Dynamic program analysis and Mining of software specifications

Pietro Braione

pietro.braione@unimib.it

(course material by Leonardo Mariani)

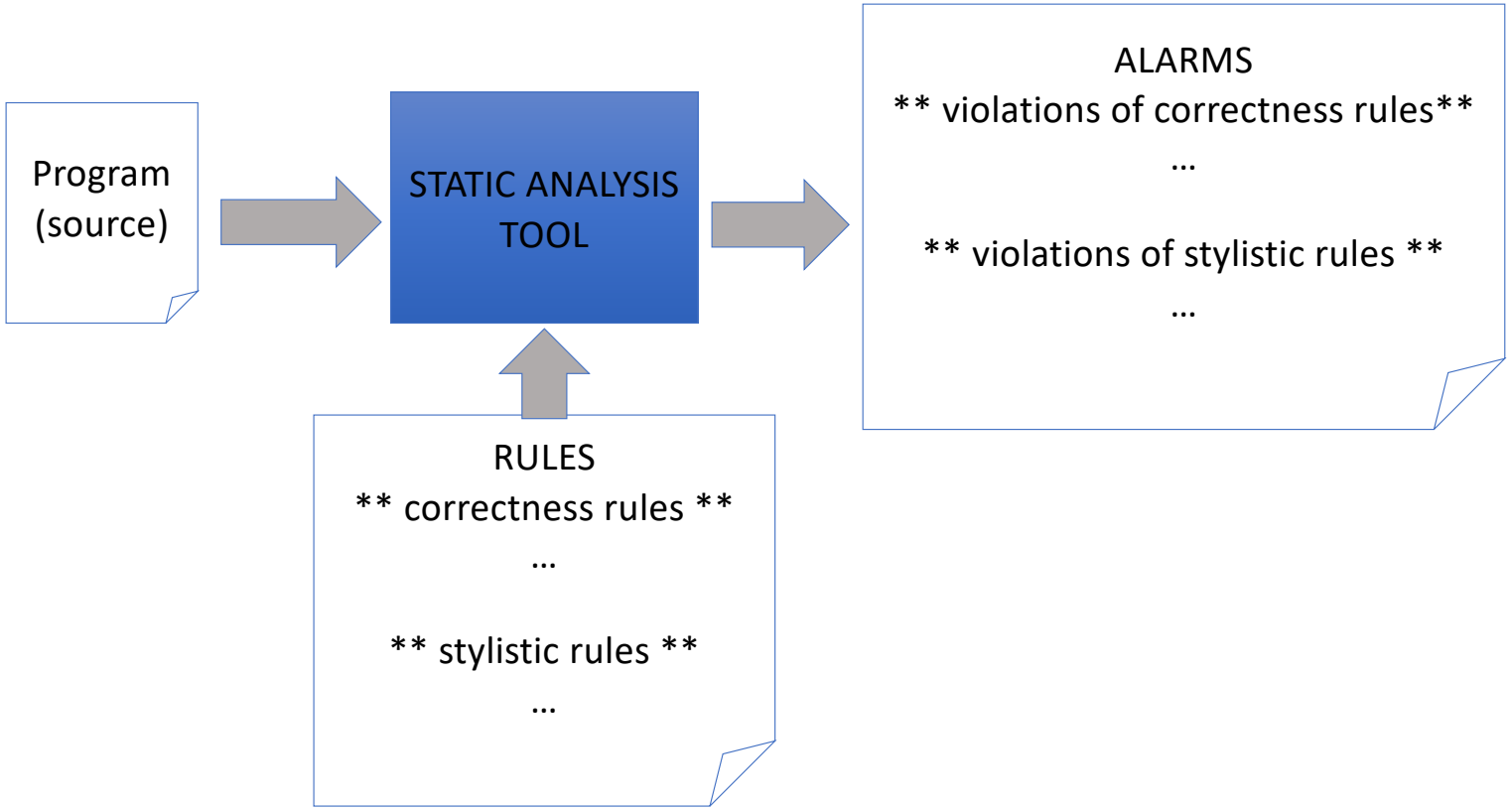
Static vs dynamic program analysis

Static vs dynamic analysis

- Static analysis: Examine program source code
  - Examine the complete execution space
  - But may lead to **false alarms**
- Dynamic analysis: Exa program execution traces
  - **No inf**
  - But **ca**

```
PowerManager::PowerManager(IMsgSender* msgSender)
  : msgSender_(msgSender) { }

void PowerManager::SignalShutdown()
{
   msgSender_->sendMsg("shutdown()");
}
```

Example: Rule-based static analysis

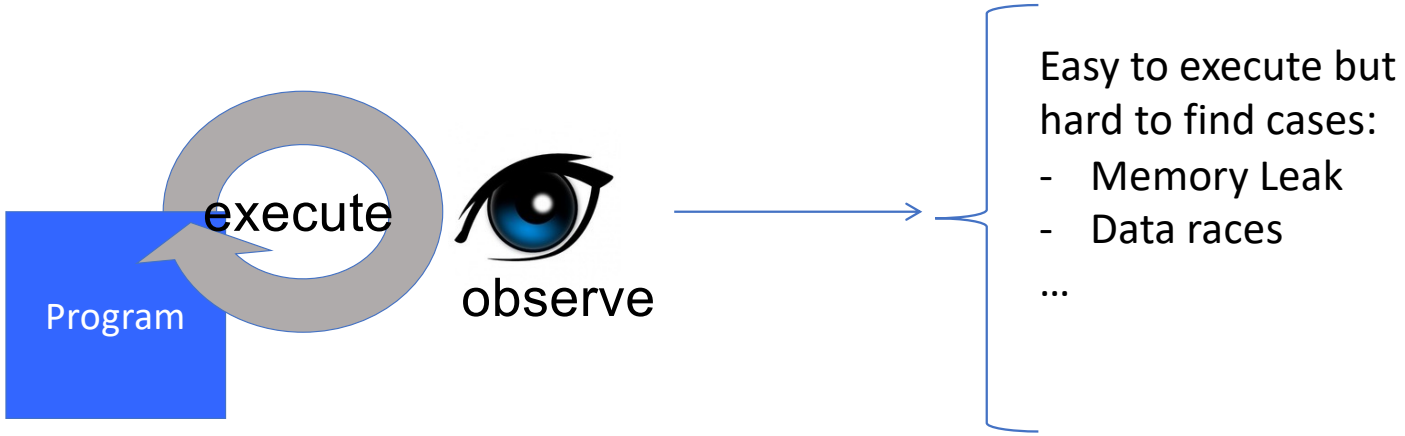


In some domains the code must comply to a standard set of rules
e.g., MISRA in the automotive domain

Rule-based static analysis: Some tools

- Java:
  - PMD
  - Checkstyle
  - Android Lint
- C/C++
  - Cppcheck
  - clang-tidy
  - vera++
  - Google cpplint

Dynamic analysis



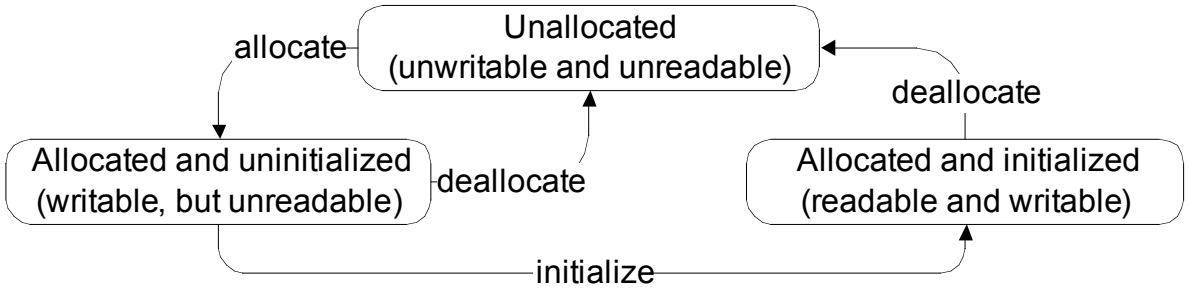
Do you see any fault?

```
void f(void) {
        int* x = malloc(10 * s
        x[10] = 0;
}
```

Heap block overrun
- sporadic failures

Memory leak
- Slow down and crashes in long running executions

Dynamic memory analysis



- Instrument program to trace memory access
- At runtime:
  - record the state of each memory location
  - detect accesses incompatible with the current state
    - attempts to access unallocated memory
    - read from uninitialized memory locations
  - array bounds violations:
    - add memory locations with state *unallocated* before and after each array
    - attempts to access these locations are detected immediately

Data race

```
#include <thread>
#include <iostream>
#include <vector>

unsigned const increment_count = 2000000;
unsigned const thread_count = 2;

unsigned i = 0;

void func() {
   for (unsigned c = 0; c < increment_count; ++c) {
      ++i;
   }
}
```

```
int main() {
   std::vector<std::thread> threads;
   for (unsigned c = 0; c < thread_count; ++c) {
      threads.push_back(std::thread(func));
   }
   for (unsigned c = 0; c < threads.size(); ++c) {
      threads[c].join();
   }

   std::cout << thread_count << " threads, final i=” << i;
   std::cout << ", increments=" << (thread_count * increment_count);
   std::cout << std::endl;
}
```

What is the output of this program?

2 threads, final i=2976075, increments=4000000
2 threads, final i=3097899, increments=4000000
2 threads, final i=4000000, increments=4000000
2 threads, final i=3441342, increments=4000000
2 threads, final i=2942251, increments=4000000

Dynamic lockset analysis

- Lockset discipline: set of rules to prevent data races
- Easiest discipline: Every variable shared between threads must be protected by a mutual exclusion lock
- Dynamic lockset analysis detects violation of the locking discipline
  - Identify set of mutual exclusion locks held by threads when accessing each shared variable
  - INIT: each shared variable is associated with all available locks
  - RUN: when a thread accesses a shared variable, intersect current set of candidate locks with locks held by the thread
  - END: set of locks after executing a test = set of locks always held by threads accessing that variable; empty set for v = no lock consistently protects v

Dynamic lockset analysis: Example

Thread	Statement	Lock held by thread	Lockset of x	
			{lck1, lck2}	INIT: all locks

Dynamic lockset analysis: Example

Thread	Statement	Lock held by thread	Lockset of x	
			{lck1, lck2}	INIT: all locks
A	lock(lck1);	{lck1}		lck1 held

Dynamic lockset analysis: Example

Thread	Statement	Lock held by thread	Lockset of x	
			{lck1, lck2}	INIT: all locks
A	lock(lck1);	{lck1}		lck1 held
	x = 1;		{lck1}	Intersect w/lock held

Dynamic lockset analysis: Example

Thread	Statement	Lock held by thread	Lockset of x	
			{lck1, lck2}	INIT: all locks
A	lock(lck1);	{lck1}		lck1 held
	x = 1;		{lck1}	Intersect w/lock held
	release(lck1);	{}		lck1 released

Dynamic lockset analysis: Example

Thread	Statement	Lock held by thread	Lockset of x	
			{lck1, lck2}	INIT: all locks
A	lock(lck1);	{lck1}		lck1 held
	x = 1;		{lck1}	Intersect w/lock held
	release(lck1);	{}		lck1 released
B	lock(lck2);	{lck2}		lck2 held

Dynamic lockset analysis: Example

Thread	Statement	Lock held by thread	Lockset of x	
			{lck1, lck2}	INIT: all locks
A	lock(lck1);	{lck1}		lck1 held
	x = 1;		{lck1}	Intersect w/lock held
	release(lck1);	{}		lck1 released
B	lock(lck2);	{lck2}		lck2 held
	x = 2;		{}	Intersect w/lock held

Dynamic lockset analysis: Example

Thread	Statement	Lock held by thread	Lockset of x	
			{lck1, lck2}	INIT: all locks
A	lock(lck1);	{lck1}		lck1 held
	x = 1;		{lck1}	Intersect w/lock held
	release(lck1);	{}		lck1 released
B	lock(lck2);	{lck2}		lck2 held
	x = 2;		{}	Intersect w/lock held
	release(lck2);			

Dynamic lockset analysis: Example

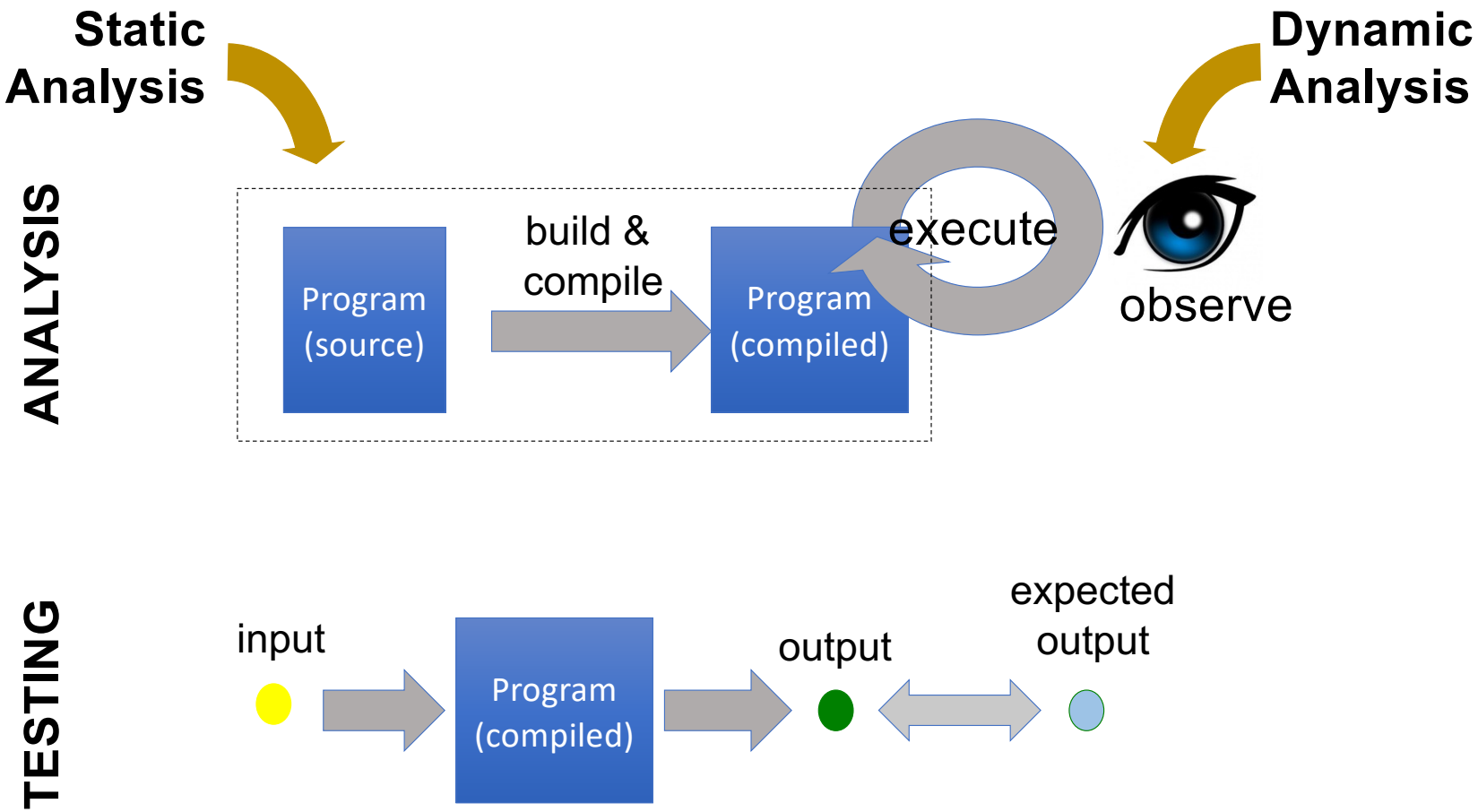
Thread	Statement	Lock held by thread	Lockset of x	
			{lck1, lck2}	INIT: all locks
A	lock(lck1);	{lck1}		lck1 held
	x = 1;		{lck1}	Intersect w/lock held
	release(lck1);	{}		lck1 released
B	lock(lck2);	{lck2}		lck2 held
	x = 2;		{}	Intersect w/lock held
	release(lck2);			

Empty lockset, potential race

Some tools

- Dynamic memory analysis:
  - Valgrind Memcheck
  - Google AddressSanitizer, LeakSanitizer, MemorySanitizer
  - Dmalloc
  - UNICOM (was Rational) PurifyPlus [commercial]
  - MicroFocus BoundsChecker [commercial]
  - Parasoft Insure++ [commercial]
- Dynamic thread analysis:
  - Valgrind Helgrind and DRD
  - Google ThreadSanitizer
  - Intel Inspector [commercial]

Take home


Static Analysis
Dynamic Analysis
ANALYSIS
Program (source)
build & compile
Program (compiled)
execute
observe
TESTING
input
Program (compiled)
output
expected output

Mining of software specifications: An introduction

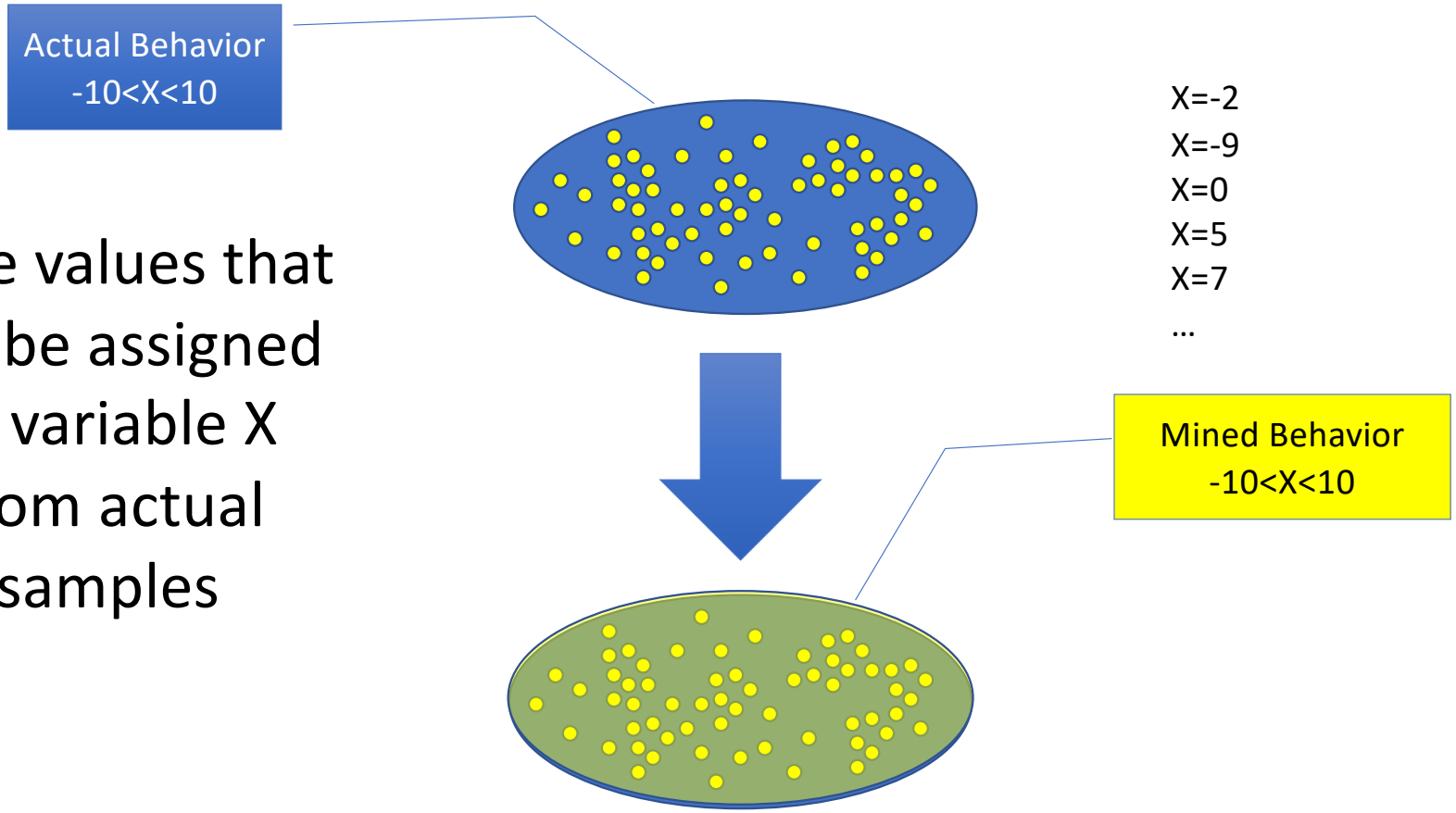
Analysis of Software Behaviors

Revealing, *Analyzing*, and
Detecting Software Failures

(semi-)automatically, when
no specification is available

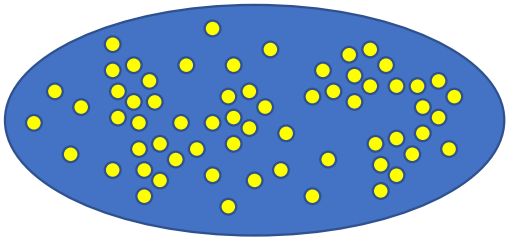
Specification Mining =
Learn specifications from
actual executions

Specification mining

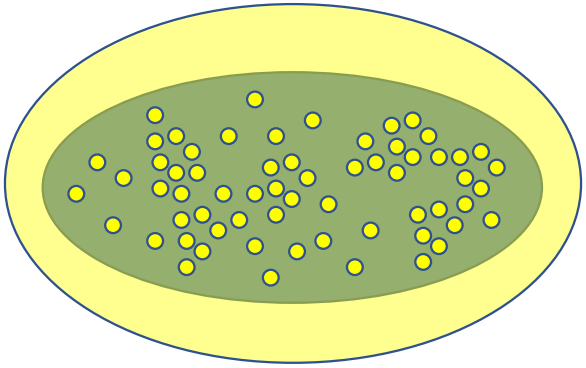


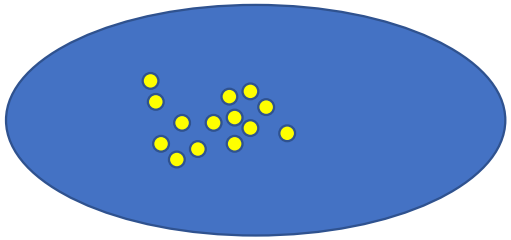
mine values that can be assigned to variable X from actual samples

