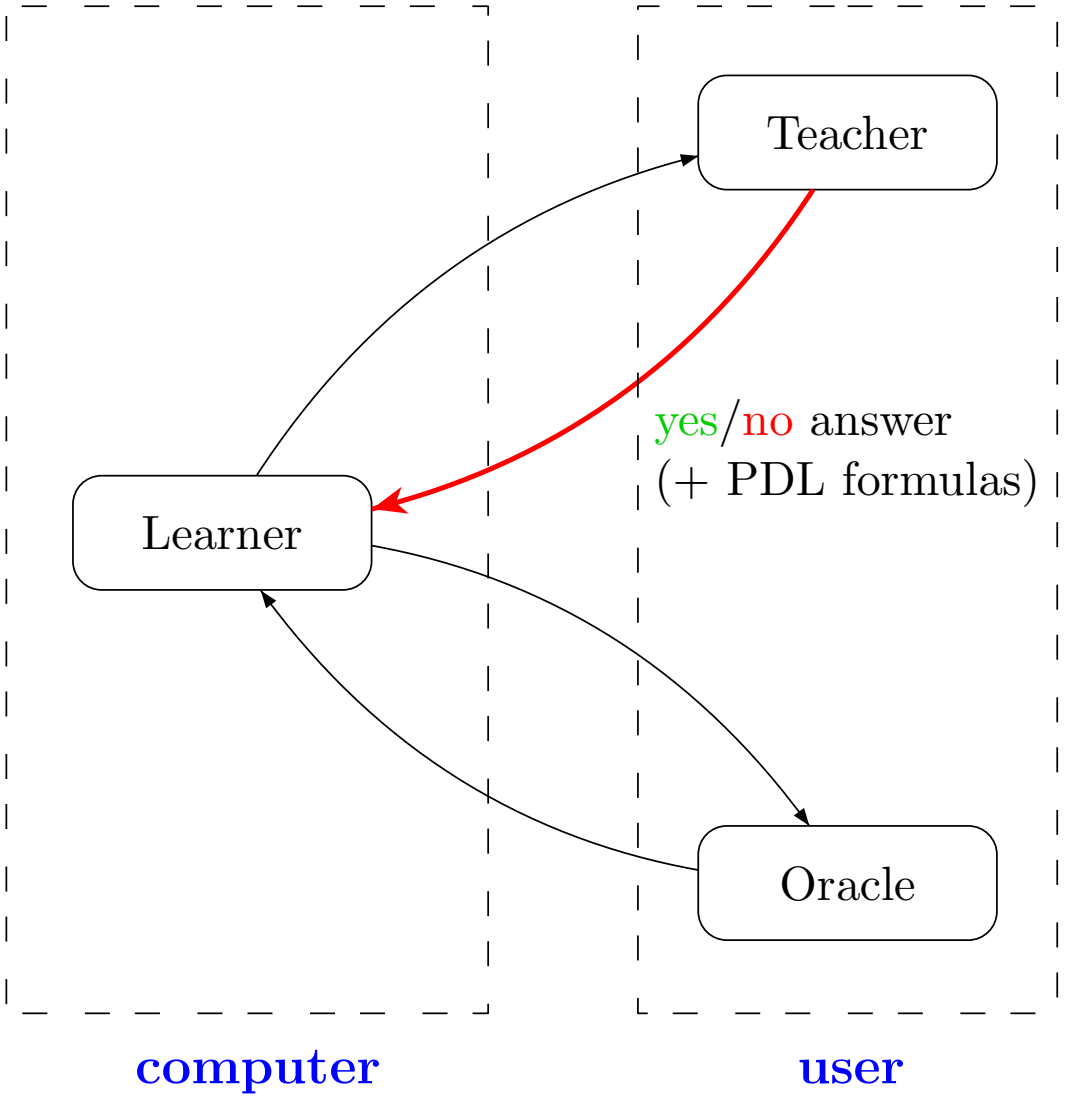
Angluin's algorithm: extended



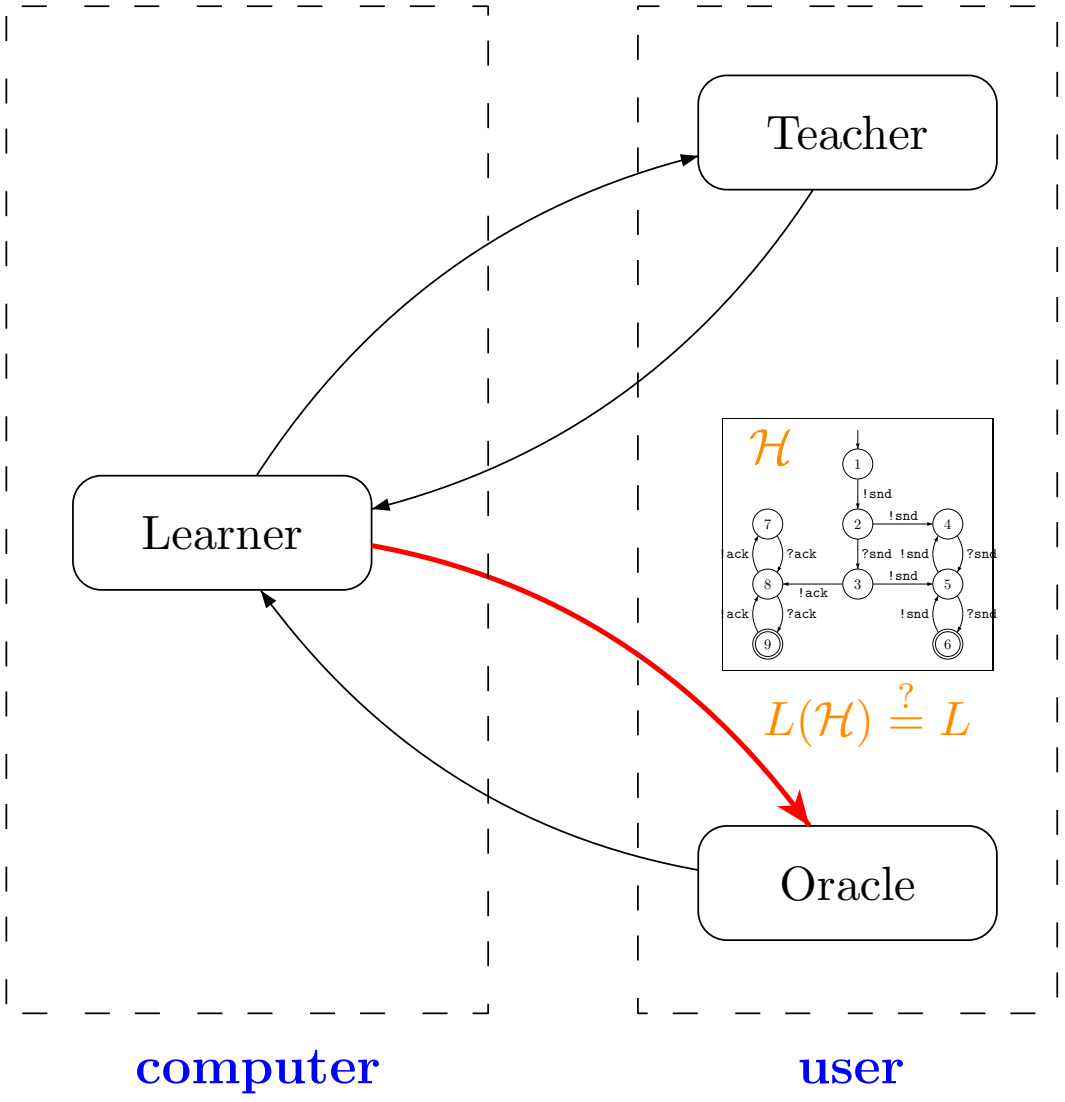