Specification mining is imprecise

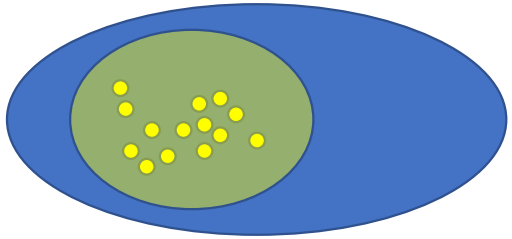


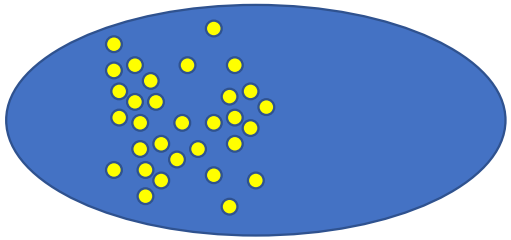
Over-Generalization



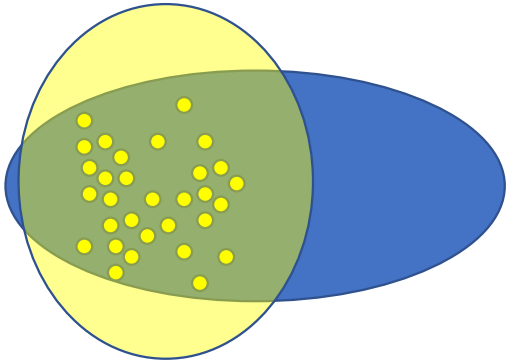


Under-Generalization





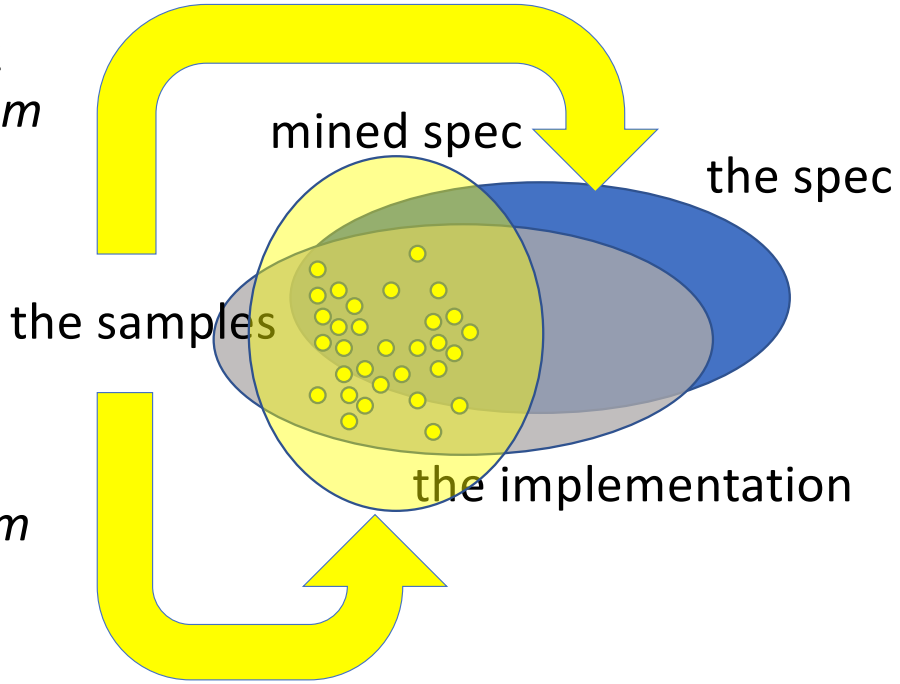
Over- and Under-Generalization



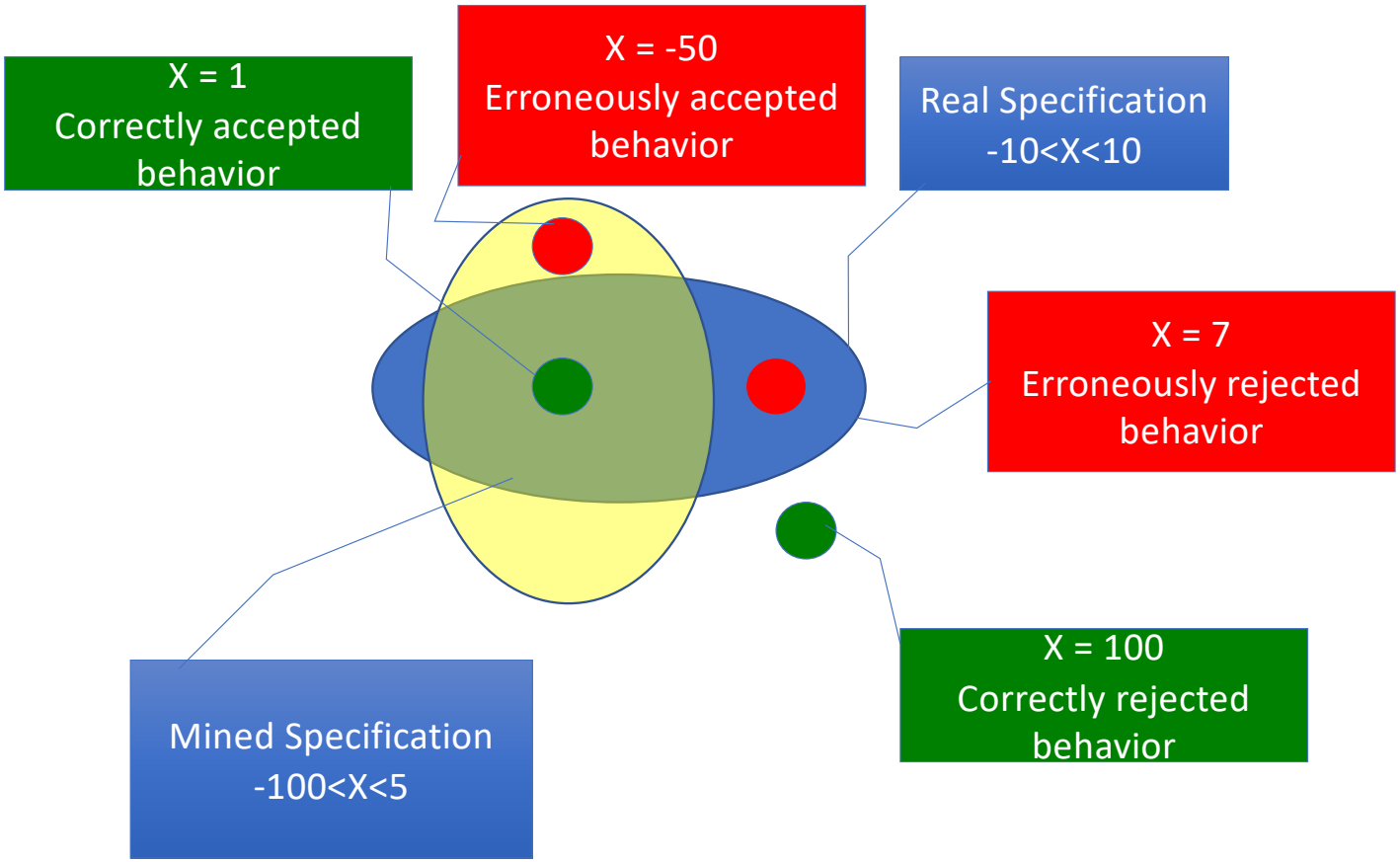
Specification mining is hard

What <u>we would like to do</u>: derive the spec from the samples

What <u>we do</u>: derive an approximated spec from the samples



Models used as specifications



Gap

Models mined from *samples* produced by the *actual implementation*

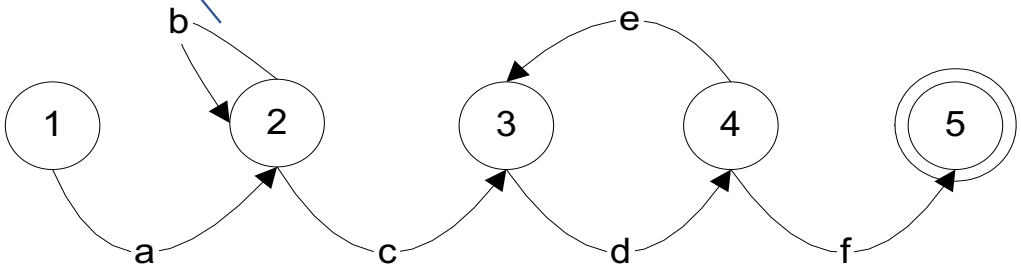
VS

Intended behavior

We need to address this gap when using the models

Model types

Events, total order



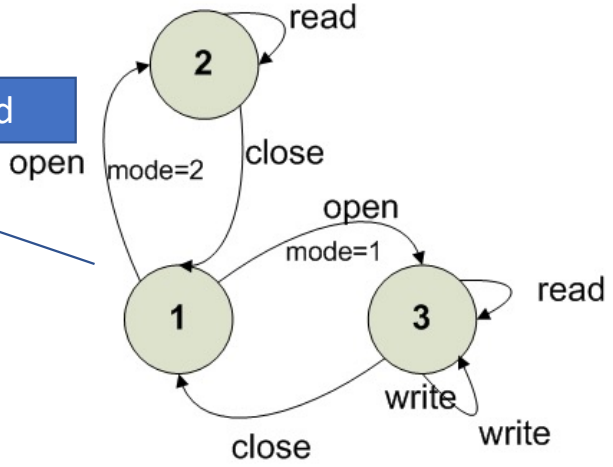
Data

$x > 0$

Events, partial order

open => close

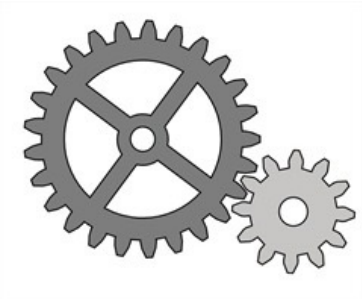
Combined



Models of events, total order

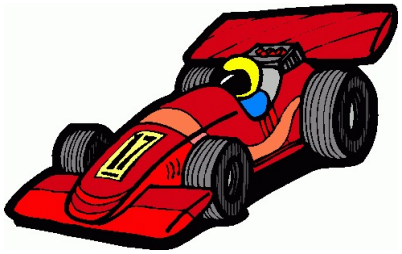
Procedural vs. Declarative

construction mechanism



inference

...



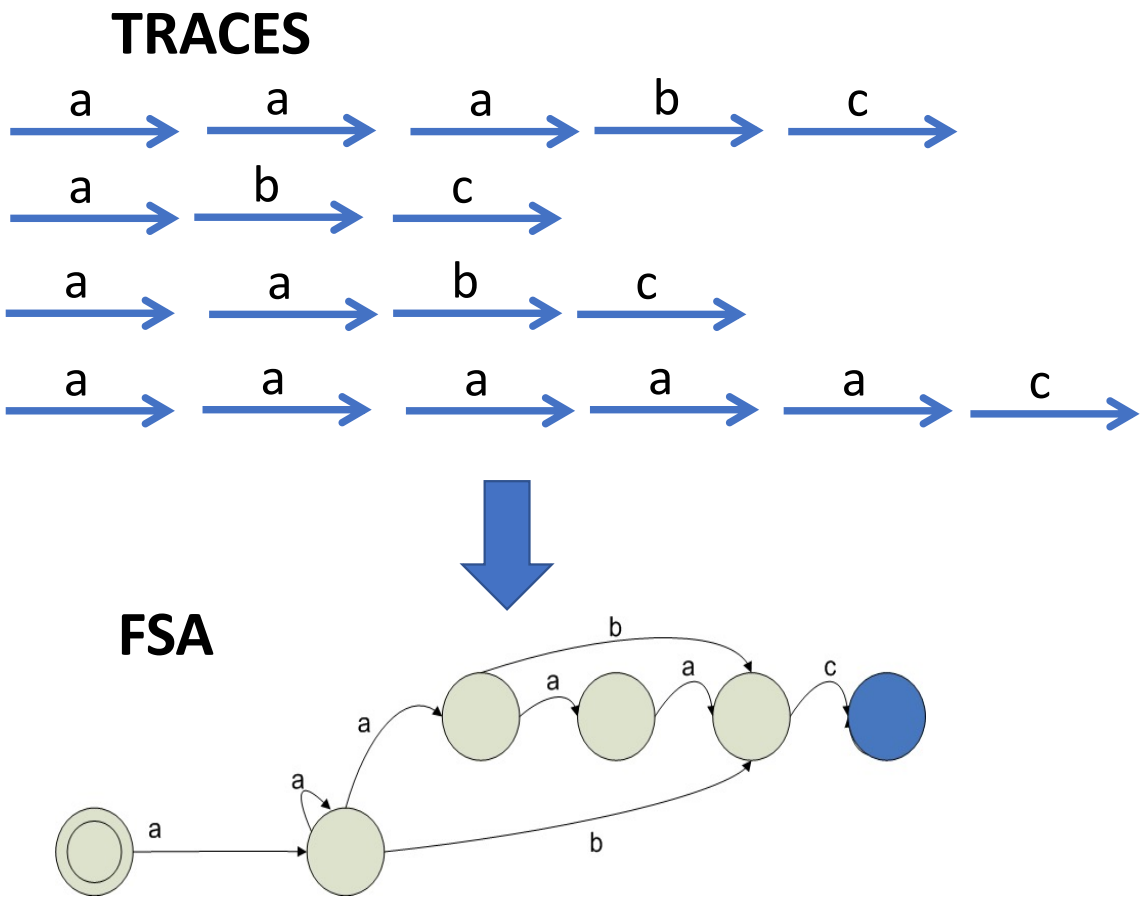
properties of the result

inference

Procedural approaches

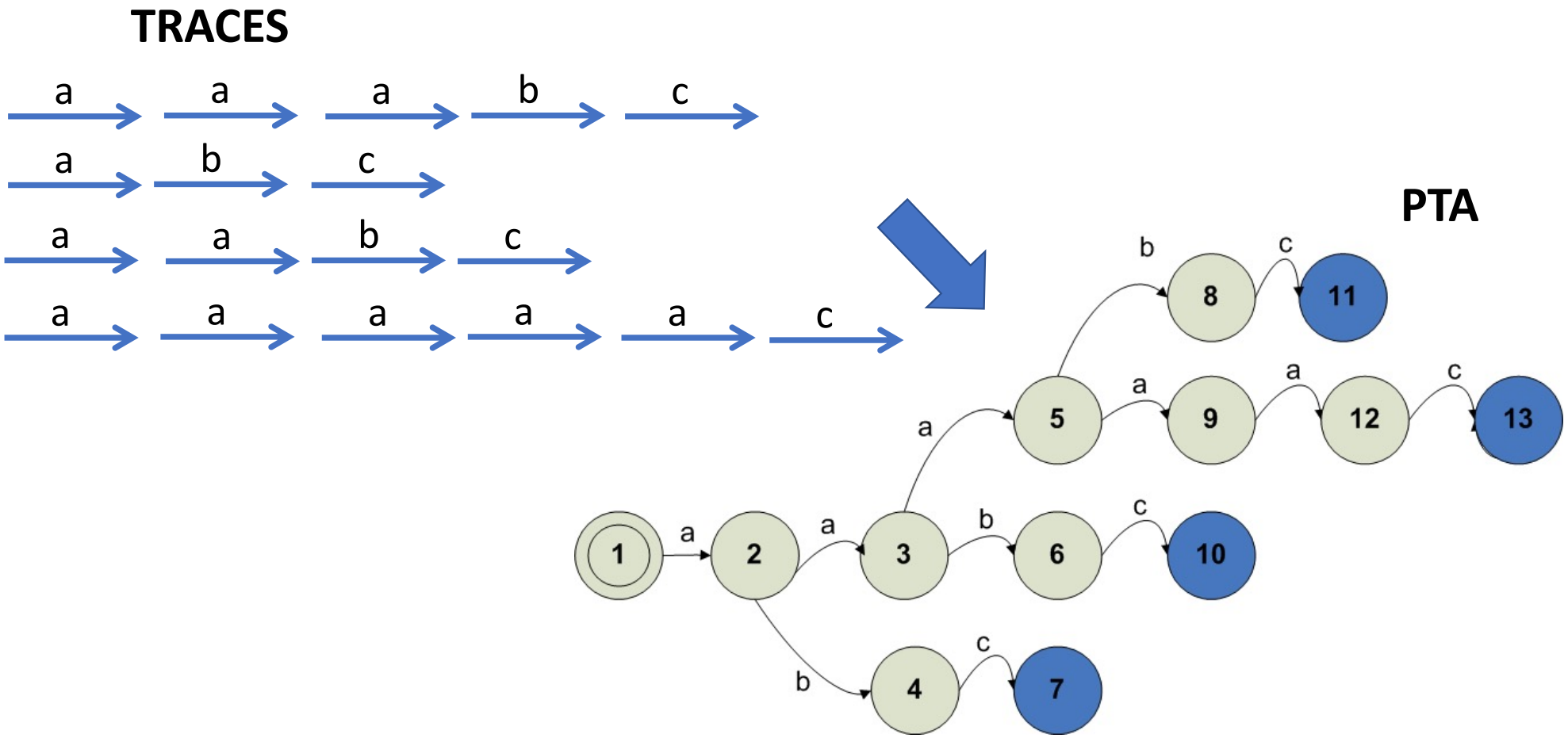
- Trace-based mining
  - State-based merging
  - Behavior-based merging
- State-based mining

k-Tail

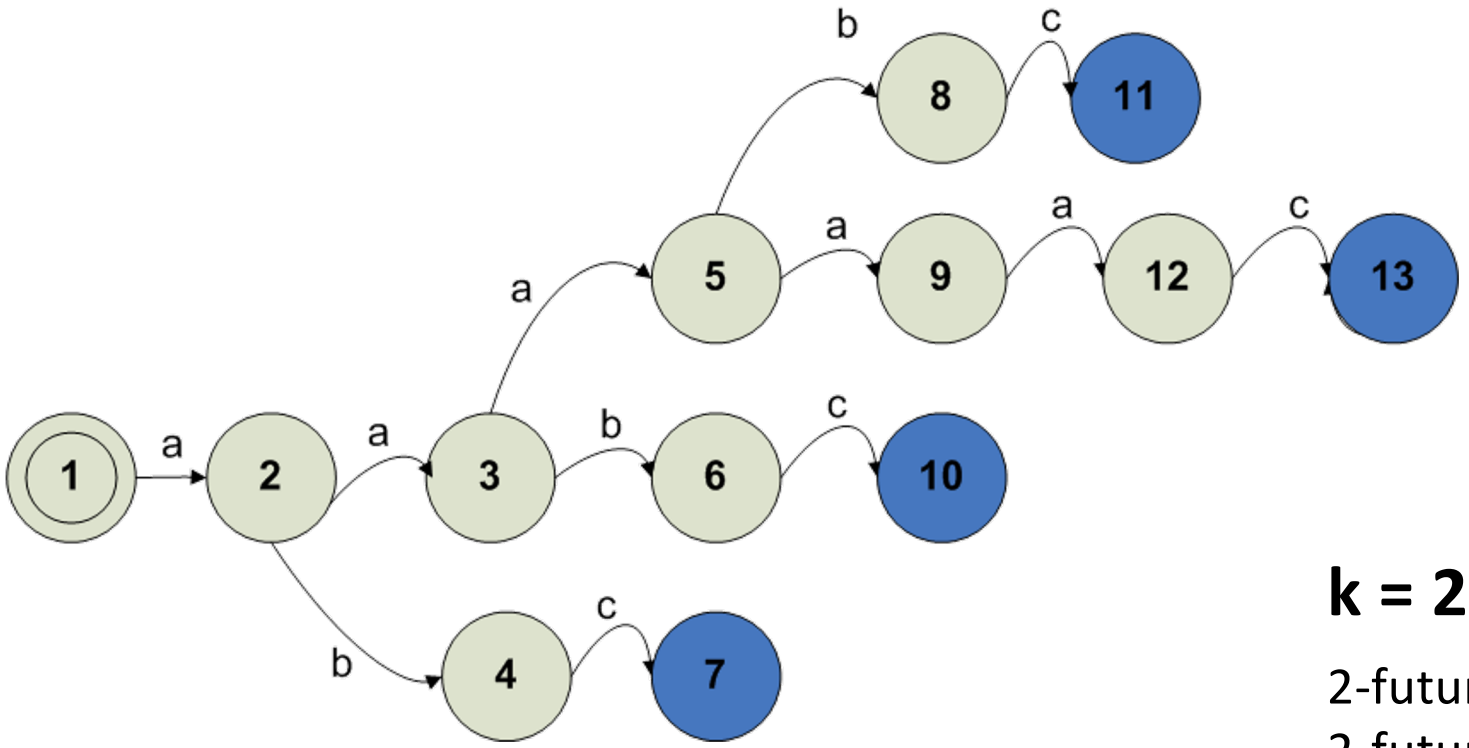


[Biermann and Feldman. On the synthesis of finite state machines from samples of their behavior. IEEE ToC, 1972]

The Prefix Tree Acceptor (PTA)



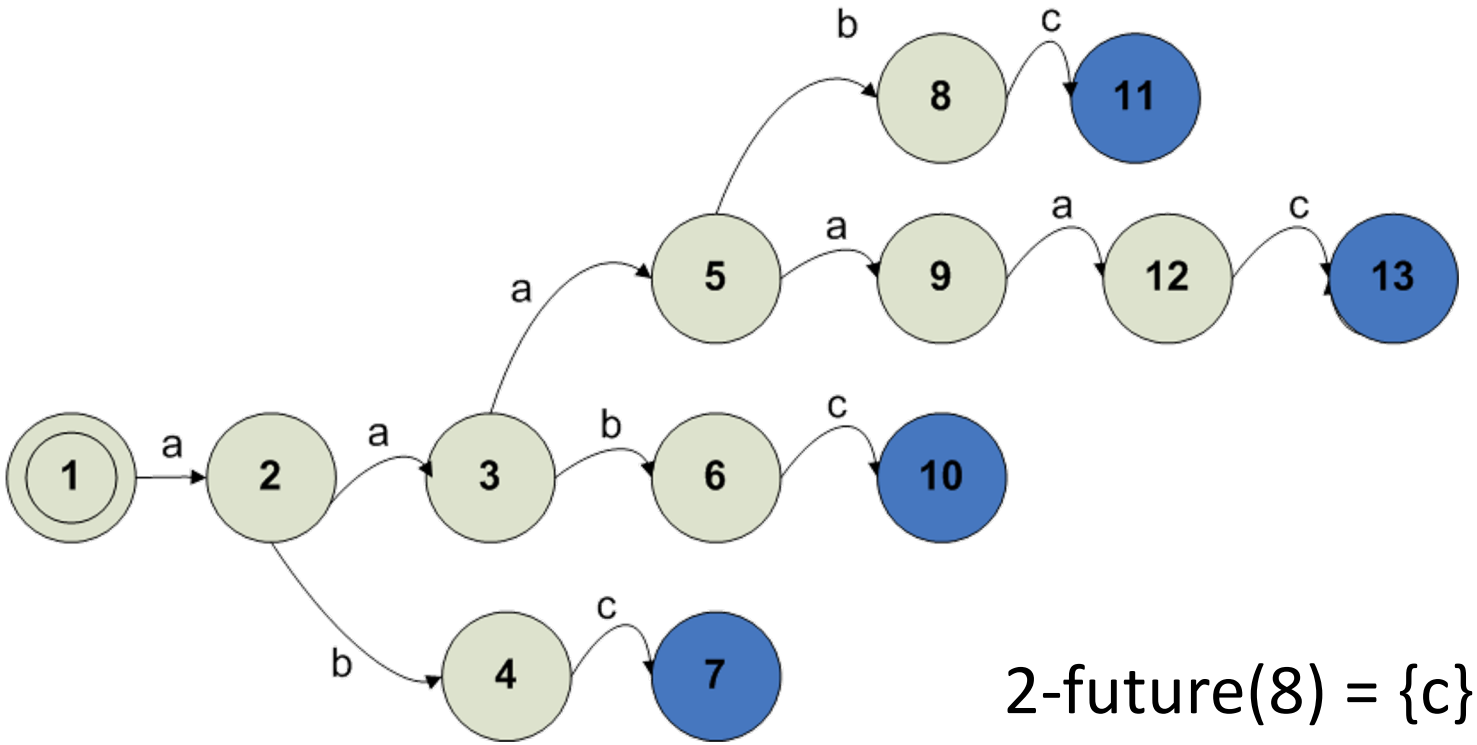
From the PTA to the FSA



k = 2 => 2-FUTURES

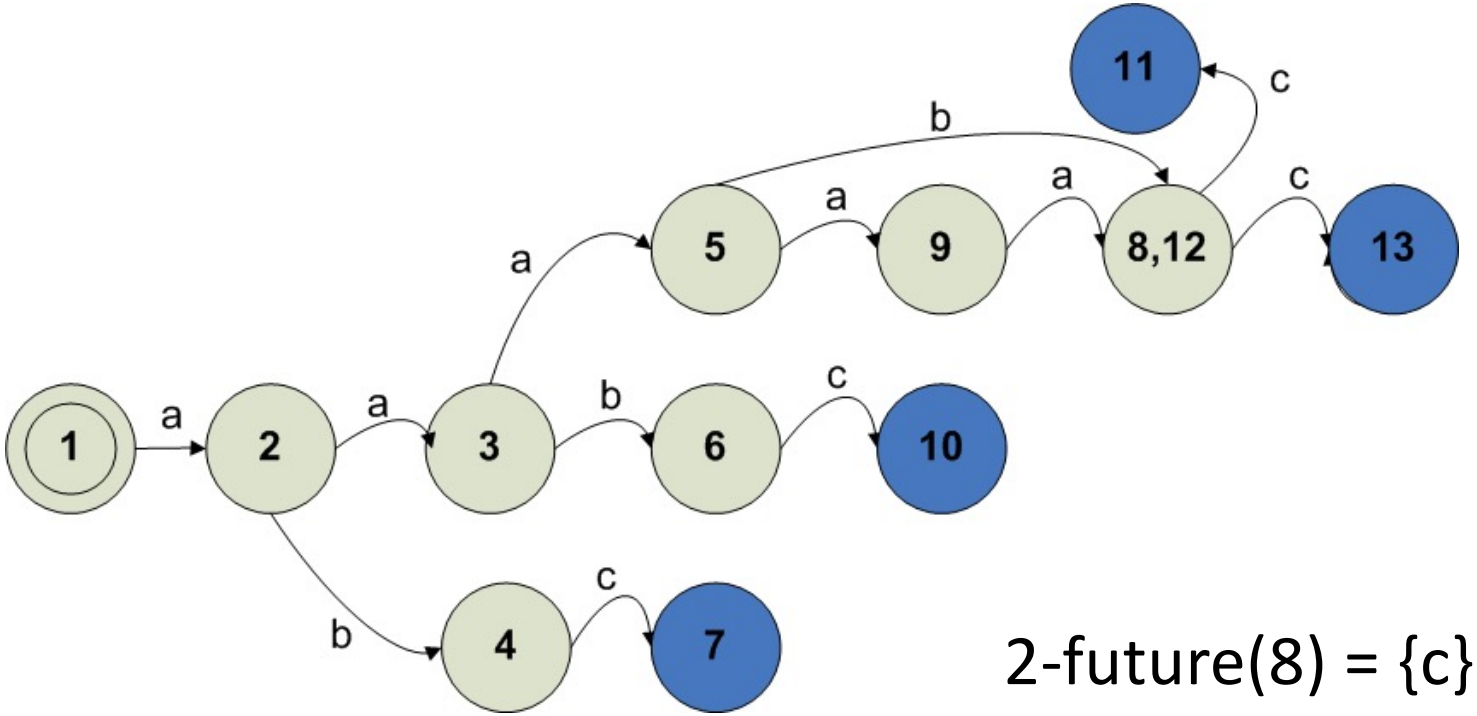
2-future(2) = {aa,ab,bc}
2-future(5) = {aa, bc}
2-future(11) = {}
2-future(8) = {c}
…

From the PTA to the FSA



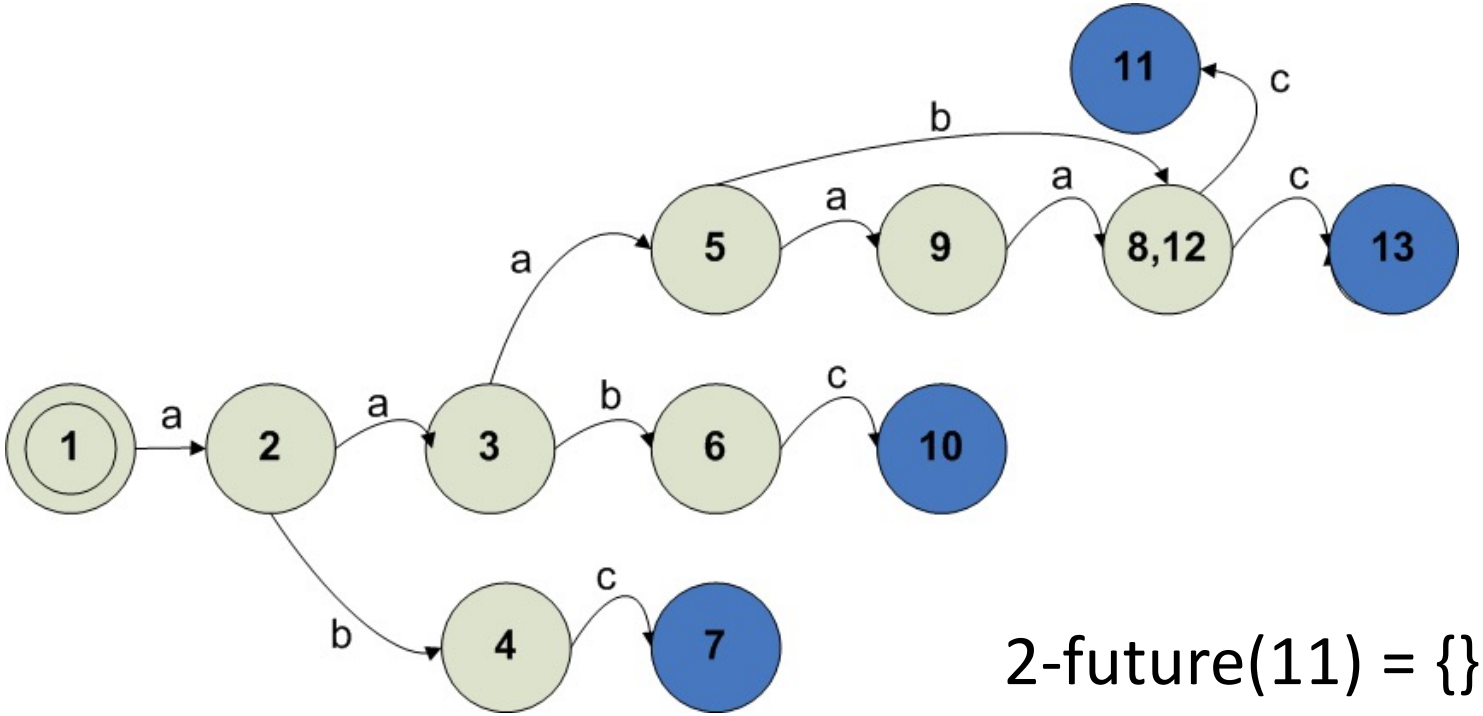
2-future(8) = {c} = 2-future(12)

From the PTA to the FSA



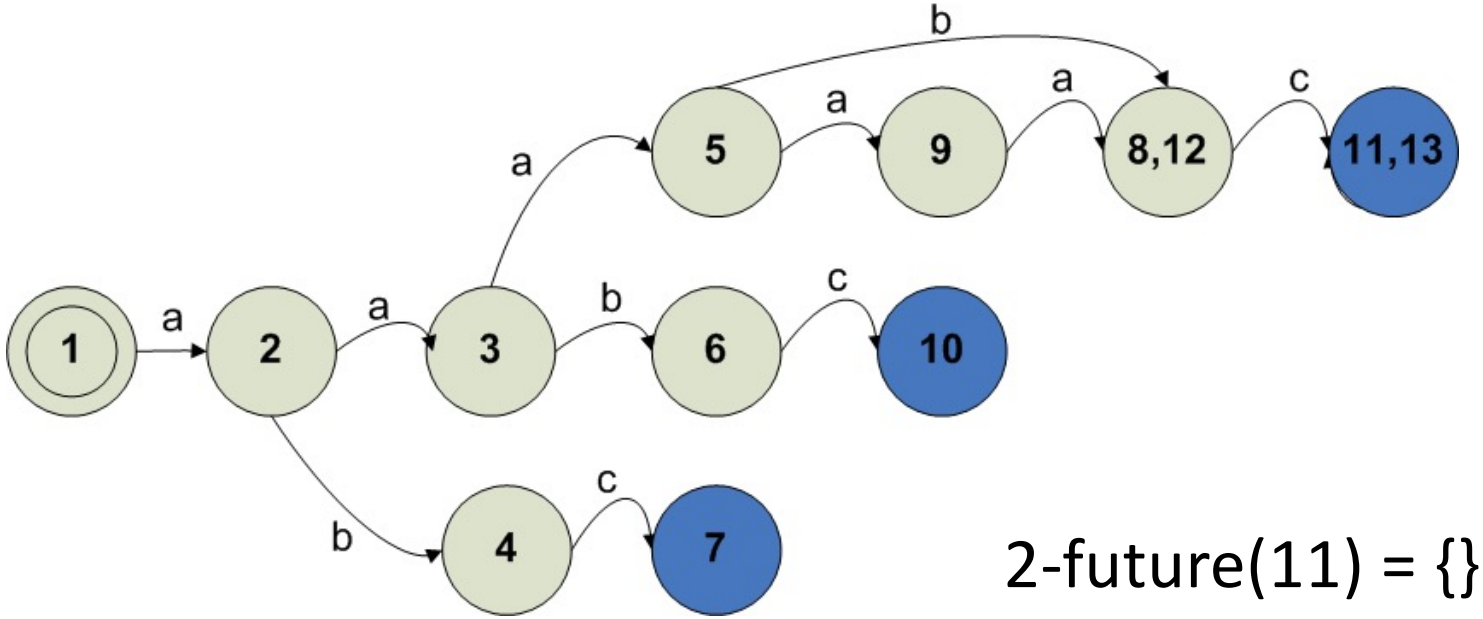
2-future(8) = {c} = 2-future(12)

From the PTA to the FSA



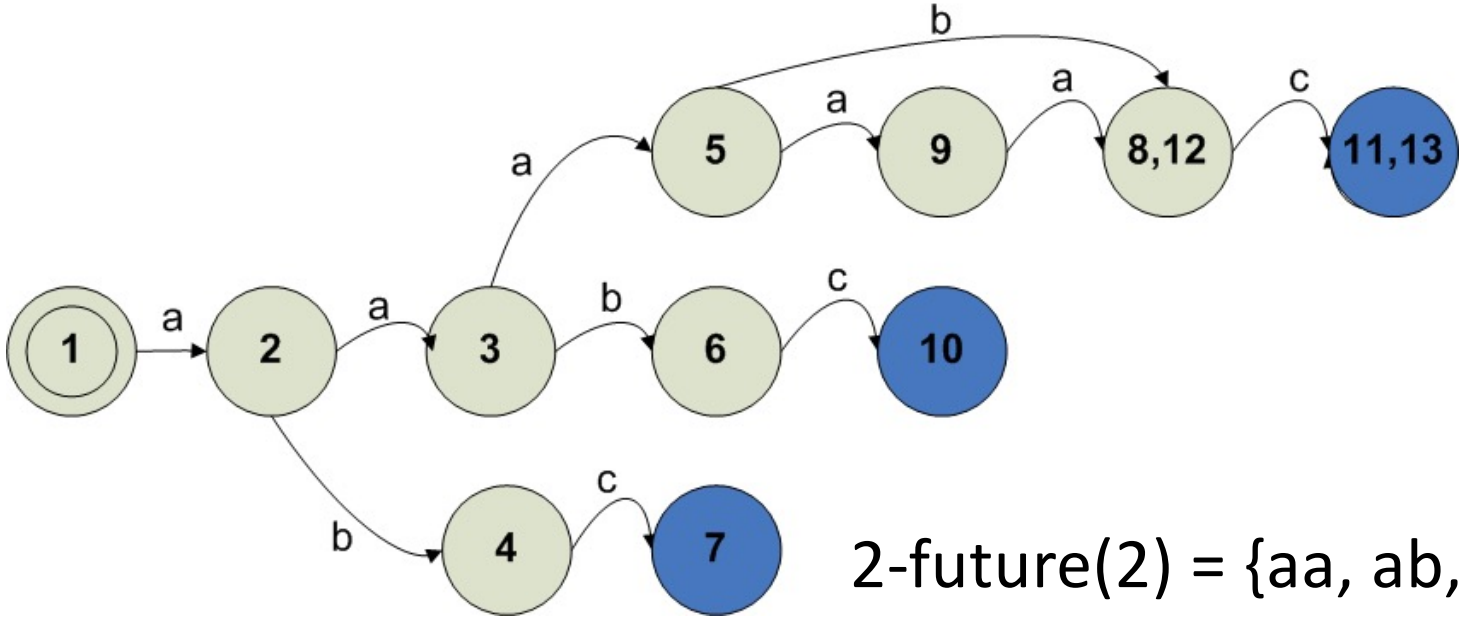
2-future(11) = {} = 2-future(13)

From the PTA to the FSA



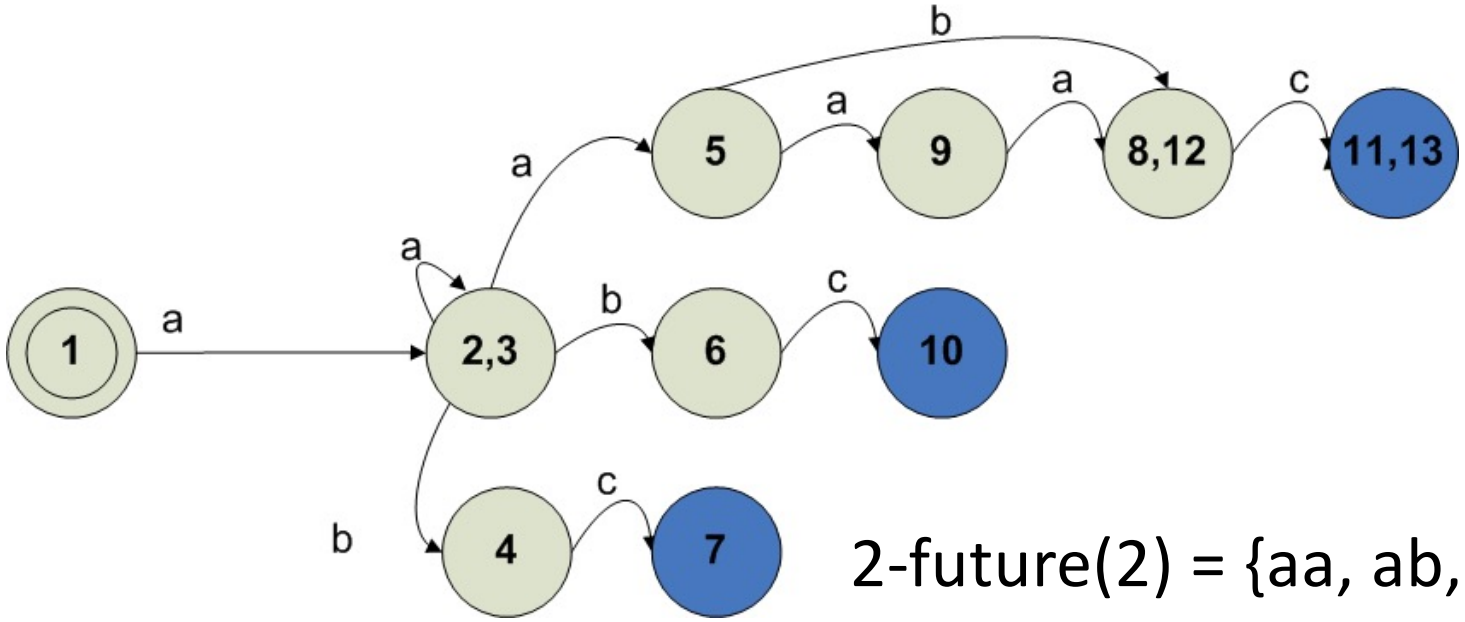
2-future(11) = {} = 2-future(13)

From the PTA to the FSA



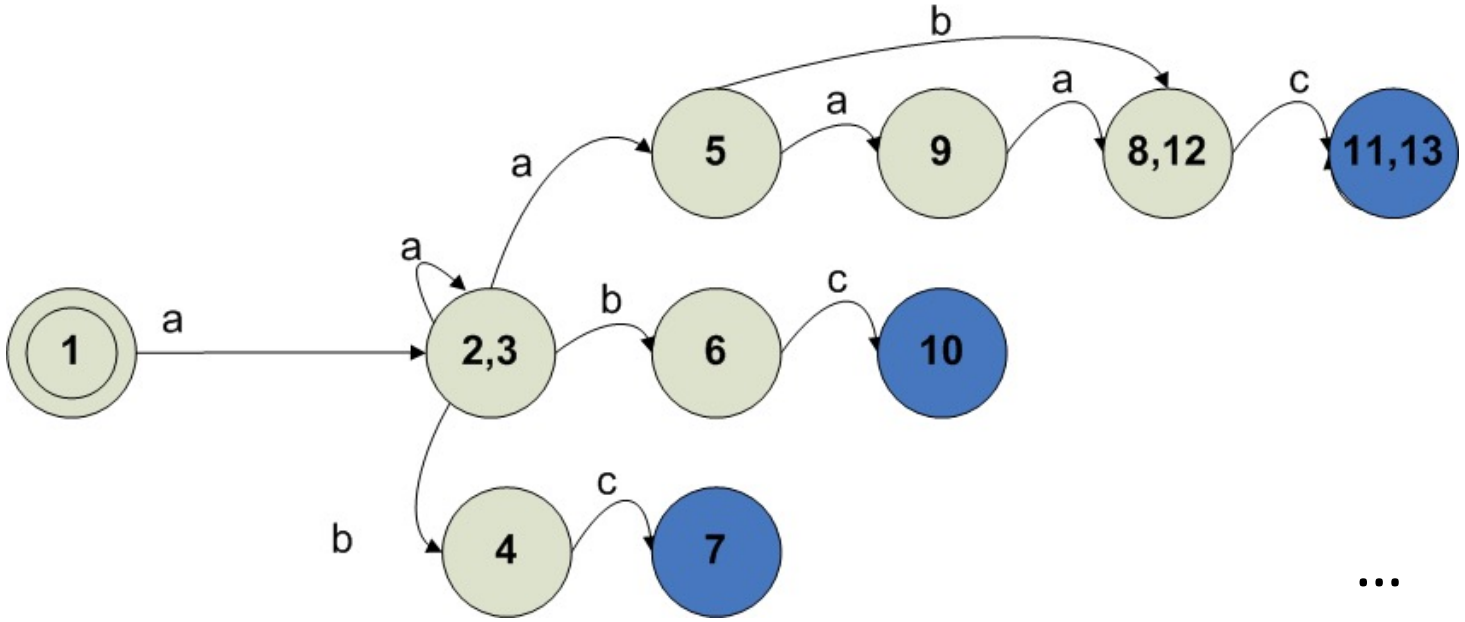
2-future(2) = {aa, ab, bc} = 2-future(3)

From the PTA to the FSA

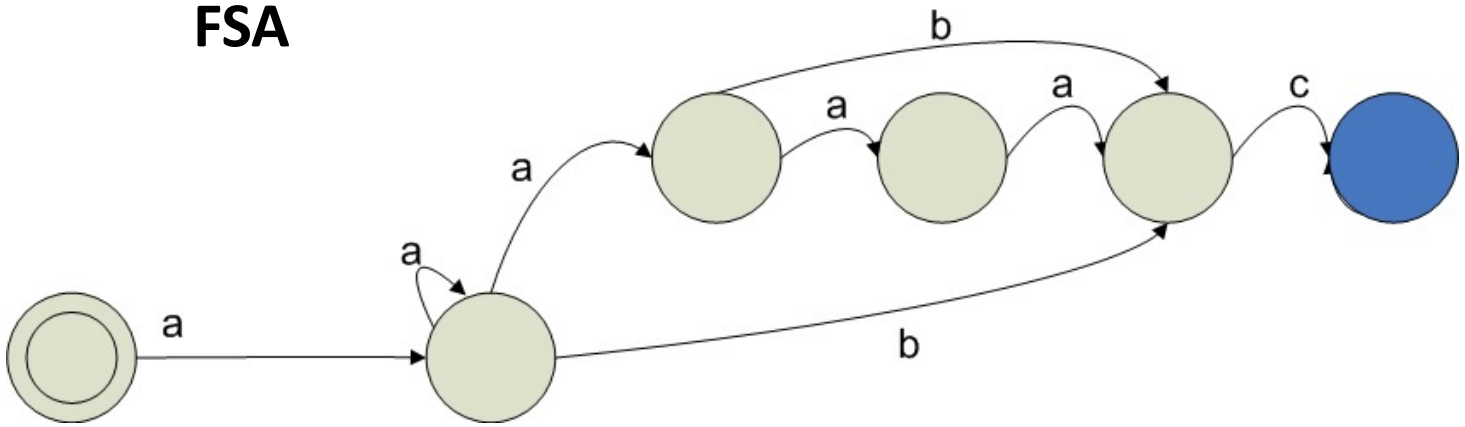


2-future(2) = {aa, ab, bc} = 2-future(3)

From the PTA to the FSA

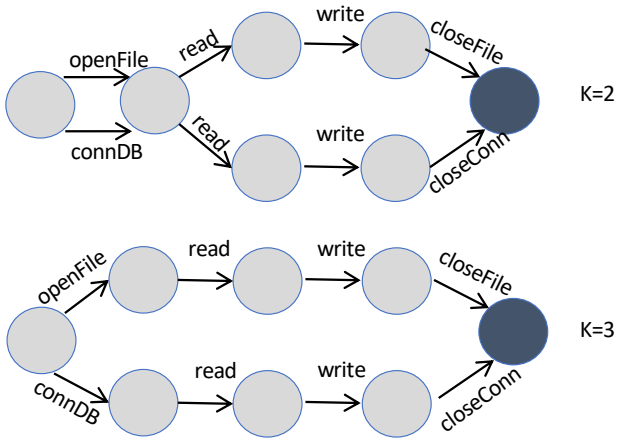


From the PTA to the FSA

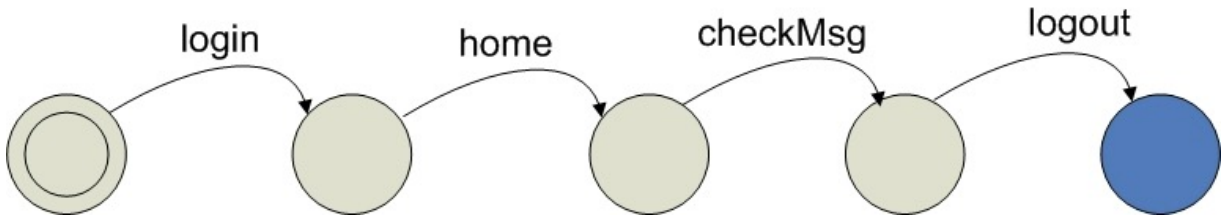


k-Tail: Features

- The FSA accepts all the traces used for inference
- But it also generalizes them (in the example the FSA accepts an infinite number of traces)
- The parameter k controls the degree of generalization:
  - High k = under-generalization
  - Low k = over-generalization

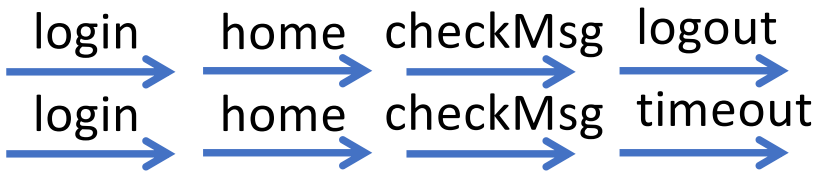


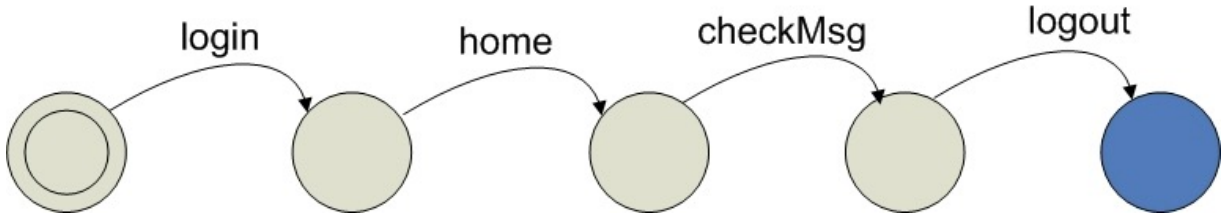
kBehavior



[Mariani, Pastore, Pezzè. Dynamic Analysis for Diagnosing Integration Faults. TSE, 2011.]