Angluin's algorithm: extended



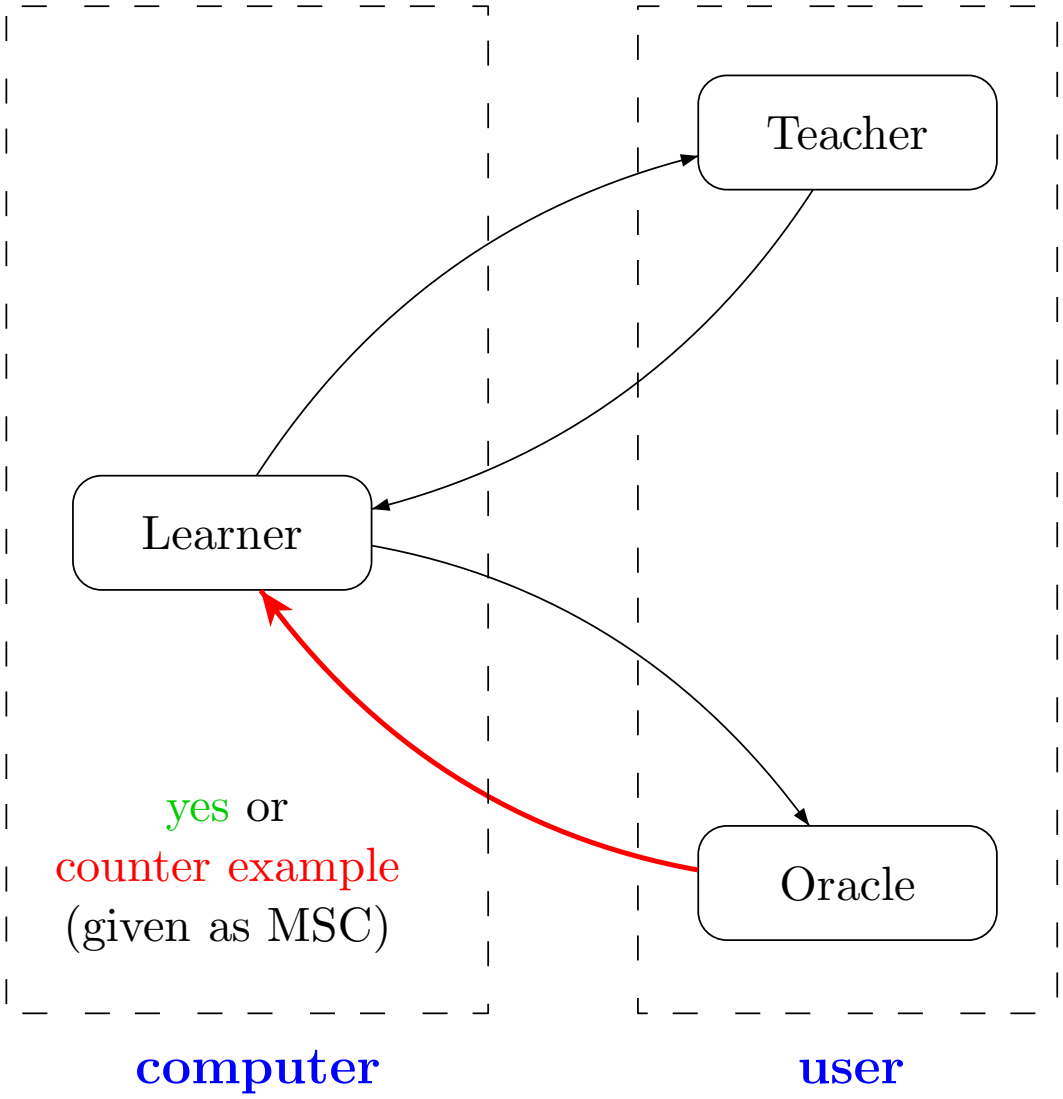
Angluin's algorithm: extended