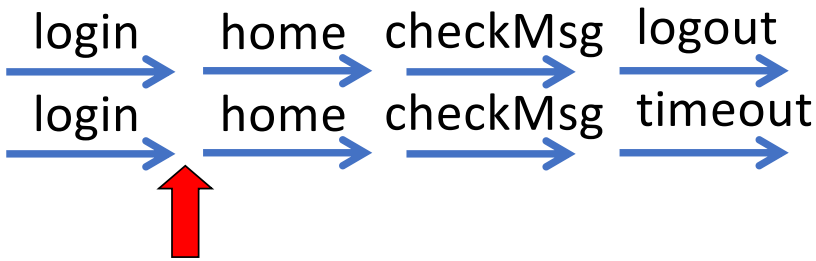
kBehavior

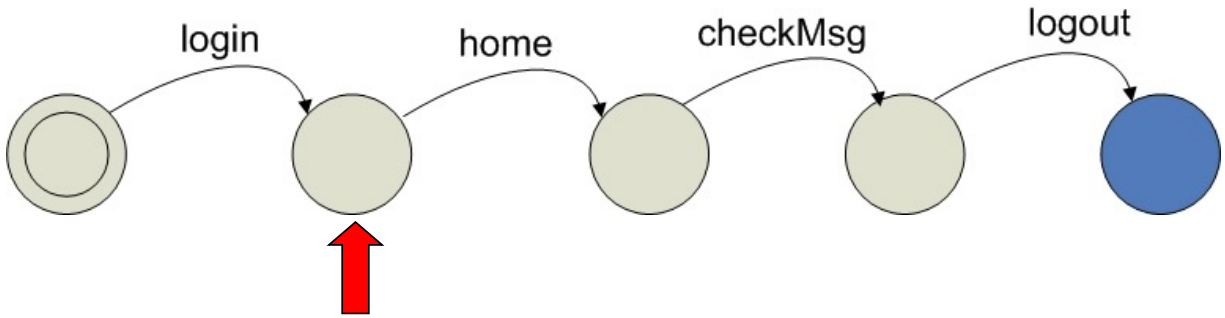




[Mariani, Pastore, Pezzè. Dynamic Analysis for Diagnosing Integration Faults. TSE, 2011.]

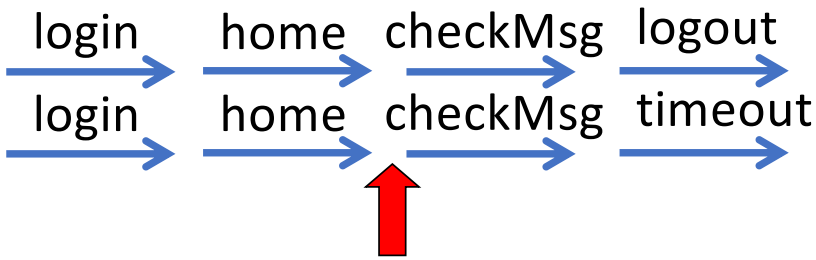
kBehavior

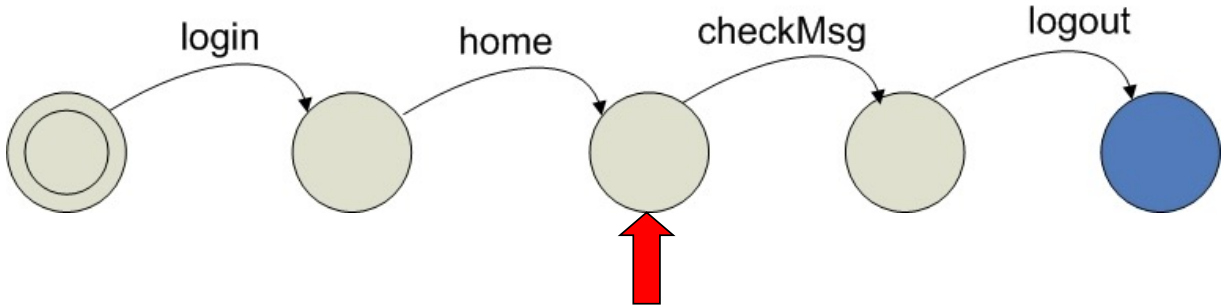




[Mariani, Pastore, Pezzè. Dynamic Analysis for Diagnosing Integration Faults. TSE, 2011.]

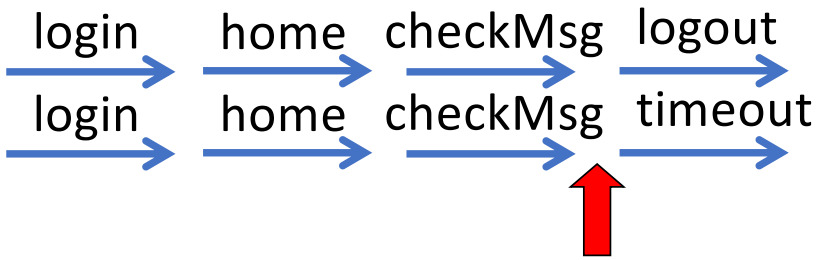
kBehavior

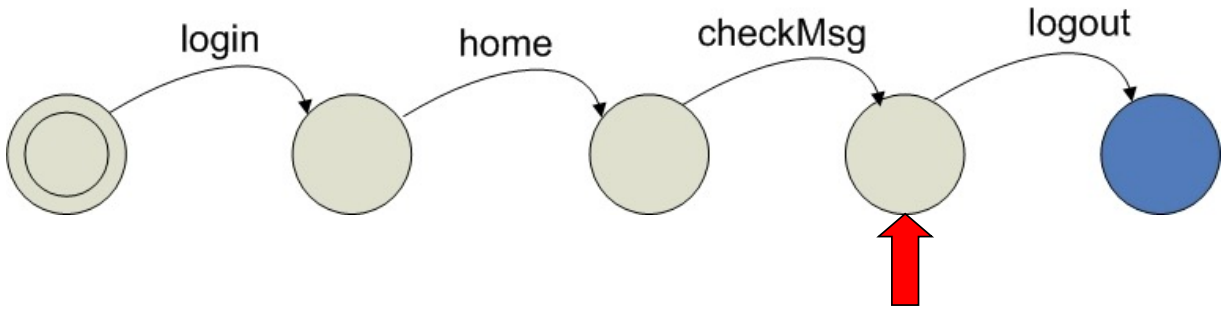




[Mariani, Pastore, Pezzè. Dynamic Analysis for Diagnosing Integration Faults. TSE, 2011.]

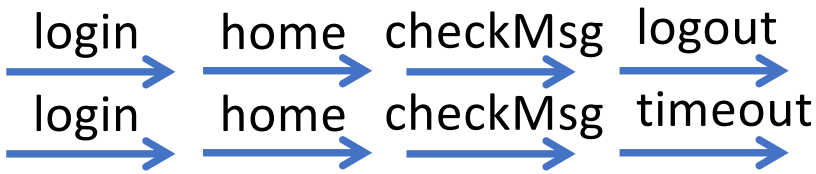
kBehavior

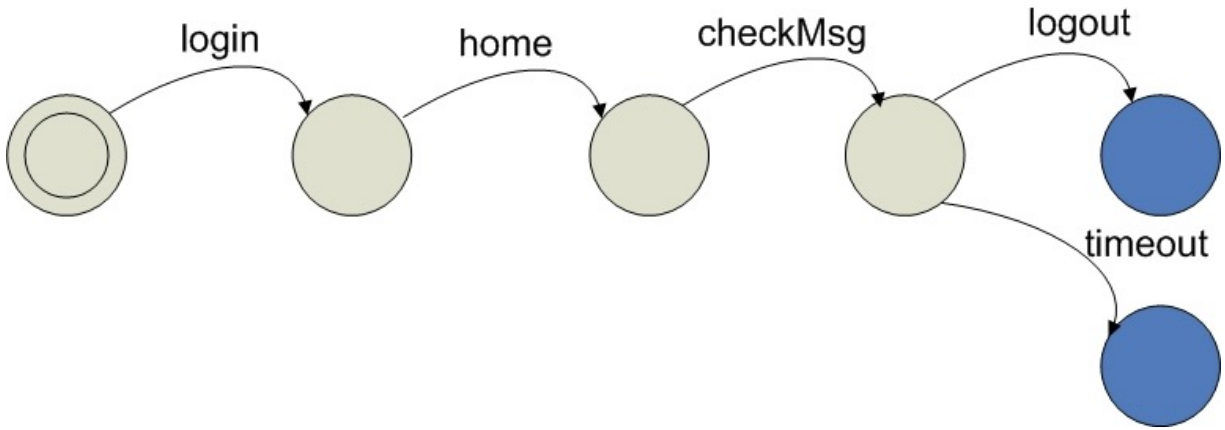




[Mariani, Pastore, Pezzè. Dynamic Analysis for Diagnosing Integration Faults. TSE, 2011.]

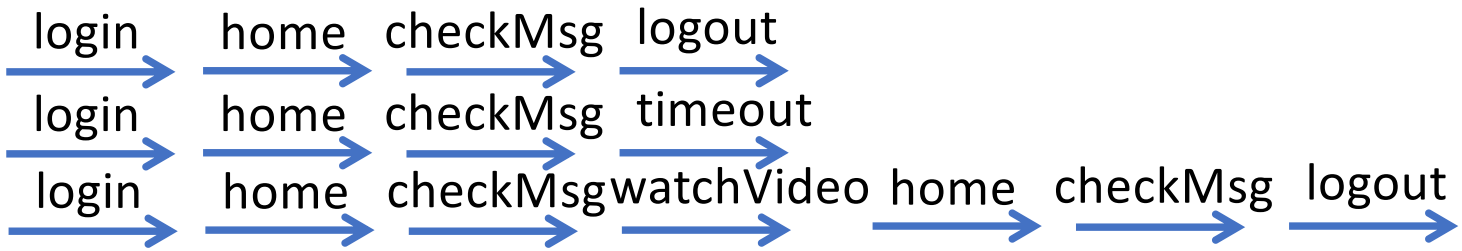
kBehavior

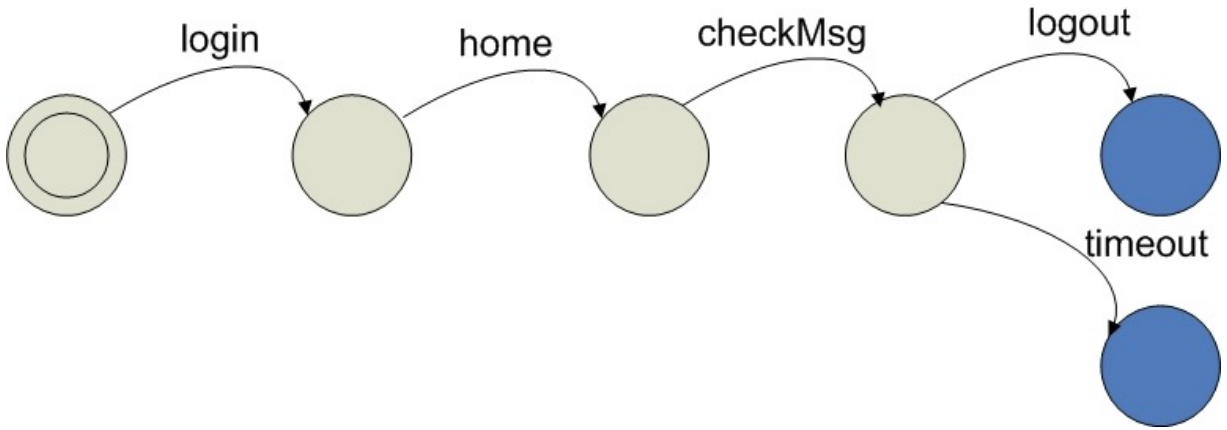




[Mariani, Pastore, Pezzè. Dynamic Analysis for Diagnosing Integration Faults. TSE, 2011.]

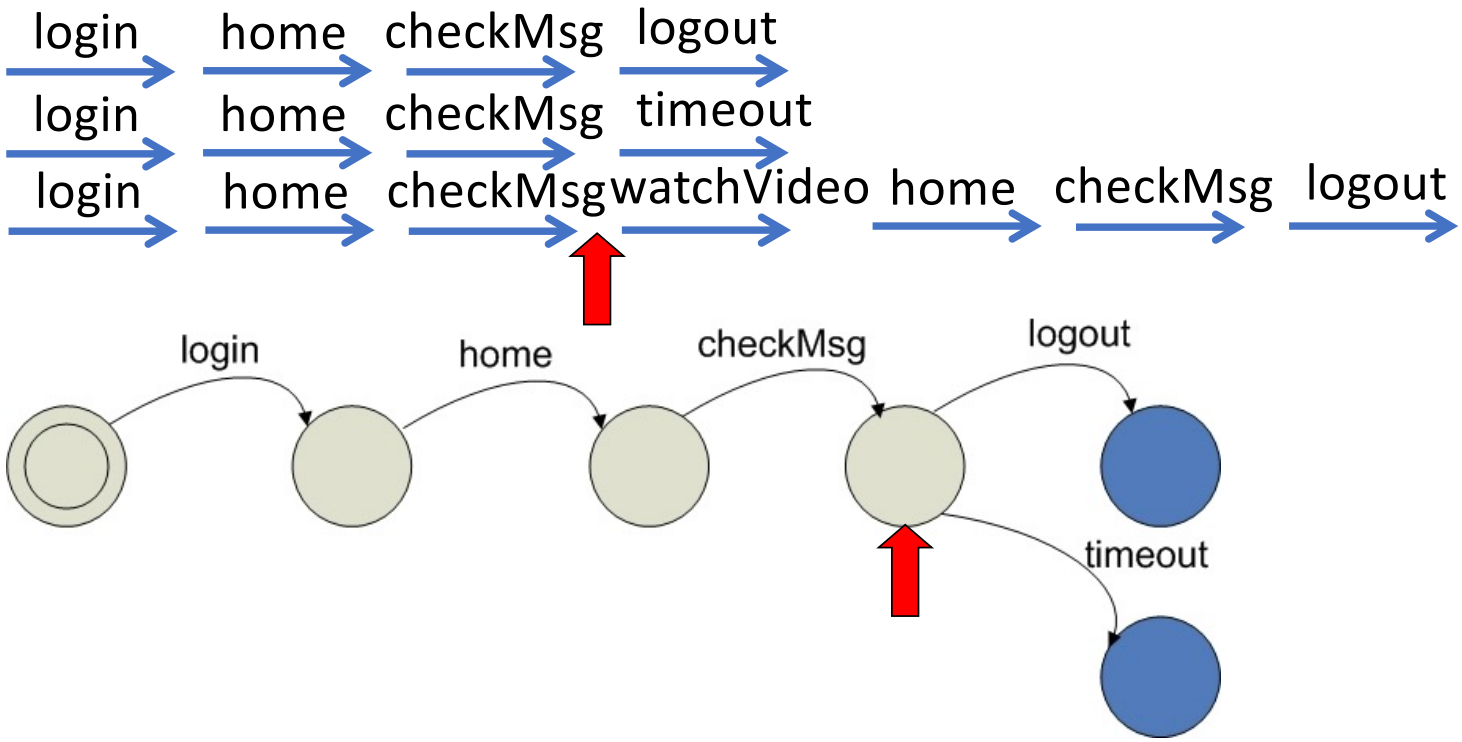
kBehavior





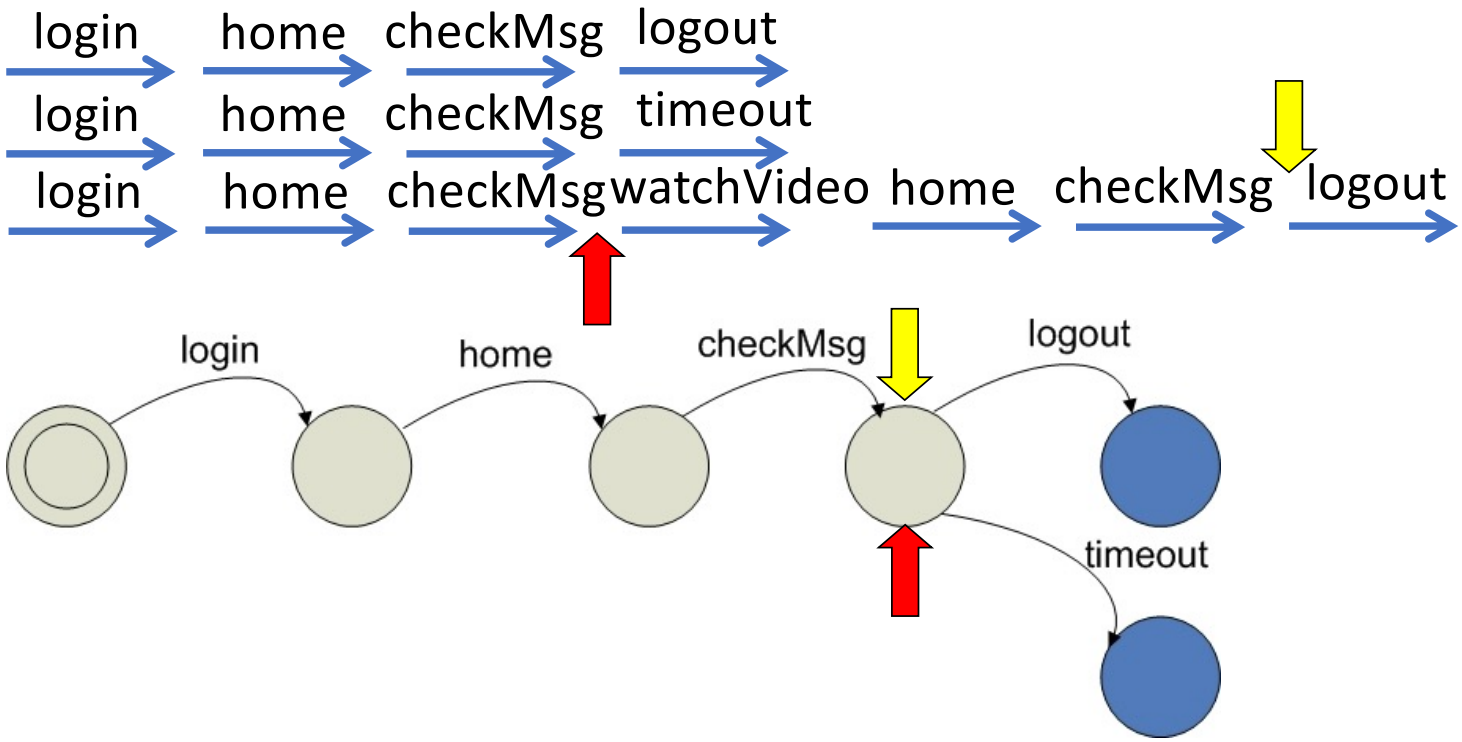
[Mariani, Pastore, Pezzè. Dynamic Analysis for Diagnosing Integration Faults. TSE, 2011.]

kBehavior



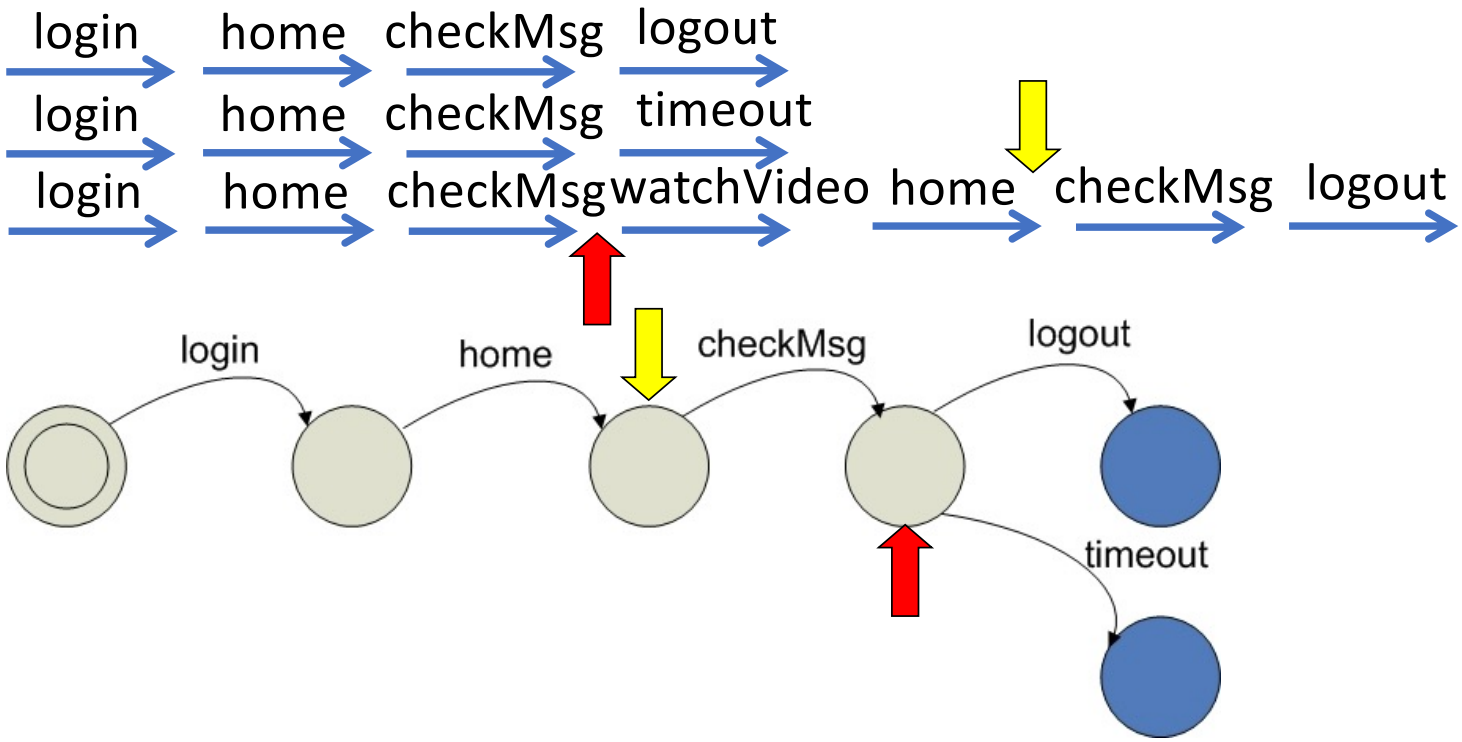
[Mariani, Pastore, Pezzè. Dynamic Analysis for Diagnosing Integration Faults. TSE, 2011.]

kBehavior



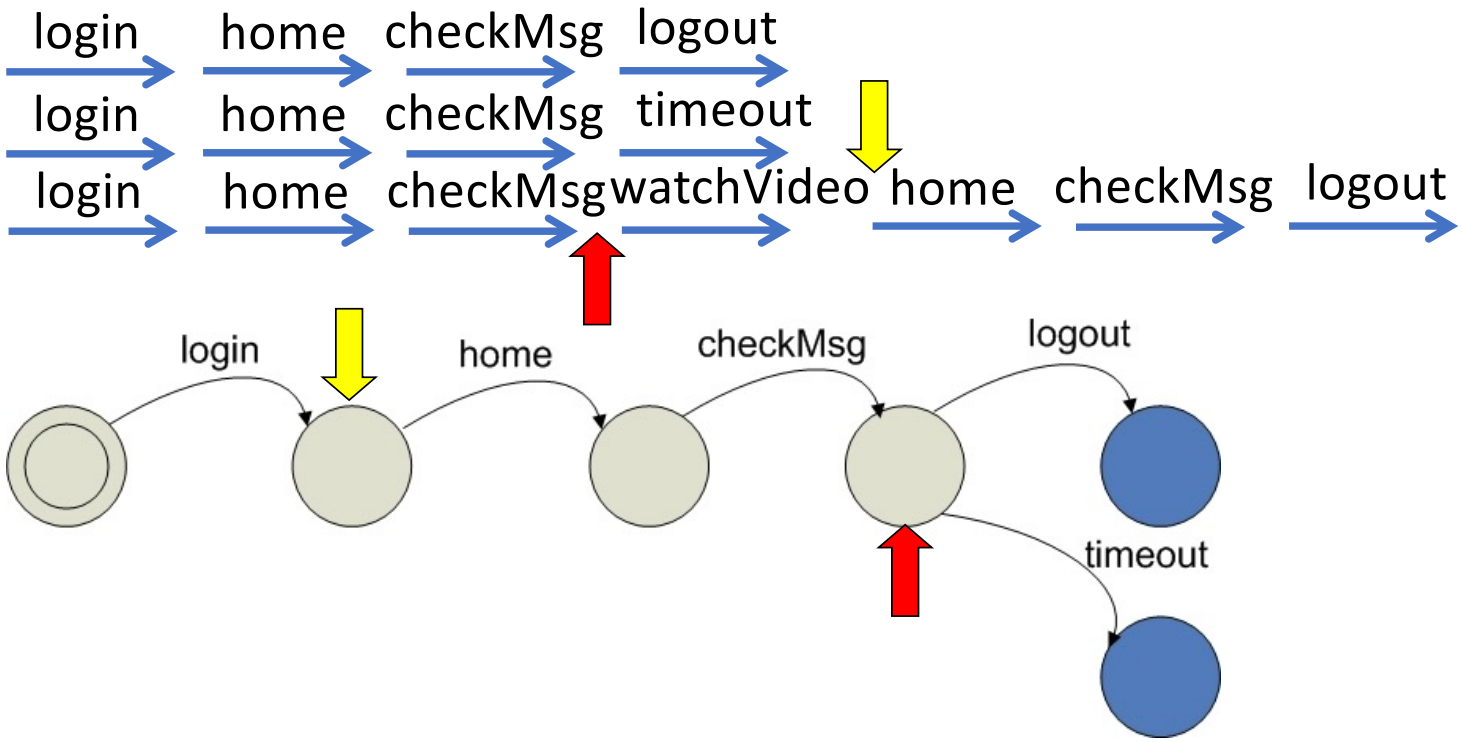
[Mariani, Pastore, Pezzè. Dynamic Analysis for Diagnosing Integration Faults. TSE, 2011.]

kBehavior



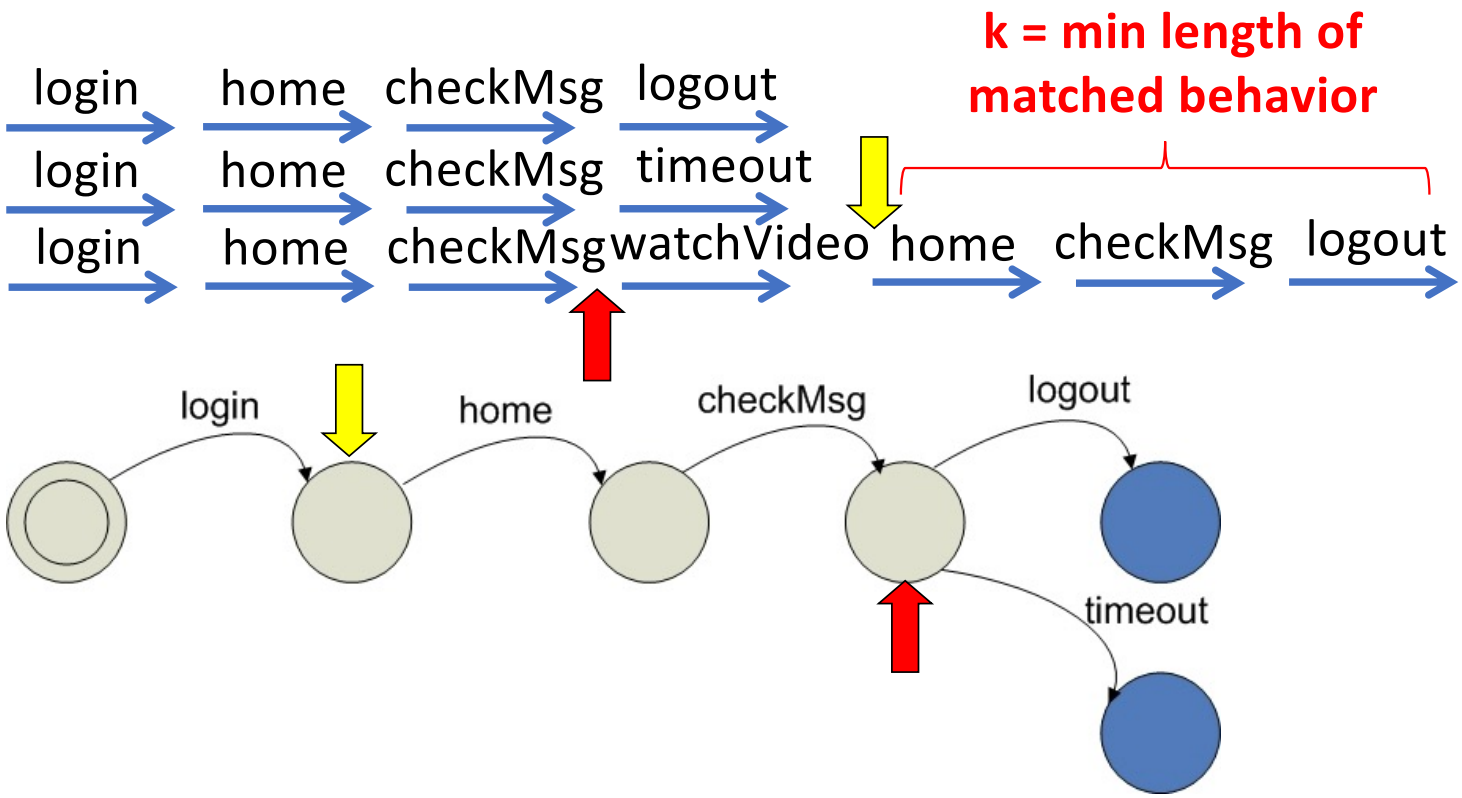
[Mariani, Pastore, Pezzè. Dynamic Analysis for Diagnosing Integration Faults. TSE, 2011.]

kBehavior



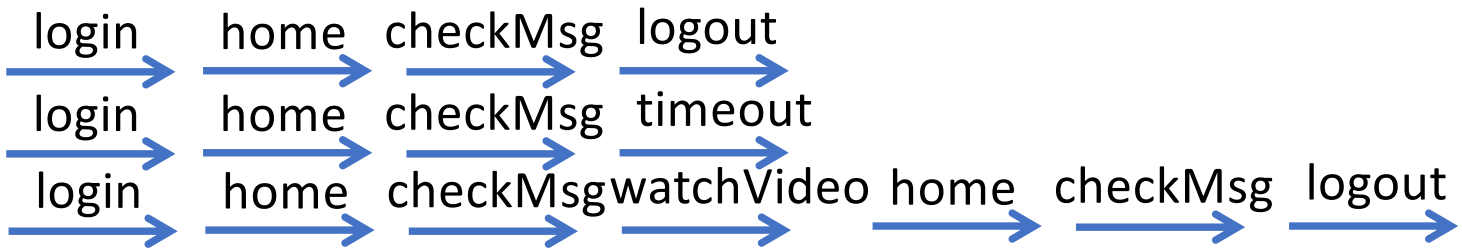
[Mariani, Pastore, Pezzè. Dynamic Analysis for Diagnosing Integration Faults. TSE, 2011.]

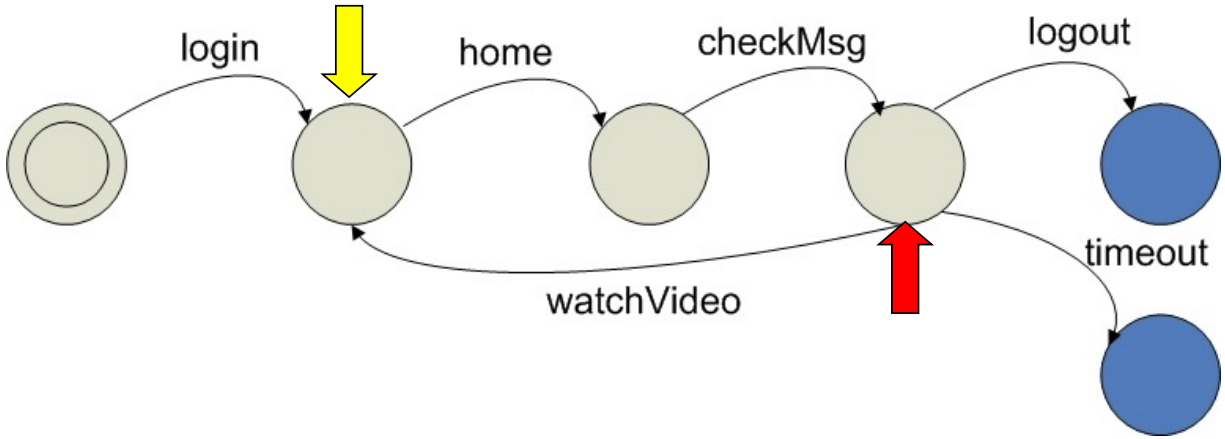
kBehavior



[Mariani, Pastore, Pezzè. Dynamic Analysis for Diagnosing Integration Faults. TSE, 2011.]

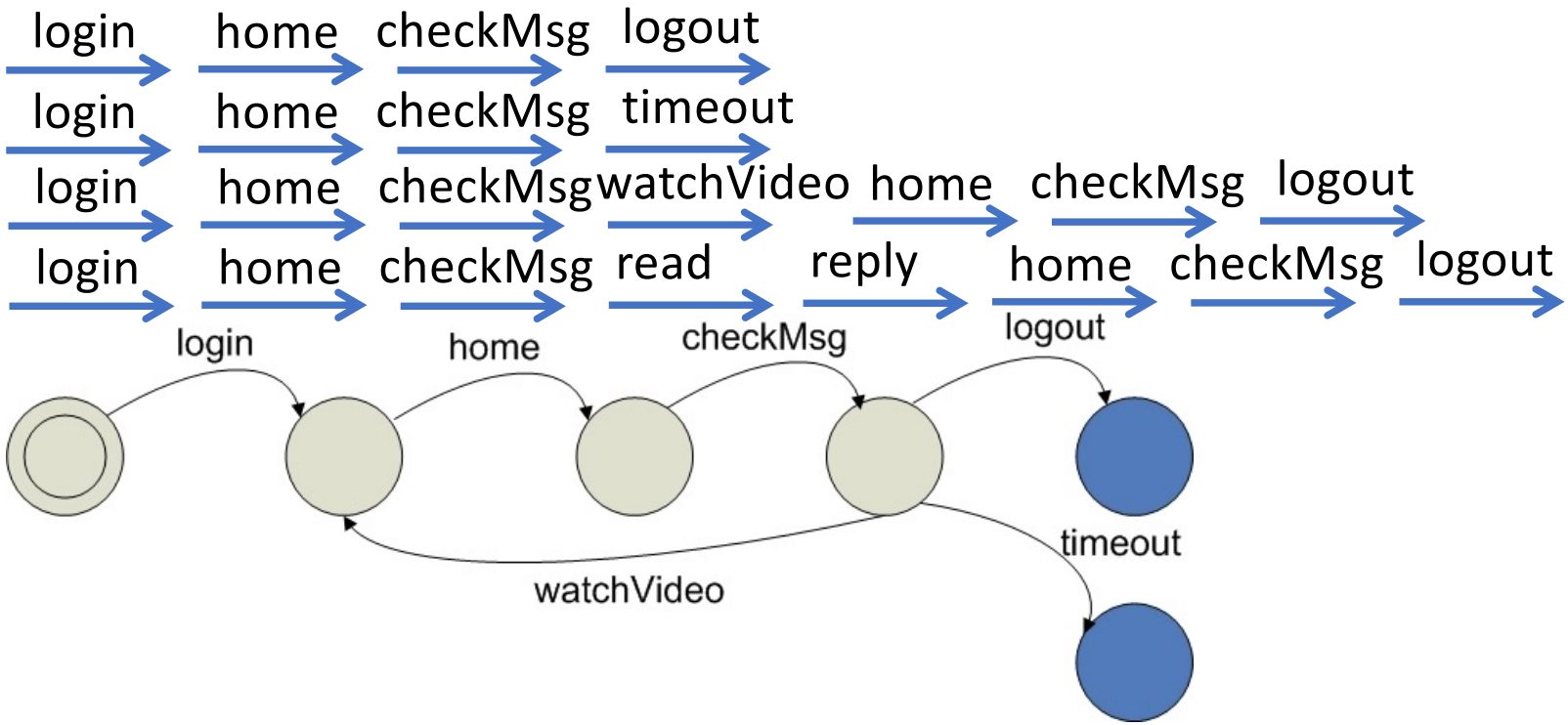
kBehavior





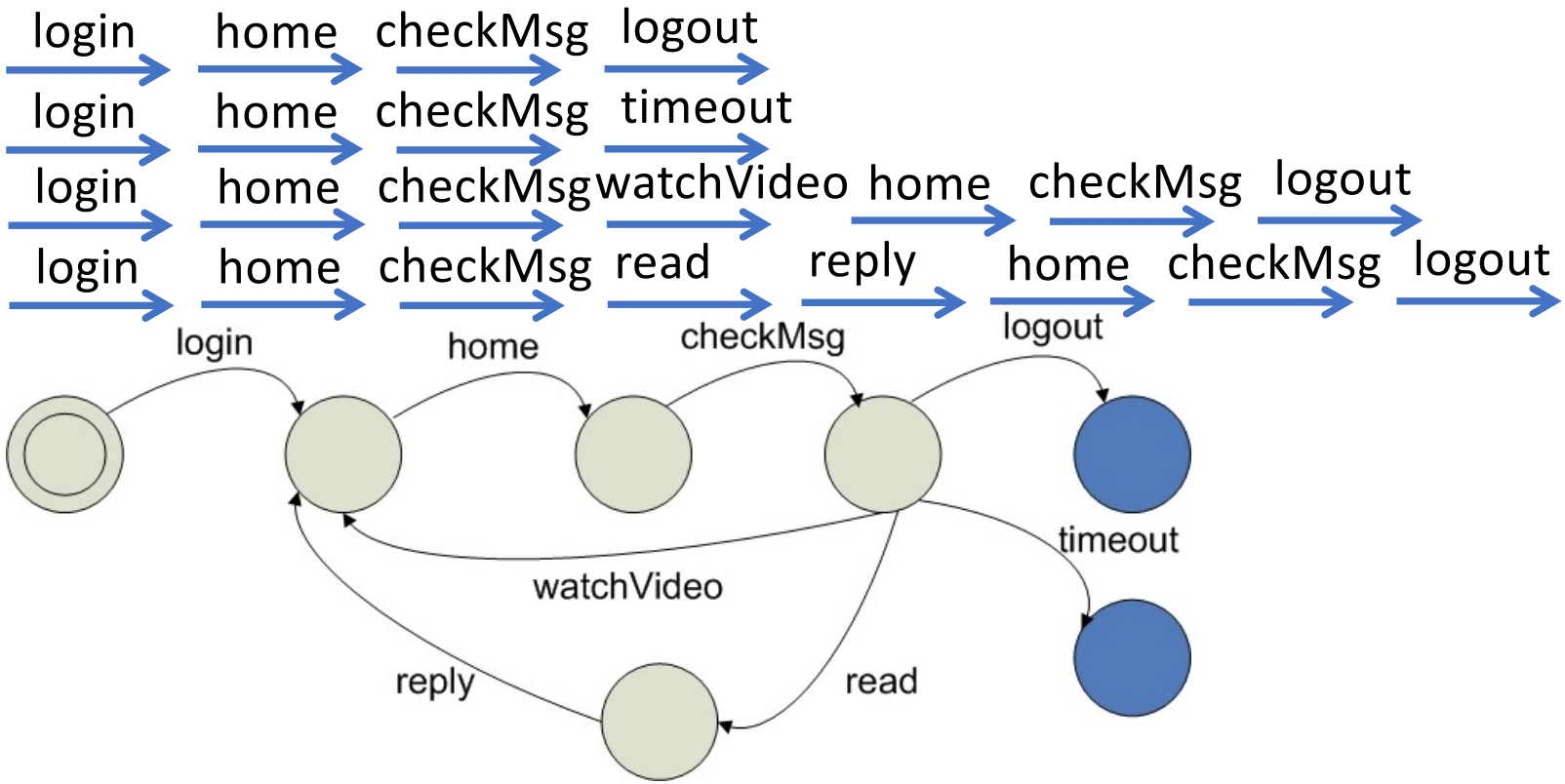
[Mariani, Pastore, Pezzè. Dynamic Analysis for Diagnosing Integration Faults. TSE, 2011.]

kBehavior



[Mariani, Pastore, Pezzè. Dynamic Analysis for Diagnosing Integration Faults. TSE, 2011.]

kBehavior

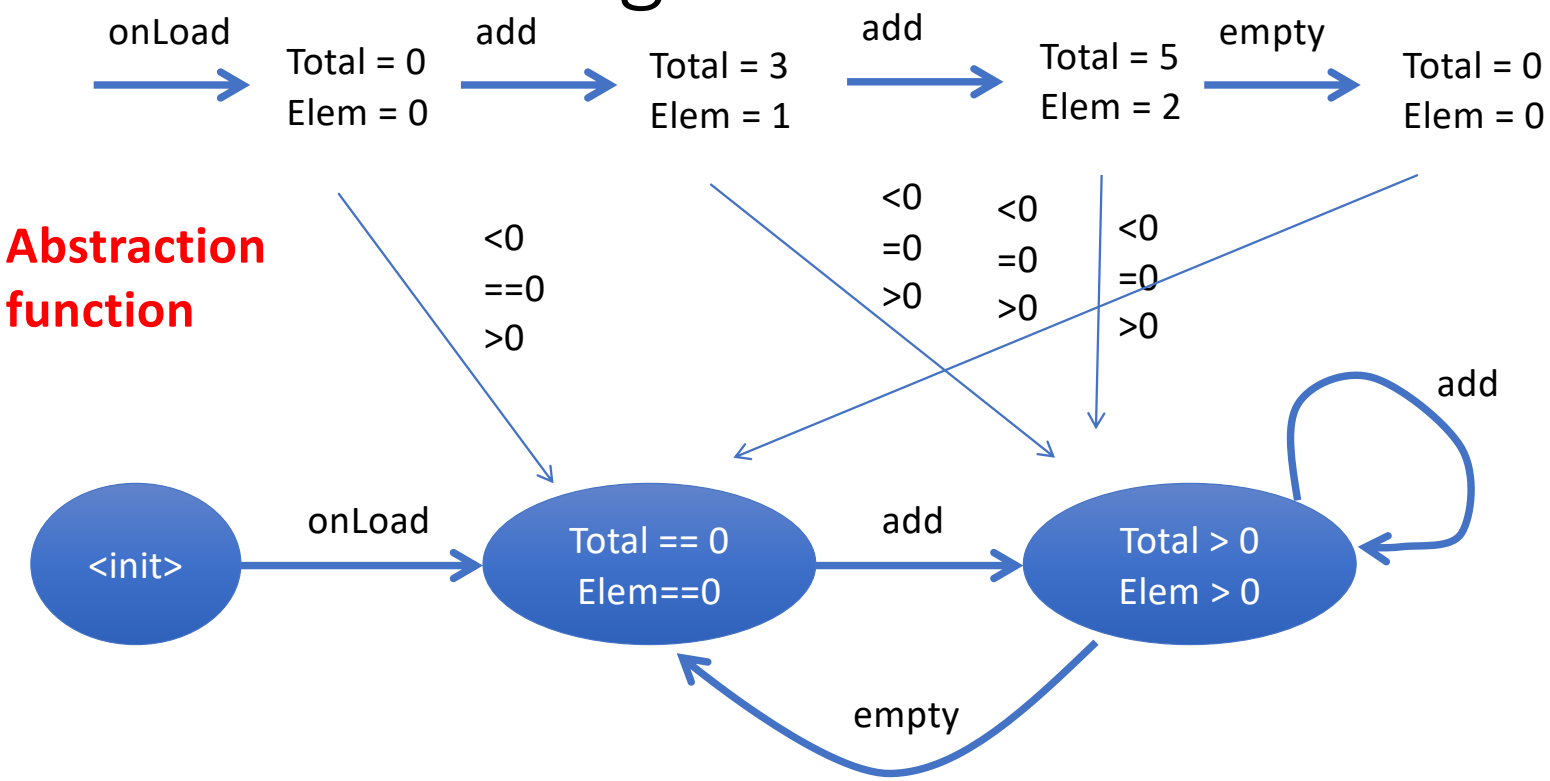


[Mariani, Pastore, Pezzè. Dynamic Analysis for Diagnosing Integration Faults. TSE, 2011.]

kBehavior: Features

- Empirically, behavior-based merging generates models that are more general than state-based merging [Lo et al., JSS, 2012]

State-based mining



[Dallmeier, Lindig, Wasylkowski, Zeller: Mining Object Behavior with ADABU. WODA 2006]
[Marchetto, Tonella, Ricca: State-Based Testing of Ajax Web Applications. ICST 2008]
[Mariani, Marchetto, Nguyen, Tonella. Revolution: Automatic evolution of mined specifications. ISSRE. 2012]

State-based mining: Features

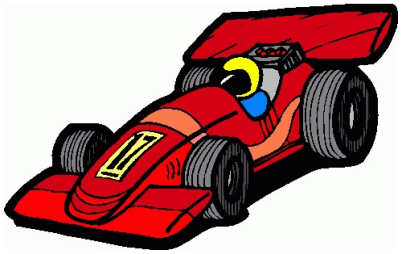
- The quality of the final model is influenced by the completeness of the state information that is traced…
- …and by the kind of abstraction implemented by the abstraction function
- ADABU: https://www.st.cs.uni-saarland.de/models/adabu.php3

Procedural vs. Declarative

construction mechanism

inference

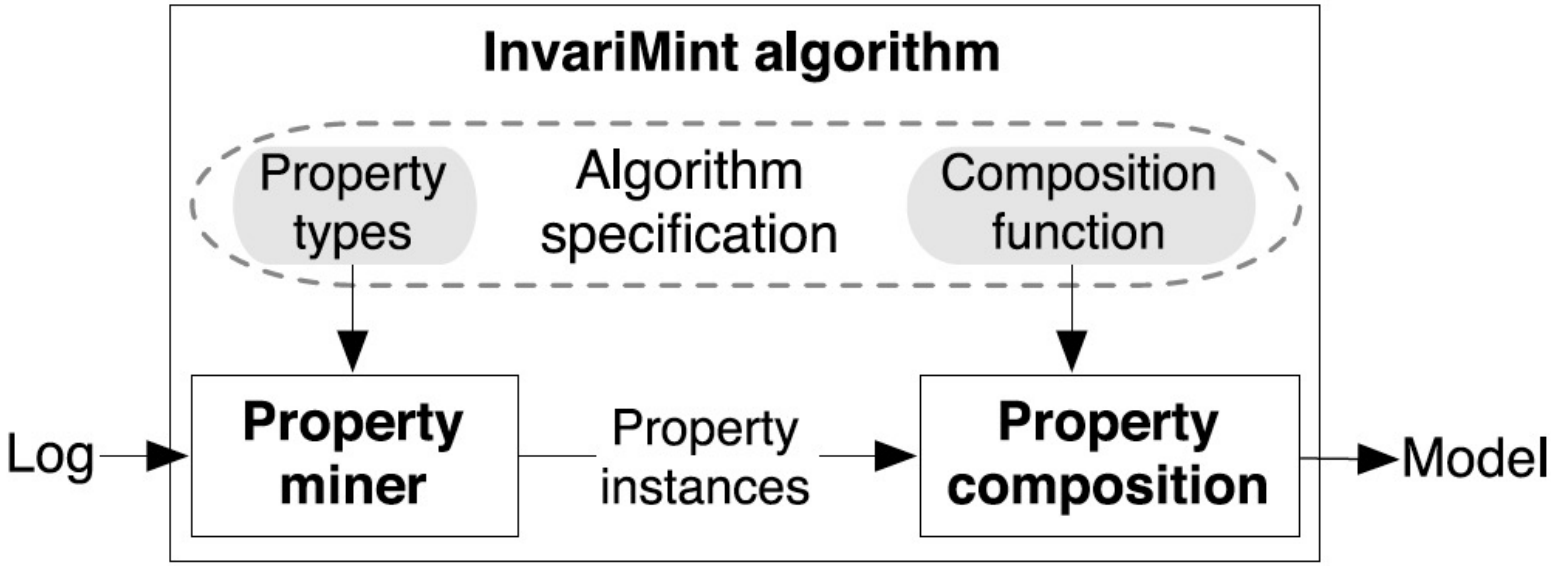
...