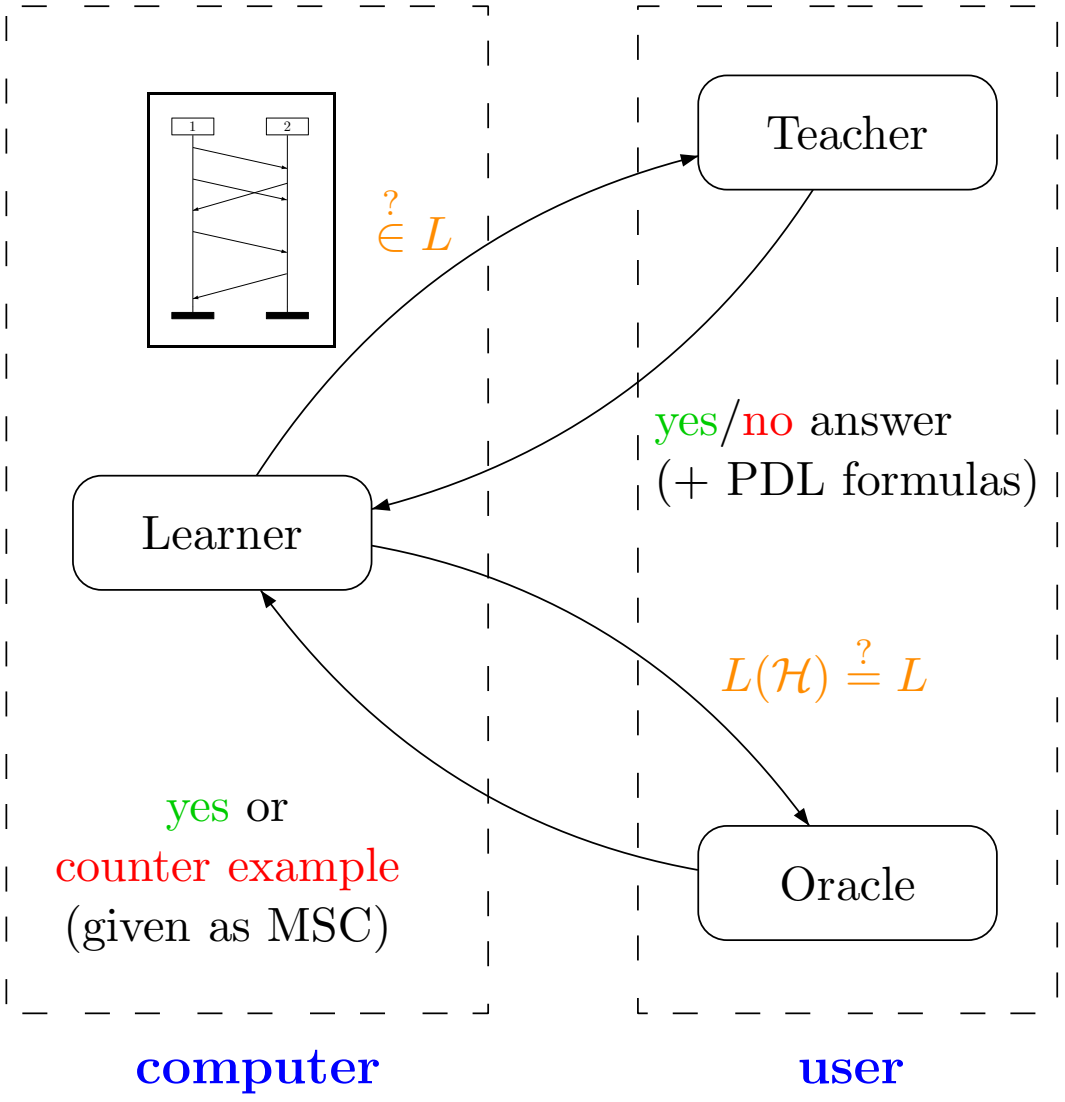
Angluin's algorithm: extended



Angluin's algorithm: extended