properties of the result

inference

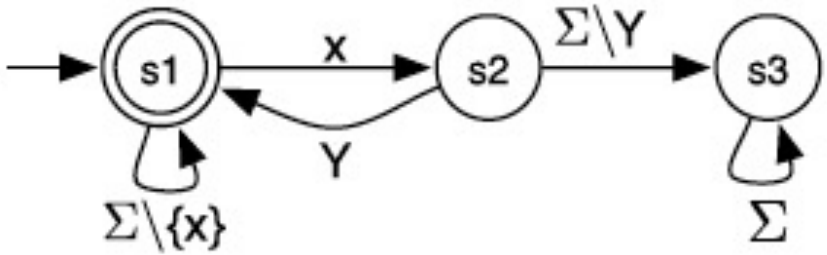
The InvariMint approach to the specification of model inference algorithm



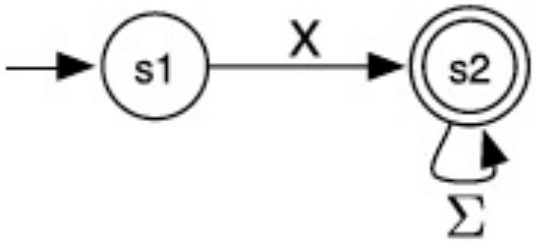
[Beschastnikh et al. Using Declarative Specification to Improve the Understanding, Extensibility, and Comparison of Model-Inference Algorithms. TSE. 2015]

An example of algorithm (1)

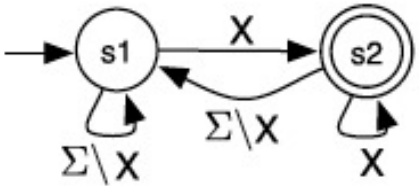
Property types



x immediately followed by an event in Y



start with x in X

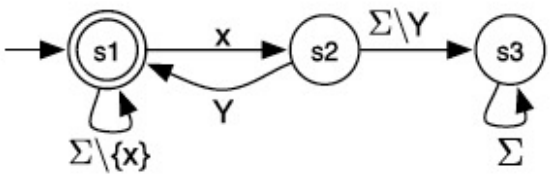


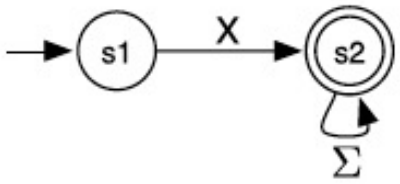
end with y in Y

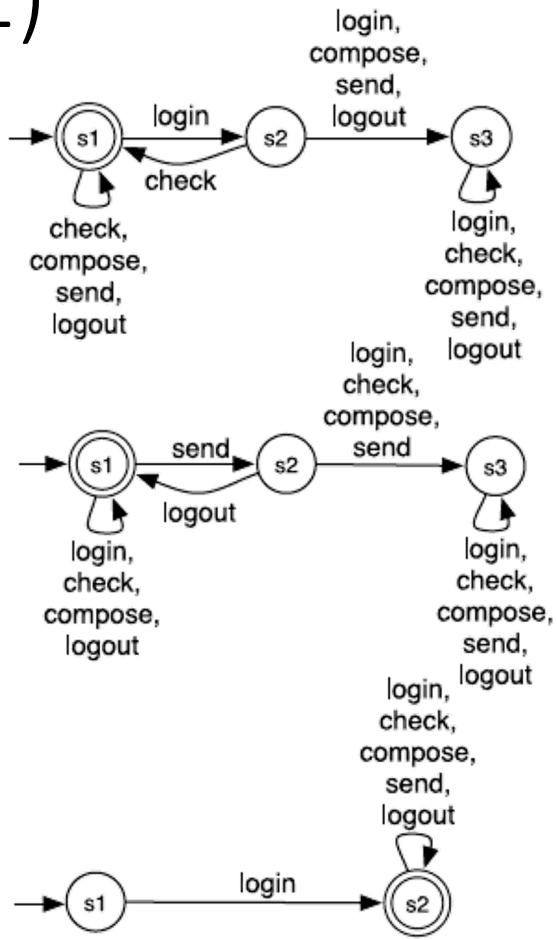
An example of algorithm (2)

trace 1:	trace 2:
login check check logout	login check compose send logout

Property mining

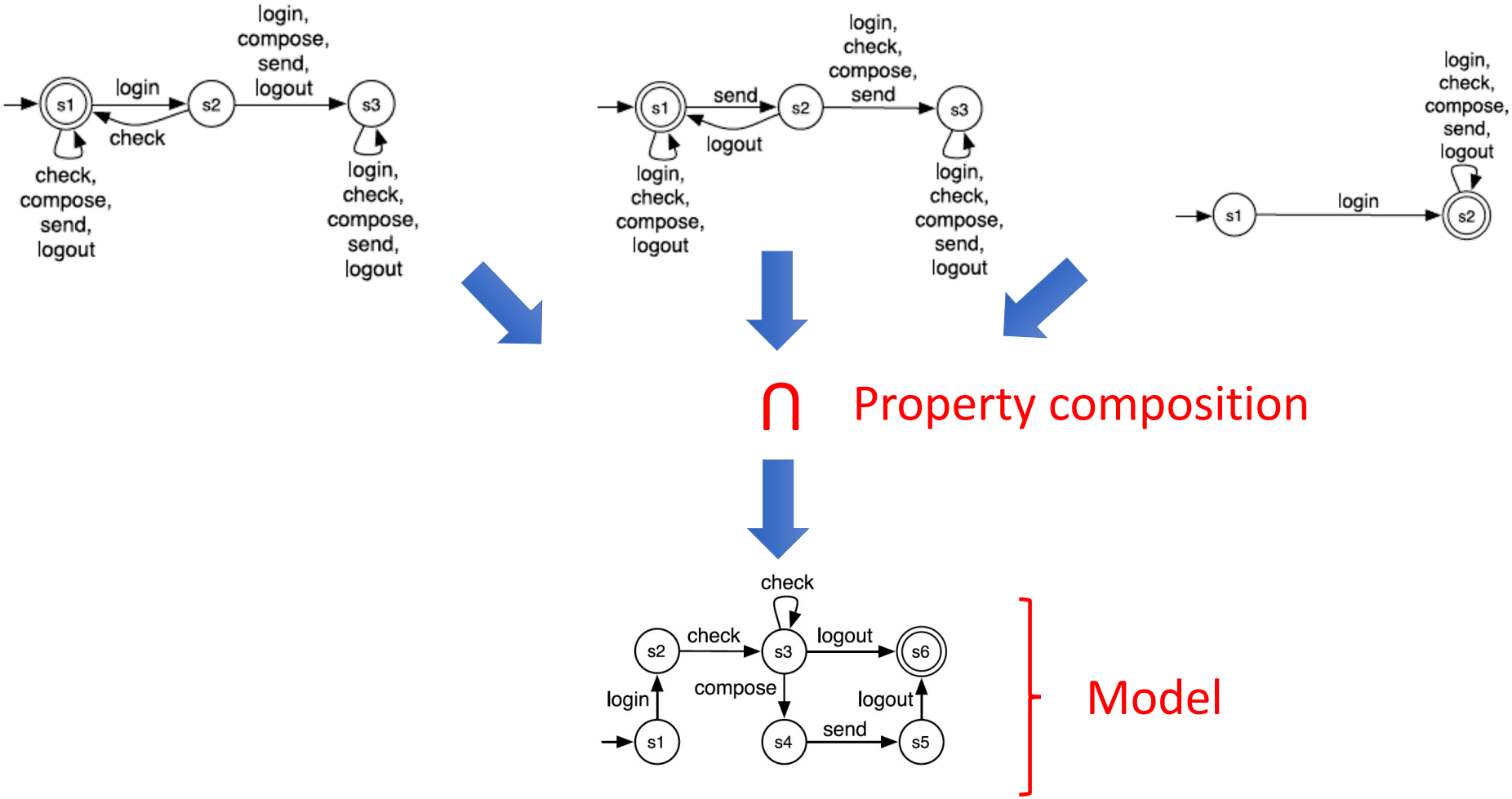






Property instances

An example of algorithm (3)



Declarative approach: Features

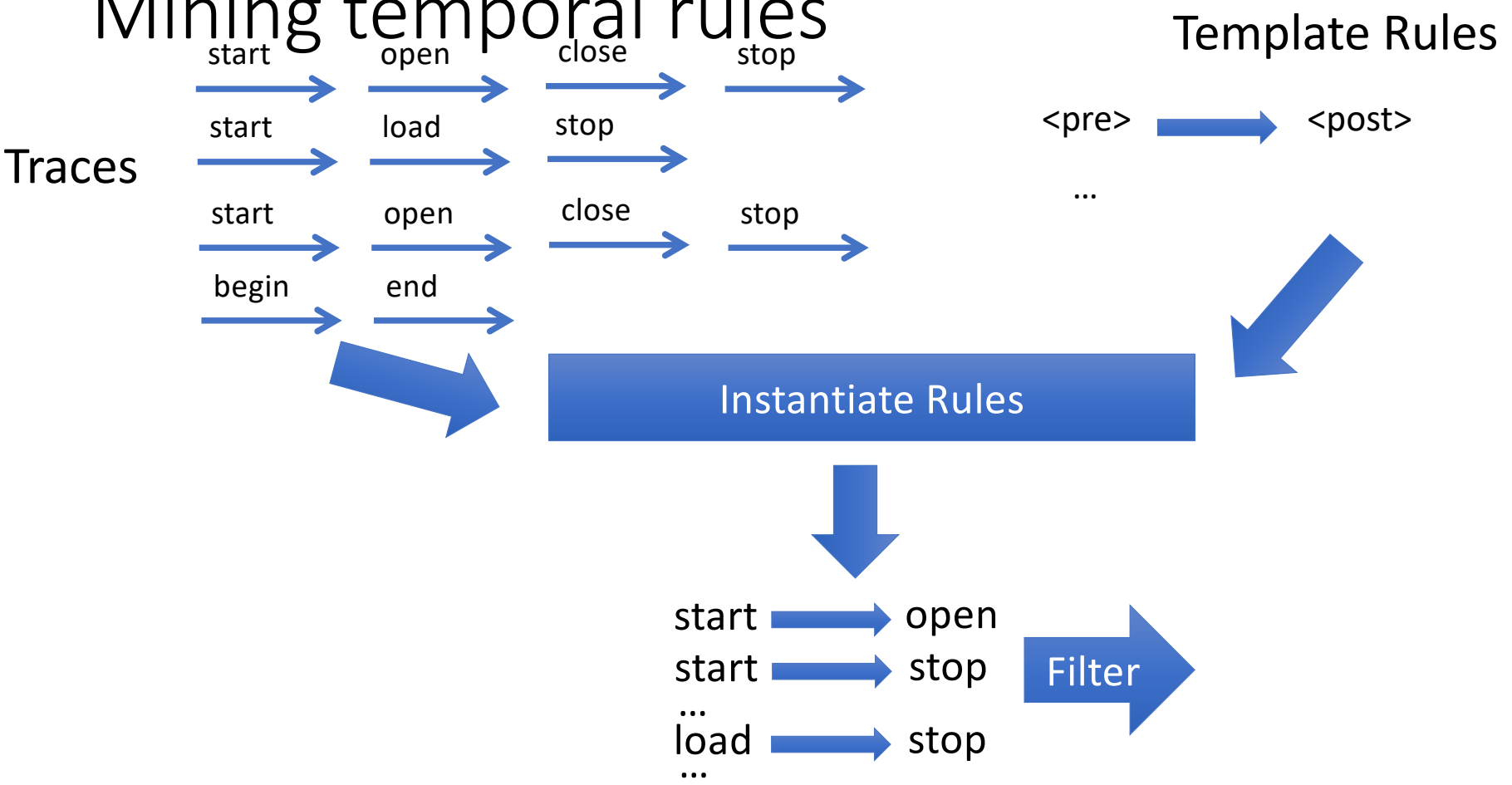
- Pros:
  - You know the properties that the inferred automaton will satisfy
  - Usually faster than procedural algorithms
- Cons:
  - You may miss emerging properties that can be captured with procedural approaches

InvariMint

- Available at https://github.com/modelinference/synoptic
- The repository contains other model inference tools:
  - **Synoptic** : a tool to infer an FSM model from a sequential log (see later in these slides)
  - **CSight** : a tool to infer a communicating FSM model from a distributed system's logs
  - **Perfume** : a tool to infer performance models from system logs

Models of events, partial order

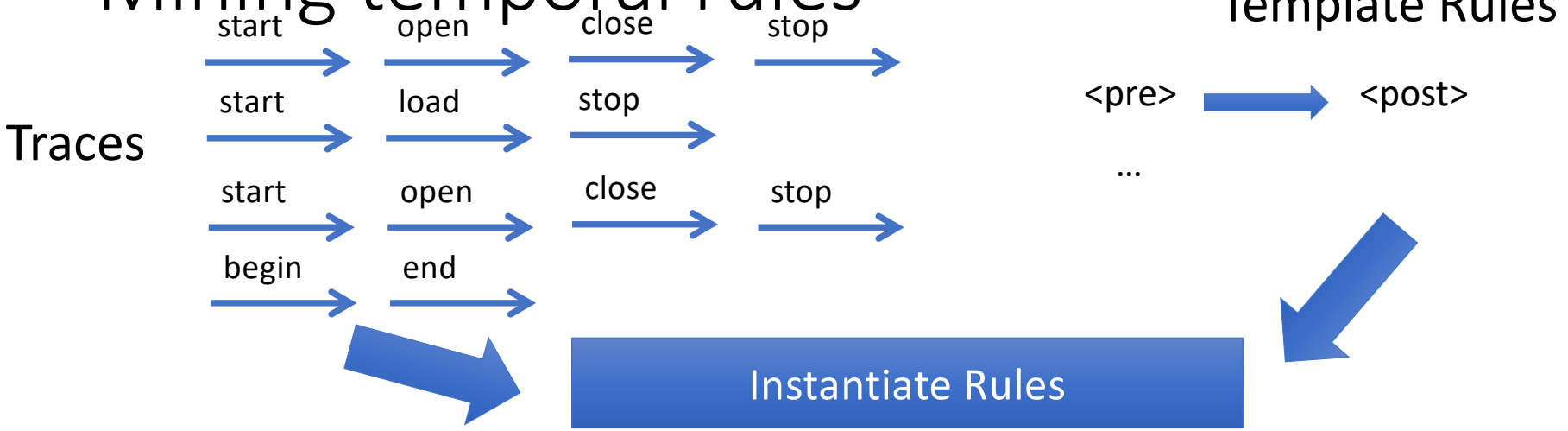
Mining temporal rules



[Lo, Khoo, Liu. Mining temporal rules for software maintenance. JSME, 2008]

[Yang, et al. Perracotta: mining temporal API Rules from Imperfect Traces. ICSE. 2006]

Mining temporal rules

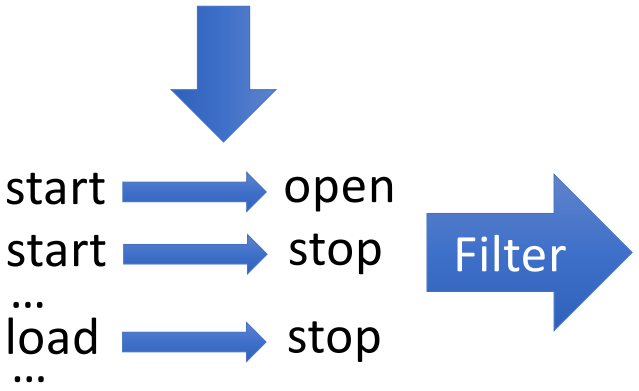


CONFIDENCE OF A RULE

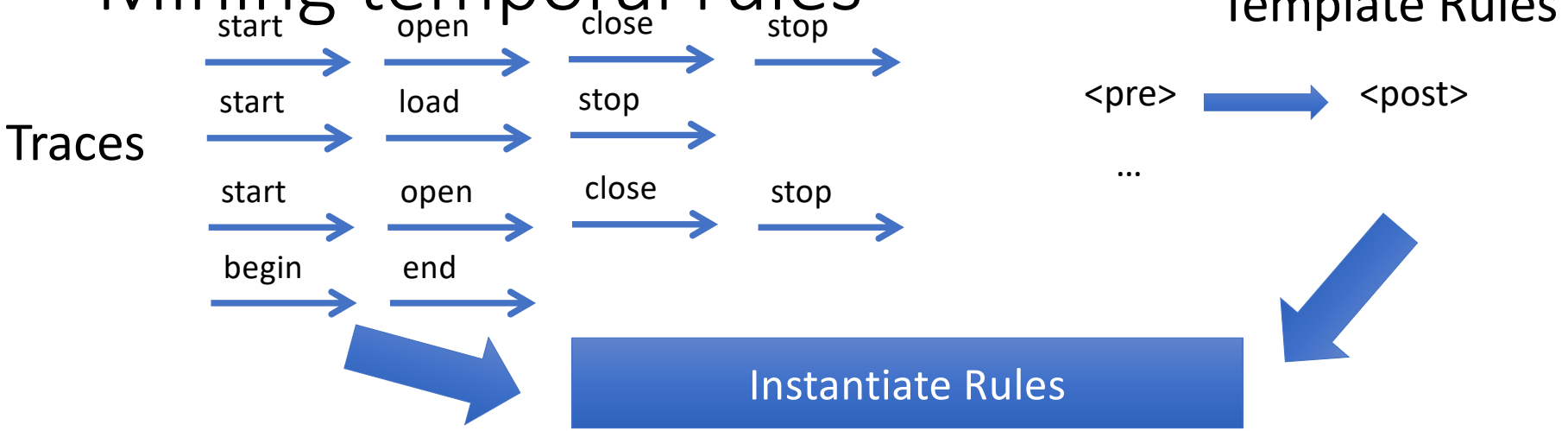
$$\frac{\#\text{ traces rule holds}}{\#\text{traces pre holds}}$$

start → open

What's its confidence?



Mining temporal rules

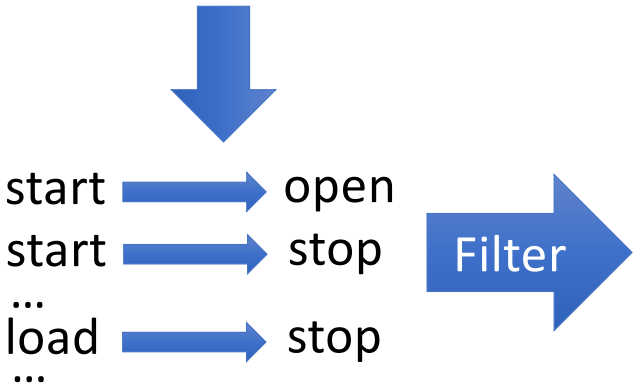


CONFIDENCE OF A RULE

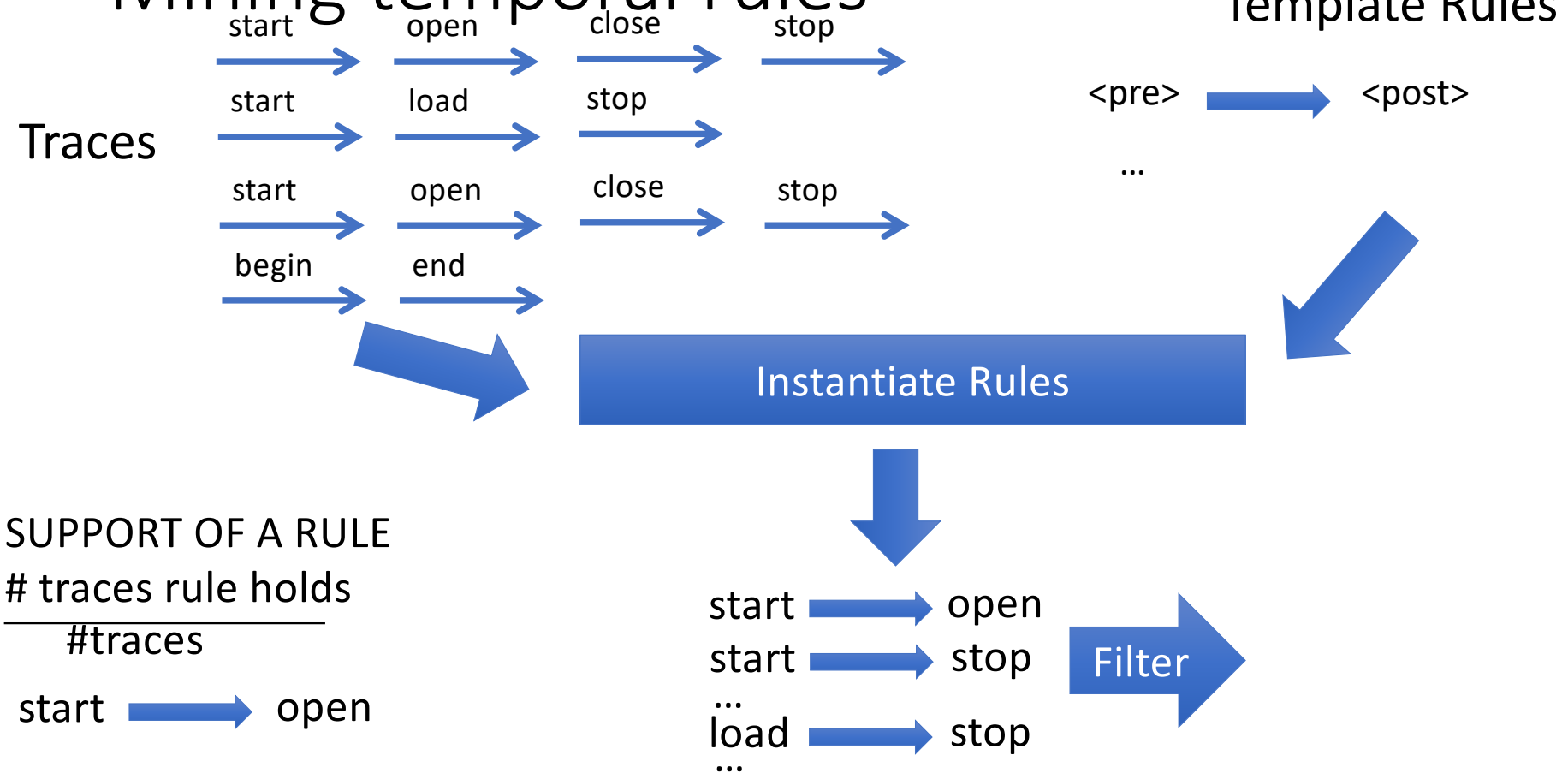
$$\frac{\#\text{ traces rule holds}}{\#\text{traces pre holds}}$$

start → open

has 67% confidence



Mining temporal rules



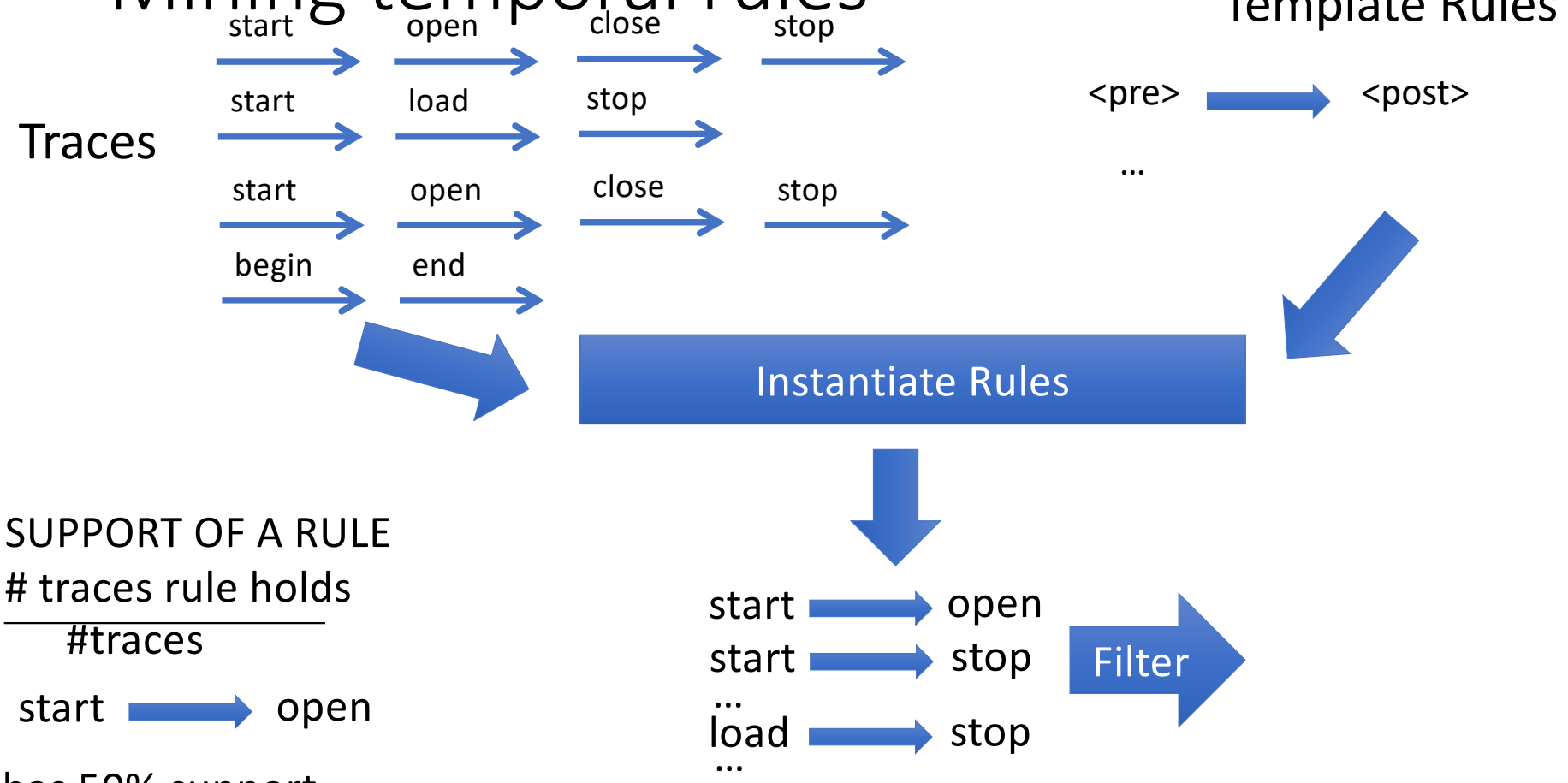
SUPPORT OF A RULE

$$\frac{\text{\# traces rule holds}}{\text{\#traces}}$$

start → open

What's its support?

Mining temporal rules



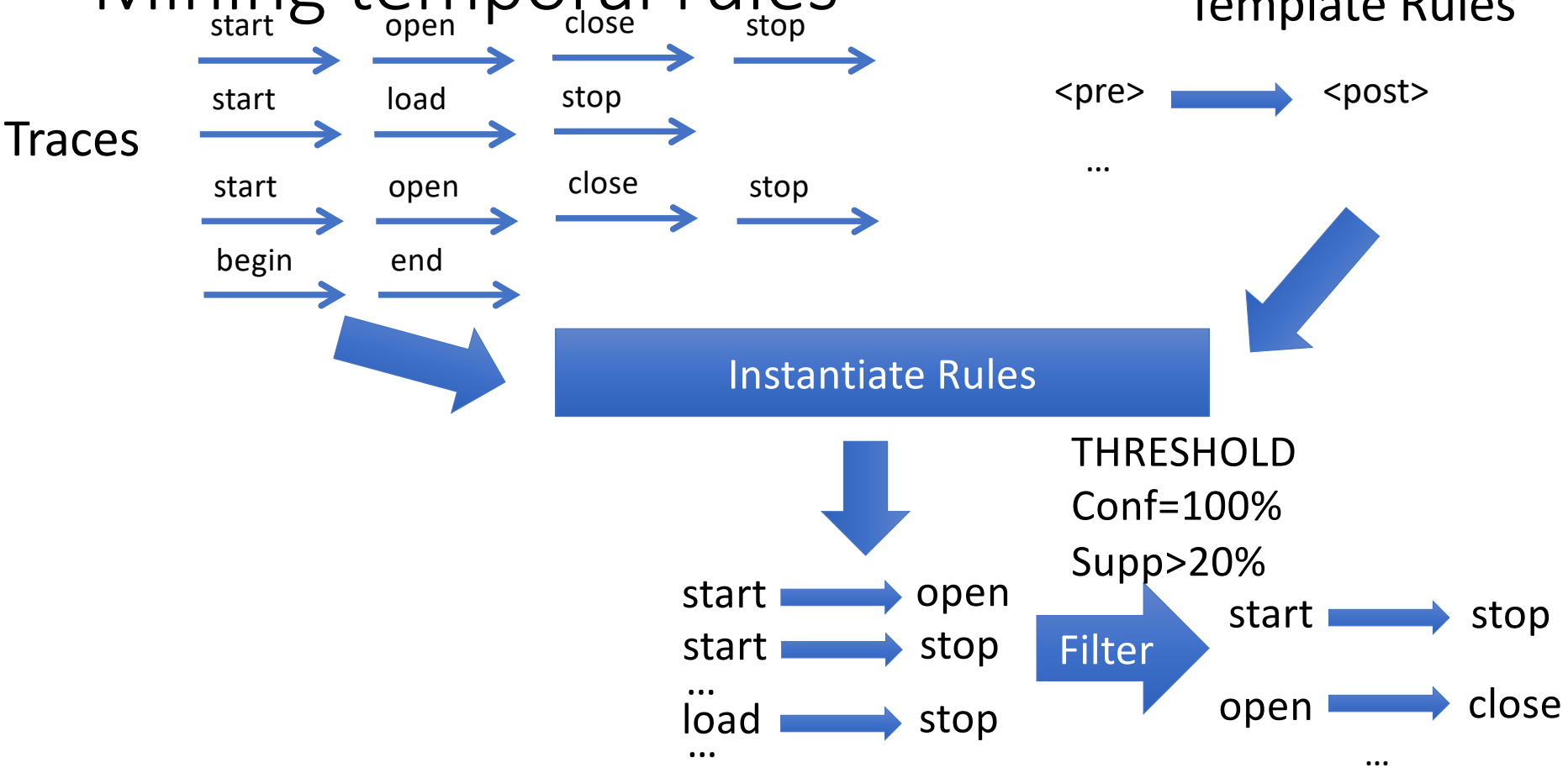
SUPPORT OF A RULE

$$\frac{\text{\# traces rule holds}}{\text{\#traces}}$$

start → open

has 50% support

Mining temporal rules



Mining temporal rules: Discussion

- Expressiveness depends on the template rules
- Confidence and support for tuning the technique wrt imperfect traces
- Example of tool: Texada
  - https://bitbucket.org/bestchai/texada
  - Supports full LTL
  - It uses conf=100% and supp>0%
  - Available as an online service, http://elaine.nss.cs.ubc.ca:8080/texada/
- Other tool: Perracotta, http://www.cs.virginia.edu/perracotta/

Models of data

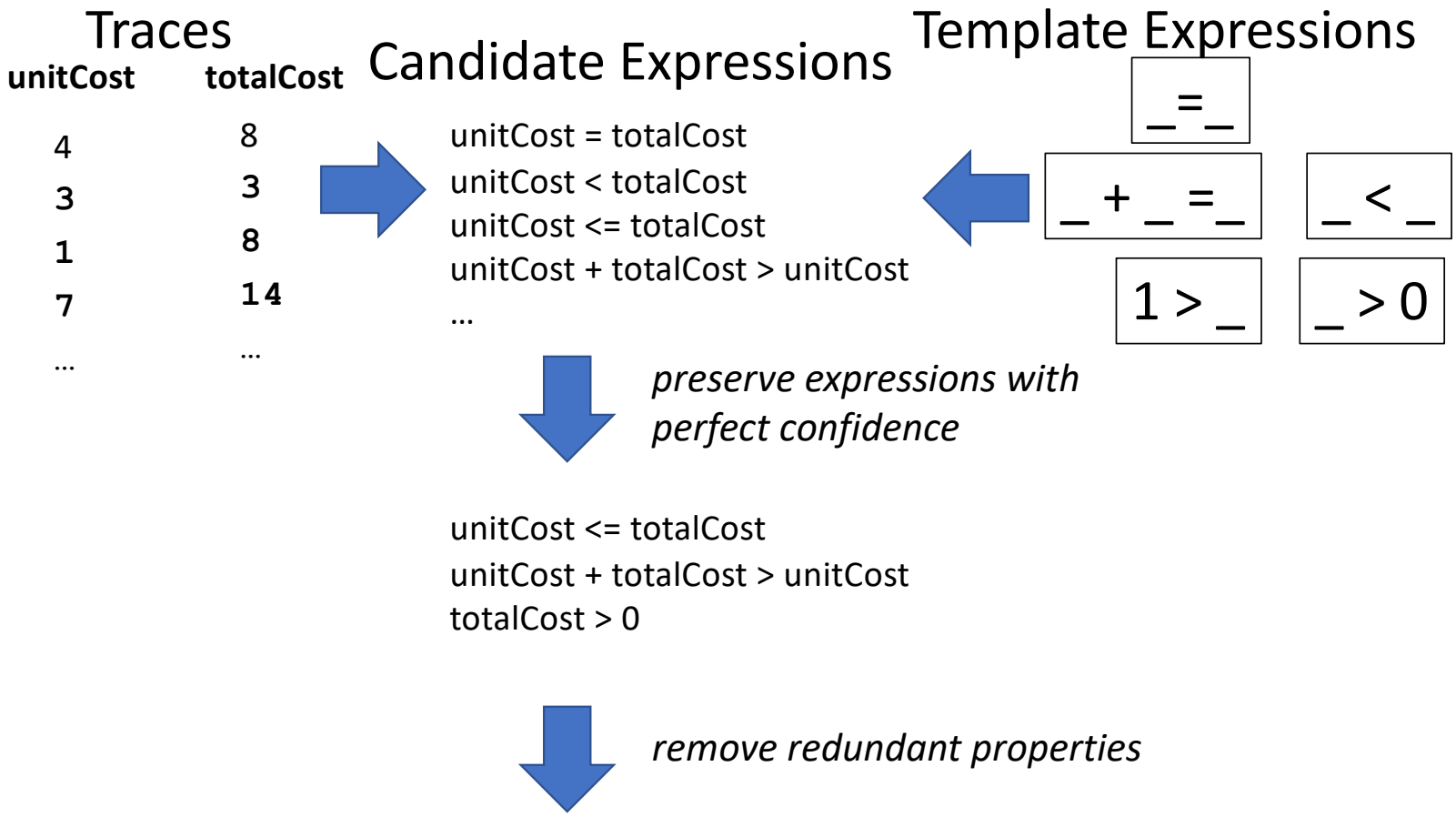
Example: program for ecommerce

Executions

Log values of vars

	unitCost	totalCost
exec 1	4	8
exec 2	3	3
exec 3	1	8
exec 4	7	14
…	…	…

Daikon



[Ernst, Cockrell, Griswold, Notkin. Dynamically Discovering Likely Program Invariants to Support Program Evolution. IEEE TSE 2001]

Daikon: Features

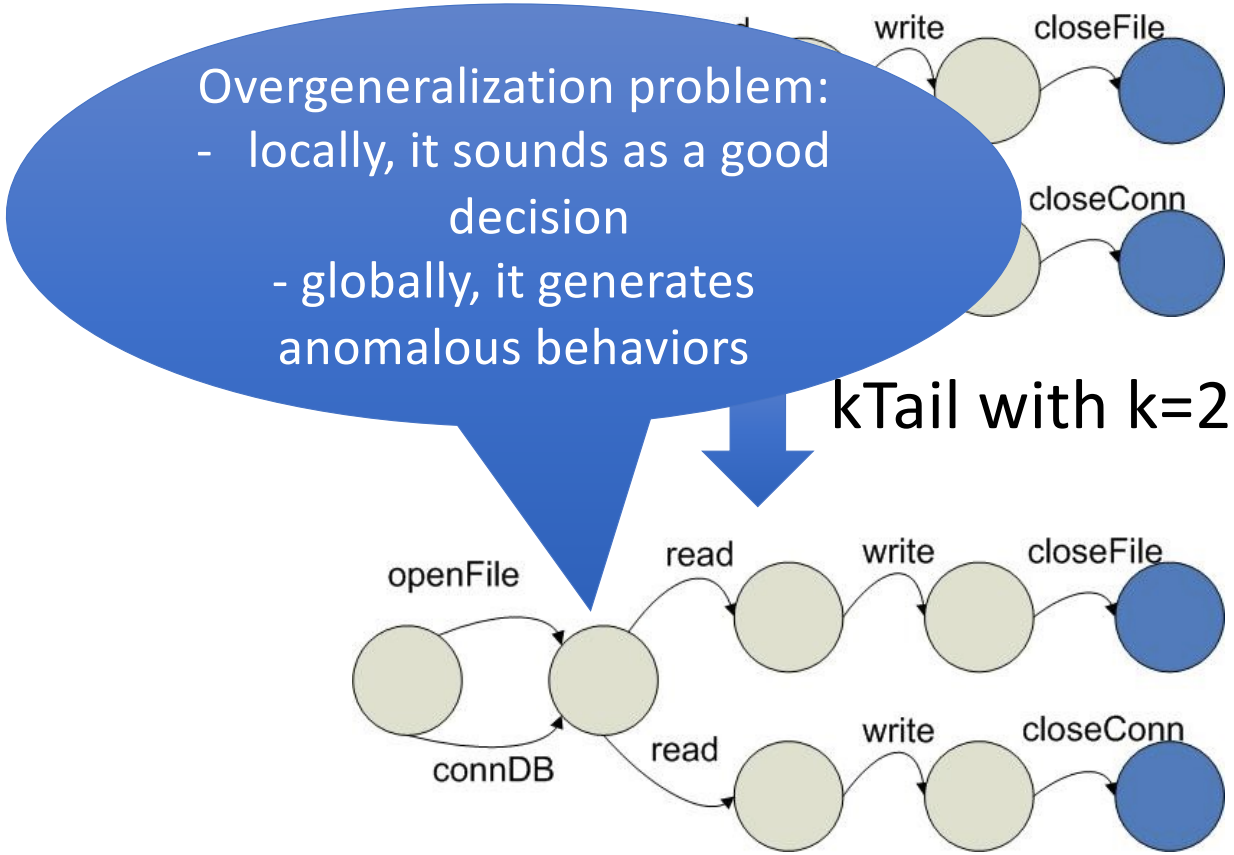
- Expressiveness depends on the set of the template expressions
  - More template expressions = more candidate expressions
  - But also higher computational cost
- See [Nguyen et al. ICSE 2012] for an approach to deal with polynomial and array expressions
- Web page: https://plse.cs.washington.edu/daikon/

Combined models

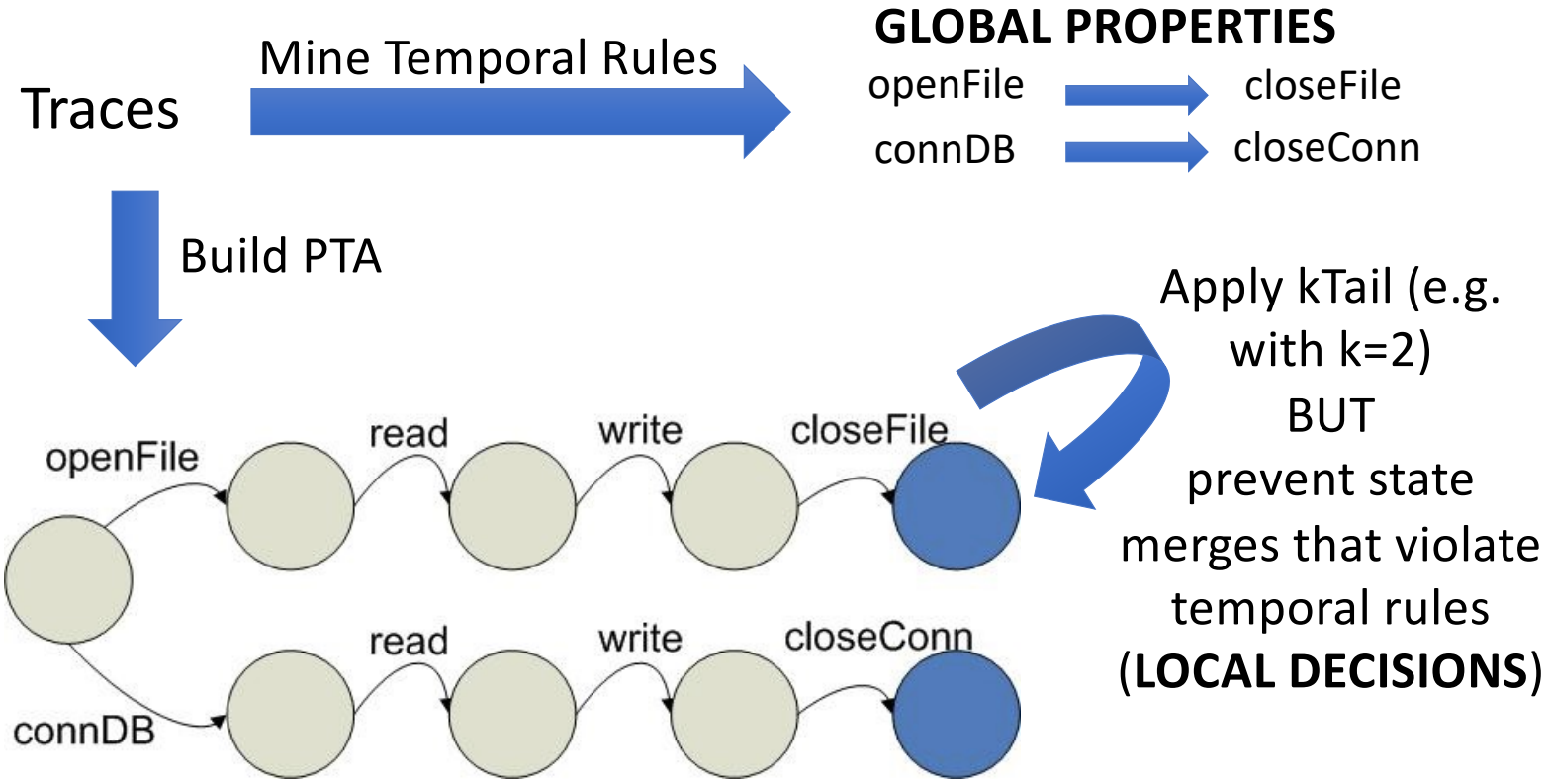
Combined models

- Constrained: k-Tail with steering, Synoptic
- Extended: GKTail, KLFA

The need for constrained models



Mine global properties, exploit them locally



[Lo, Mariani, Pezzè. Automatic Steering of Behavioral Model Inference. ESEC/FSE 2009]
[Schneider et al. Synoptic: Summarizing systems logs with refinement. SLAML 2010]

k-Tail with steering: Discussion

Application	Number of Events	k-Tail		k-Tail + steering		Overhead
		Prec	Recall	Prec	Recall	
X11 Win Library	356K	0.873	1	0.905	1	3%
CVS Client	2M	0.169	0.97	1	0.97	11%
WebSphere	9M	1	0.99	1	0.99	5%

Measure "the absence of illegal behaviors in the model"

Measure "the completeness of the model"

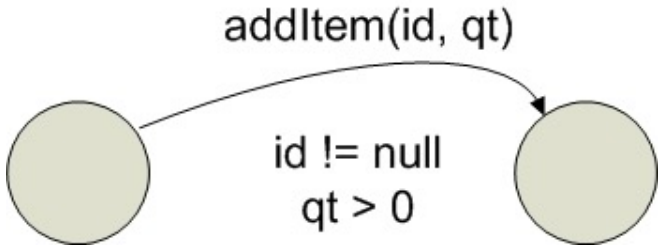
[Lo, Mariani, Pezzè. Automatic Steering of Behavioral Model Inference. ESEC/FSE 2009]

Download tools

- Synoptic: https://github.com/modelinference/synoptic
- (same repo as InvariMint)