Angluin's algorithm: extended



Presentation outline

Smyle

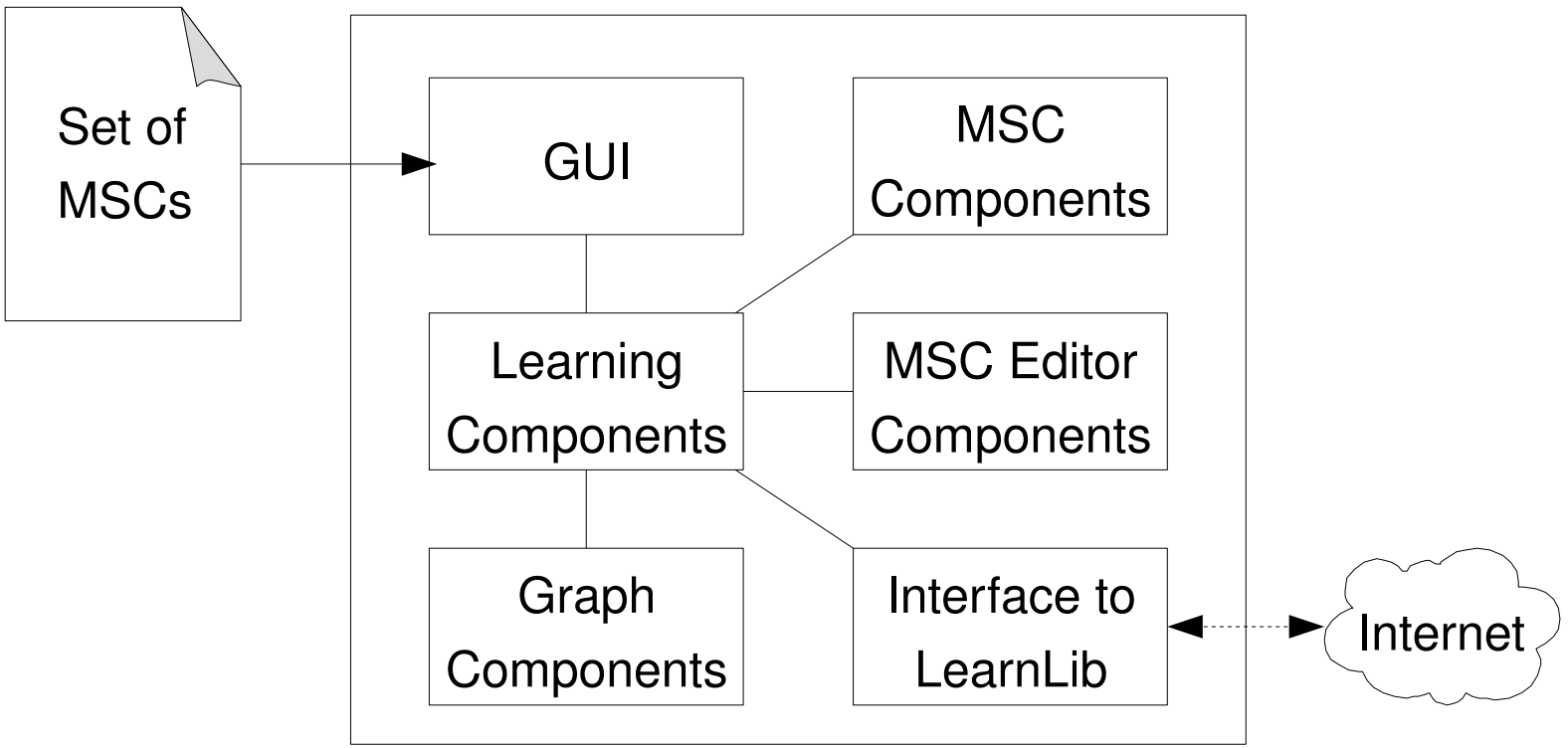
Technical details

- written in Java 1.5 (available for Java 1.5 and 1.6)
- approx. 21.000 lines of code (not including MSC Editor)

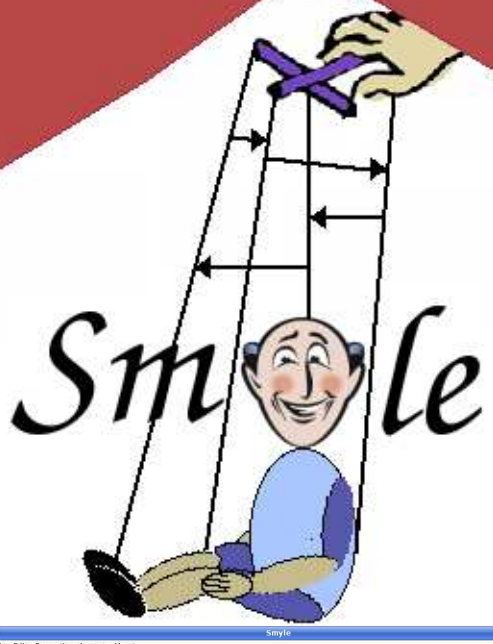
Libraries

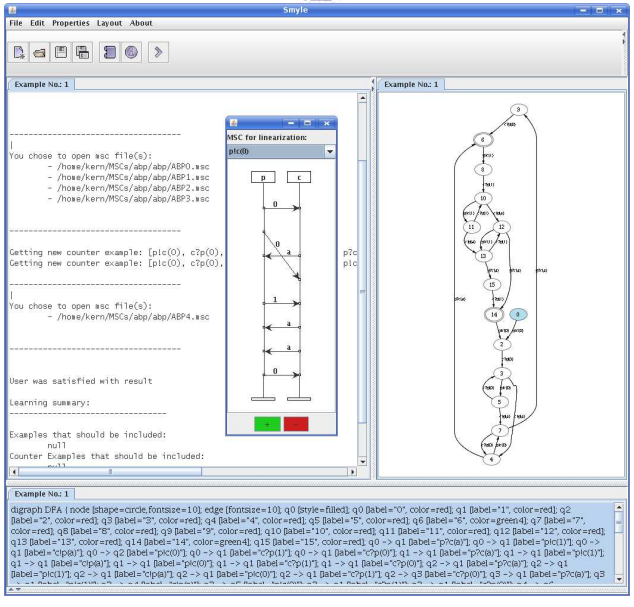
- currently uses `LearnLib` library (Dortmund, Prof. Dr. B. Steffen)
- `GRAPPA` (visualization of automata)
- `JGraph` (visualization of MSCs)
- `MSC2000` (Parser for MSC documents)

Package Overview



Tool Demo


Smyle


Smyle
File Edit Properties Layout About
Example No.: 1
MSC for linearization:
plc(0)
You chose to open msc file(s):
- /home/kern/MSCs/abp/abp/ABP0.msc
- /home/kern/MSCs/abp/abp/ABP1.msc
- /home/kern/MSCs/abp/abp/ABP2.msc
- /home/kern/MSCs/abp/abp/ABP3.msc
Getting new counter example: [plc(0), c?p(0),
Getting new counter example: [plc(0), c?p(0),
You chose to open msc file(s):
- /home/kern/MSCs/abp/abp/ABP4.msc
User was satisfied with result
Learning summary:
Examples that should be included:
null
Counter Examples that should be included:

Some protocols learned by Smyle

Protocol	#membership queries	#user queries	#states of $\mathcal{H}$	memory consumption	learning setup
part of *USB 1.1*	1.999	12	9	$\sim$ 3.5 MB	$\exists 2$
continuous update	4.747	36	8	$\sim$ 4.5 MB	$\exists 1$
simple negotiation	8.267	46	9	$\sim$ 5.5 MB	$\exists 1$
alternating bit	11.164	64 (24)	15	$\sim$ 8.5 MB	$\exists 1$
alternating bit	91.710	145 (64)	25	$\sim$ 16.5 MB	$\exists 2$
alternating bit	242891	199 (90)	31	$\sim$ 22 MB	$\exists 3$
leader election (1 round)	35.974	44	13	$\sim$ 12.0 MB	$\forall 1$
leader election ($>$ 1 rounds)	584.739	495	17	$\sim$ 85.0 MB	$\forall 1$

Protocol	#user queries	#reduced user queries $(*)$
alternating bit ($\exists 1$)	64	24
alternating bit ($\exists 2$)	145	64
alternating bit ($\exists 3$)	199	90

$(*)$ using 4 PDL formulas

1. "p receives two subsequent a"
2. "p receives a subsequent 1 and 0"
3. "q has to receive a 0 at first"
4. "p must receive an a at last"

Presentation outline

Outlook

Current State

- learning of several classes of MPA
- integrated simulation component
- integrated MSC editor
- currently only hard-coded formulas

Future Work

- full integration of PDL
- more efficient partial order treatment
- discover new/broader classes of learnable MPA
- dealing with *don't know* answers
- case studies
- ...

S(ynthesizing) M(odels) (b)Y L(earning from) E(xamples)

http://www.smyle-tool.org

Thank you for your attention!