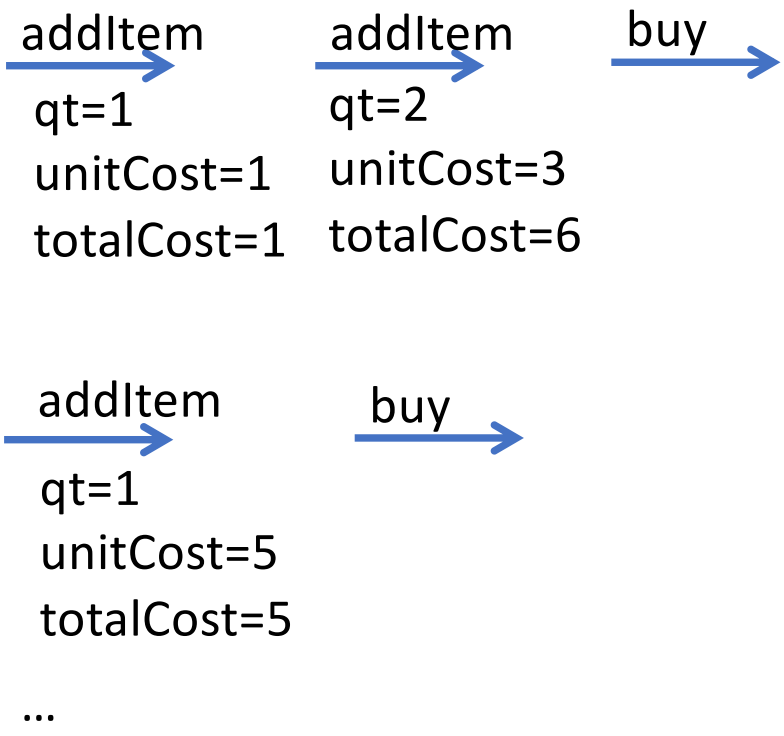
Extended FSA models

- GKTail: Adds information about the **ranges** of parameters values

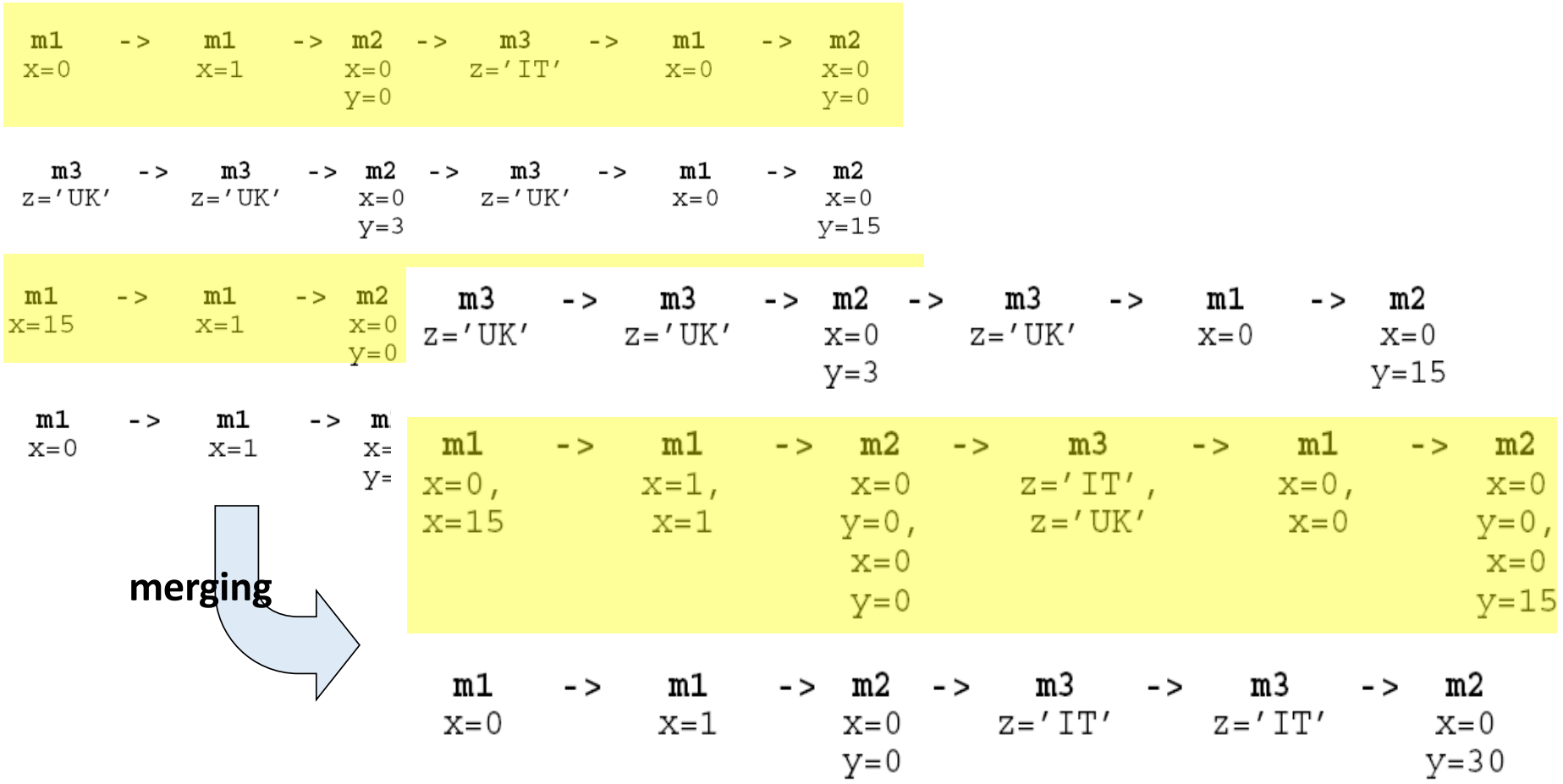


- KLFA: Adds information about the **recurrence** of parameters values

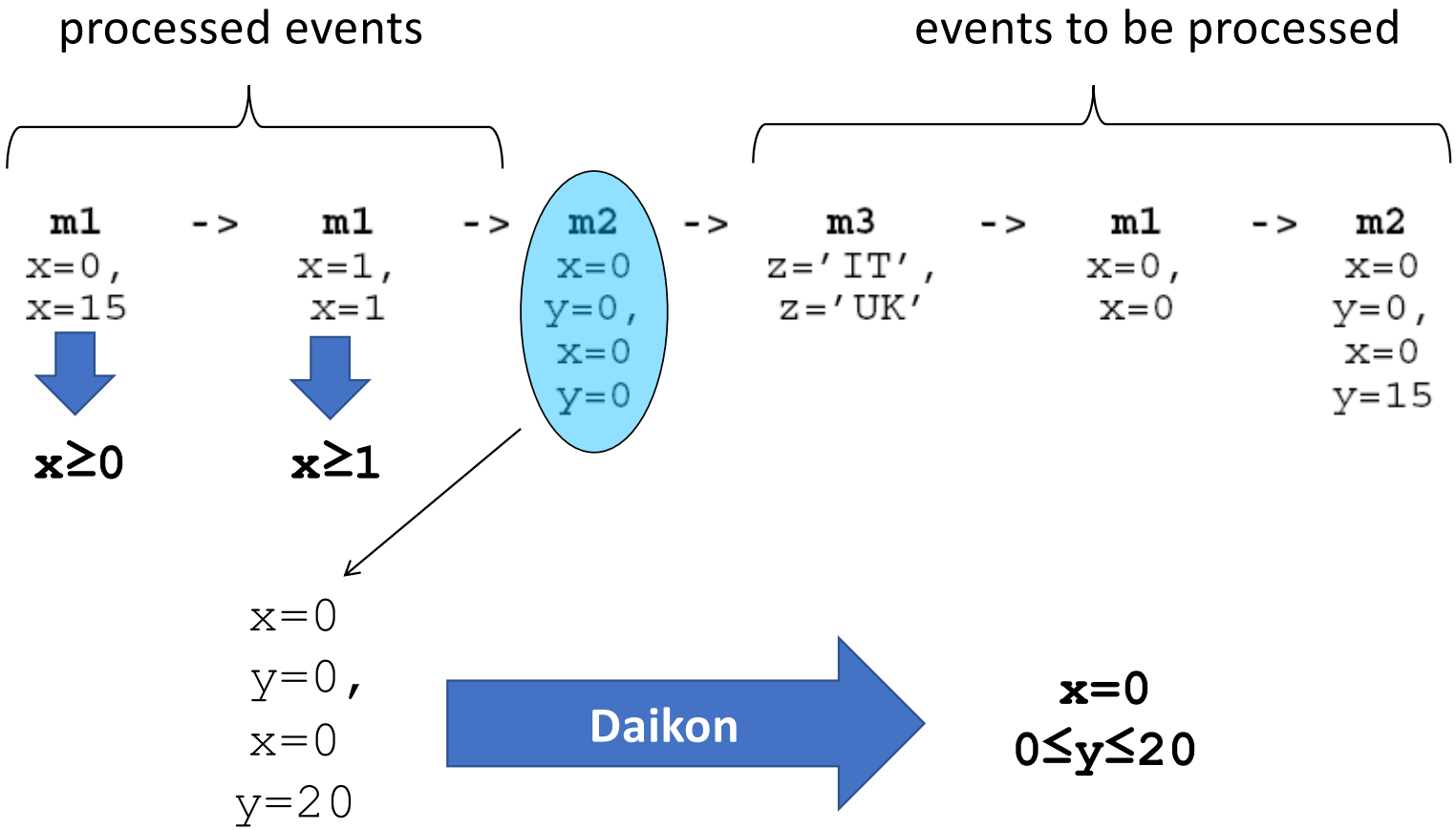
Trace with parameter values



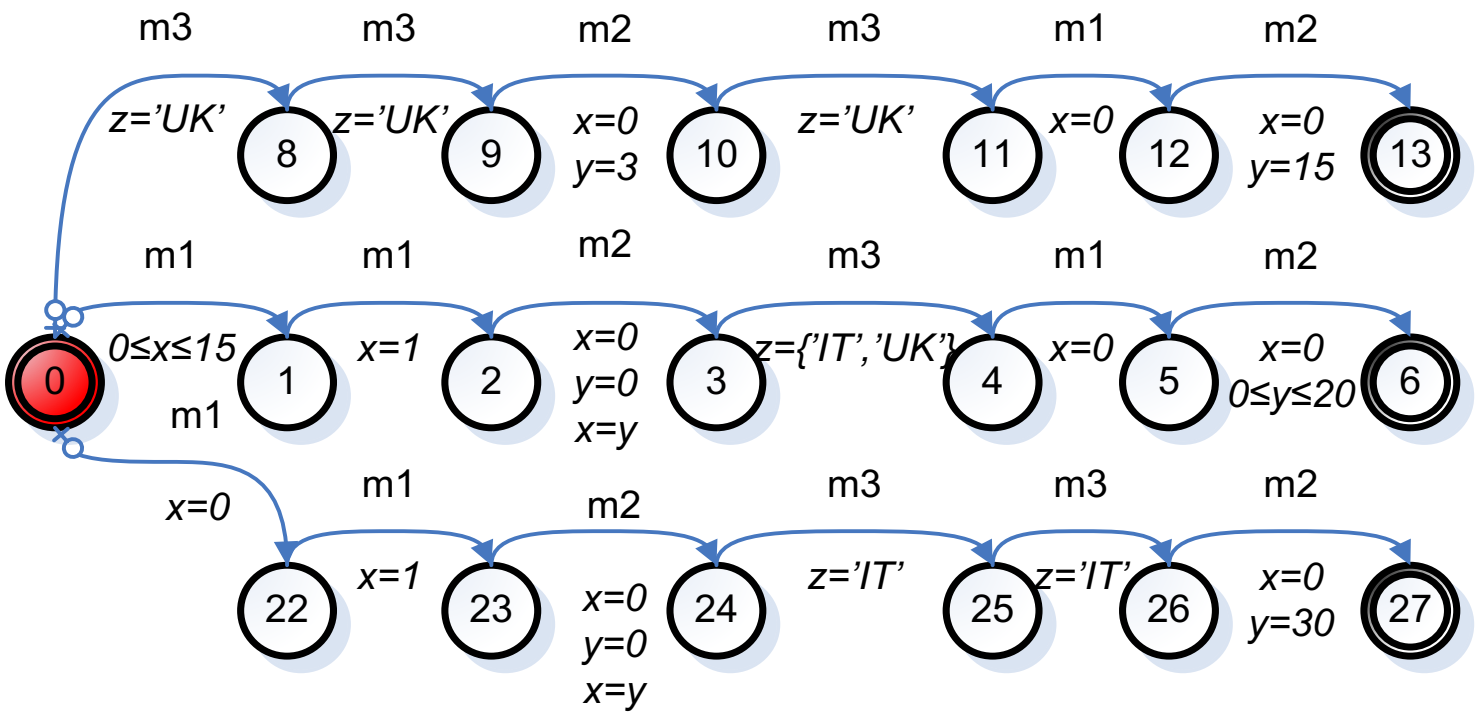
GKTail: Merging similar traces



GKTail: Deriving guards



GKTail: Synthesis of PTA (EFSM)

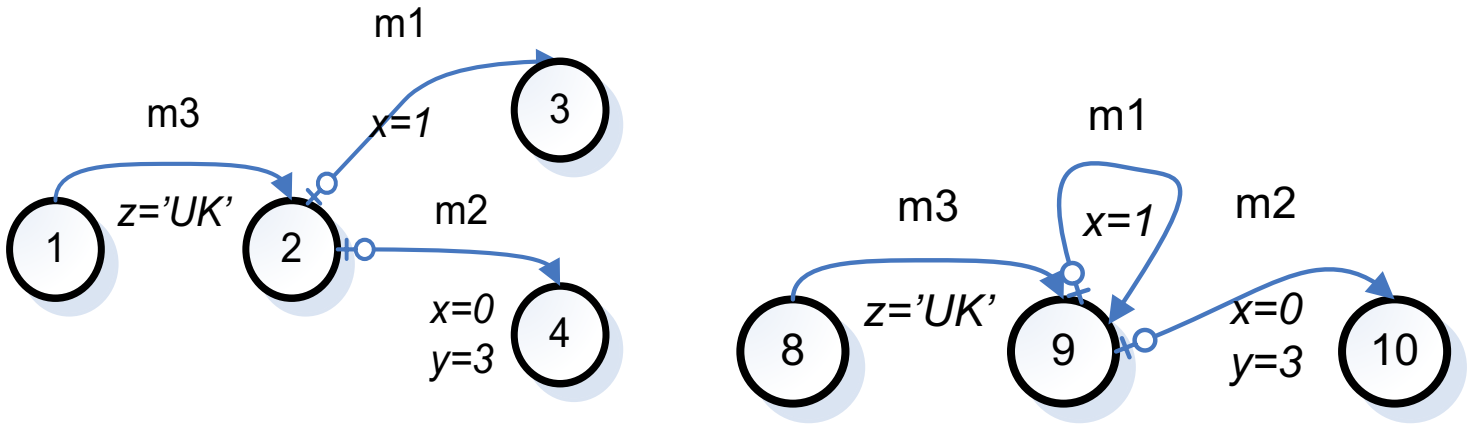


GKTail: State merging

- Still based on k-future
- Criteria:
  - Equivalence
  - Weak subsumption
  - Strong subsumption

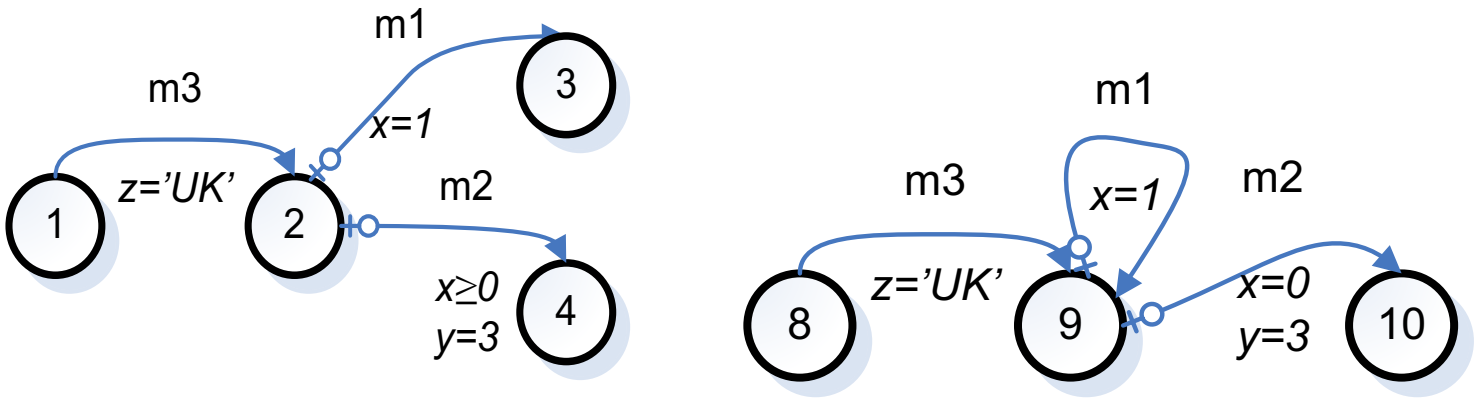
GKTail: State merging by equivalence

1 is 2-equivalent to 8



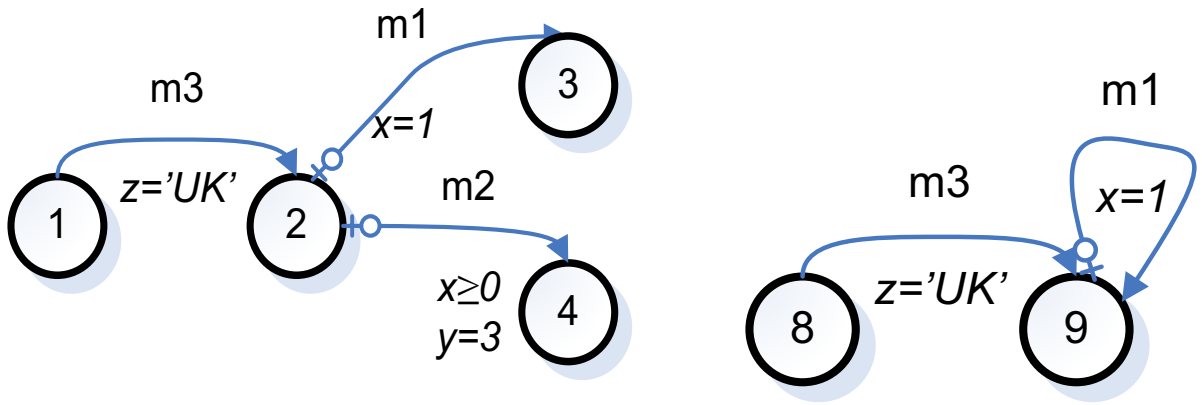
GKTail: State merging by weak subsumption

1 2-weakly-subsumes 8



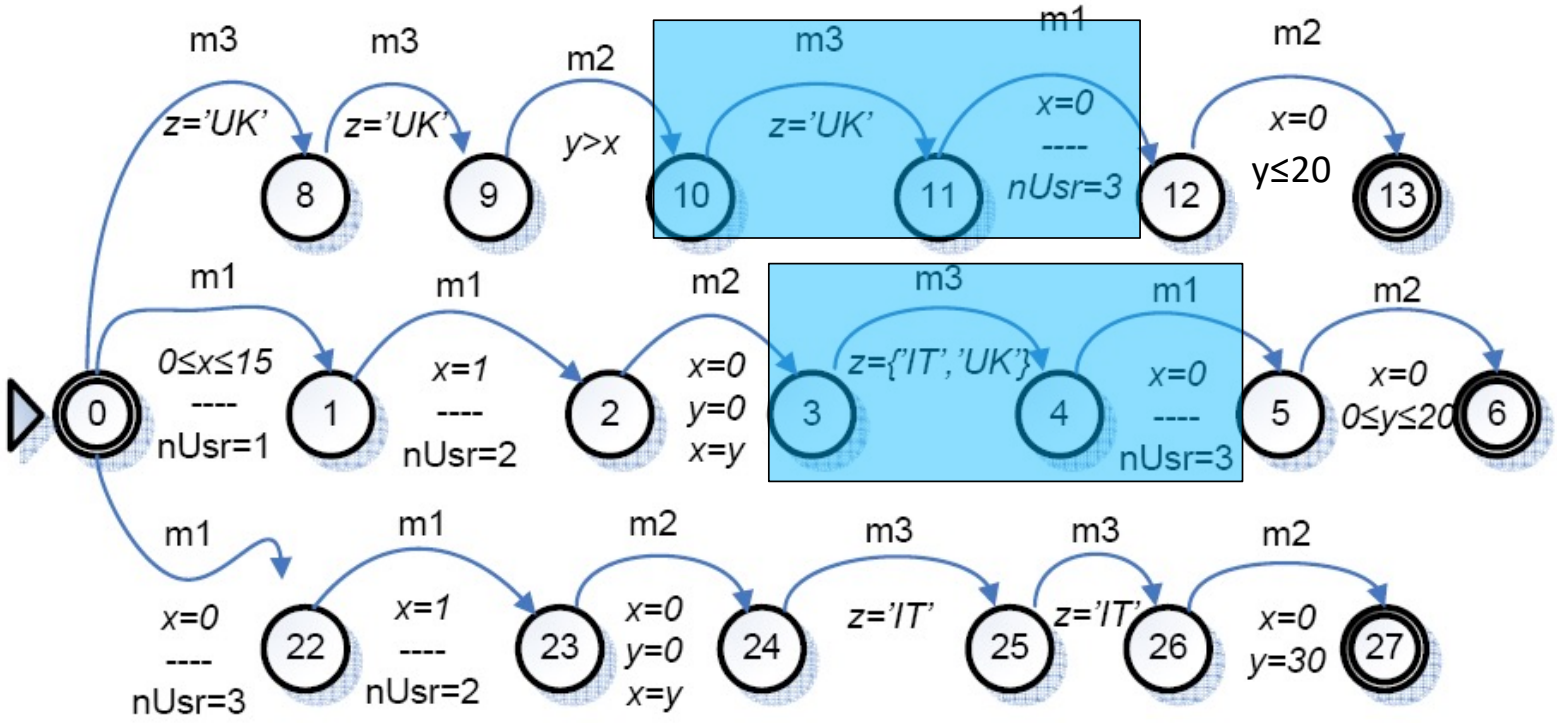
GKTail: State merging by strong subsumption

1 2-strongly-subsumes 8



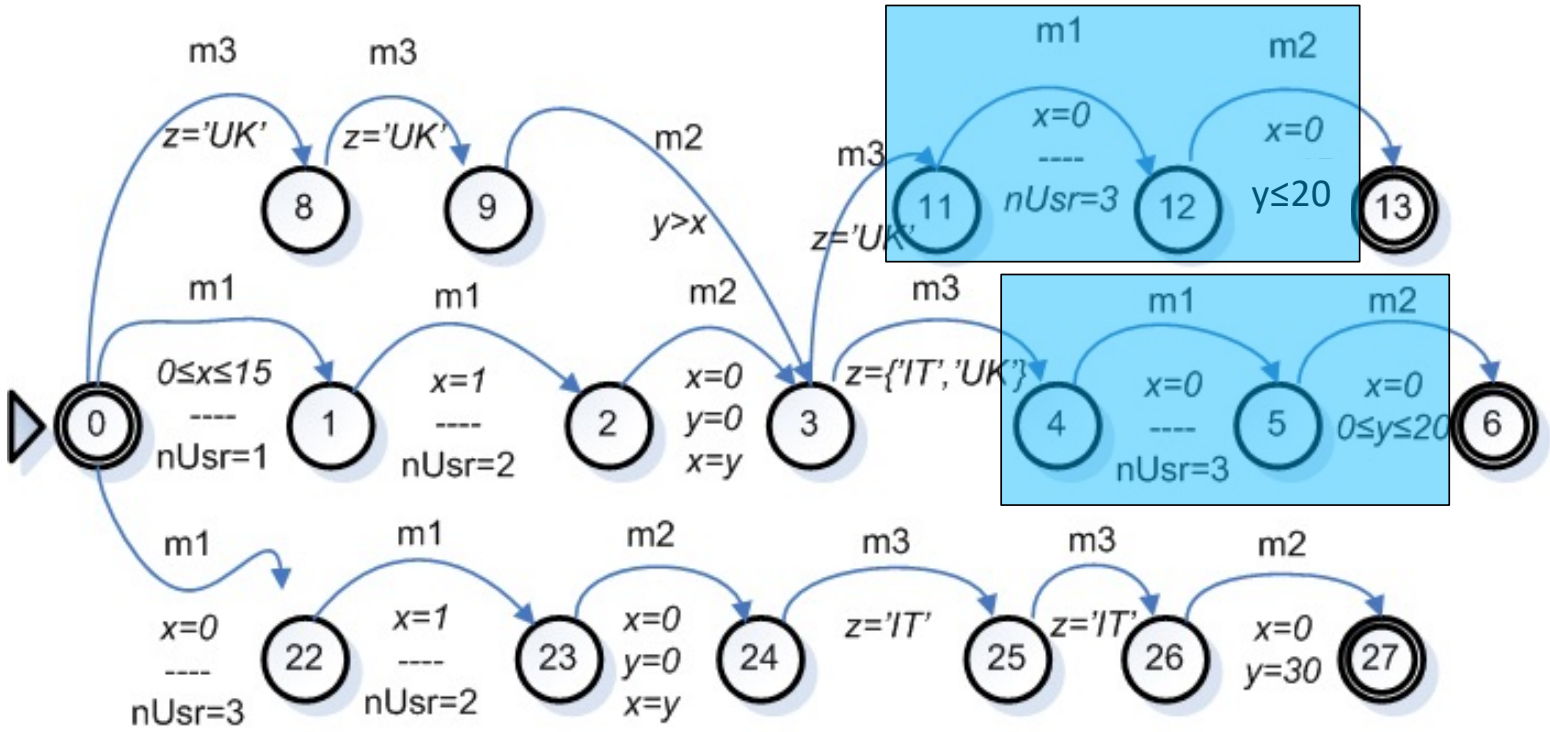
GKTail: Example

2-weak-subsumption

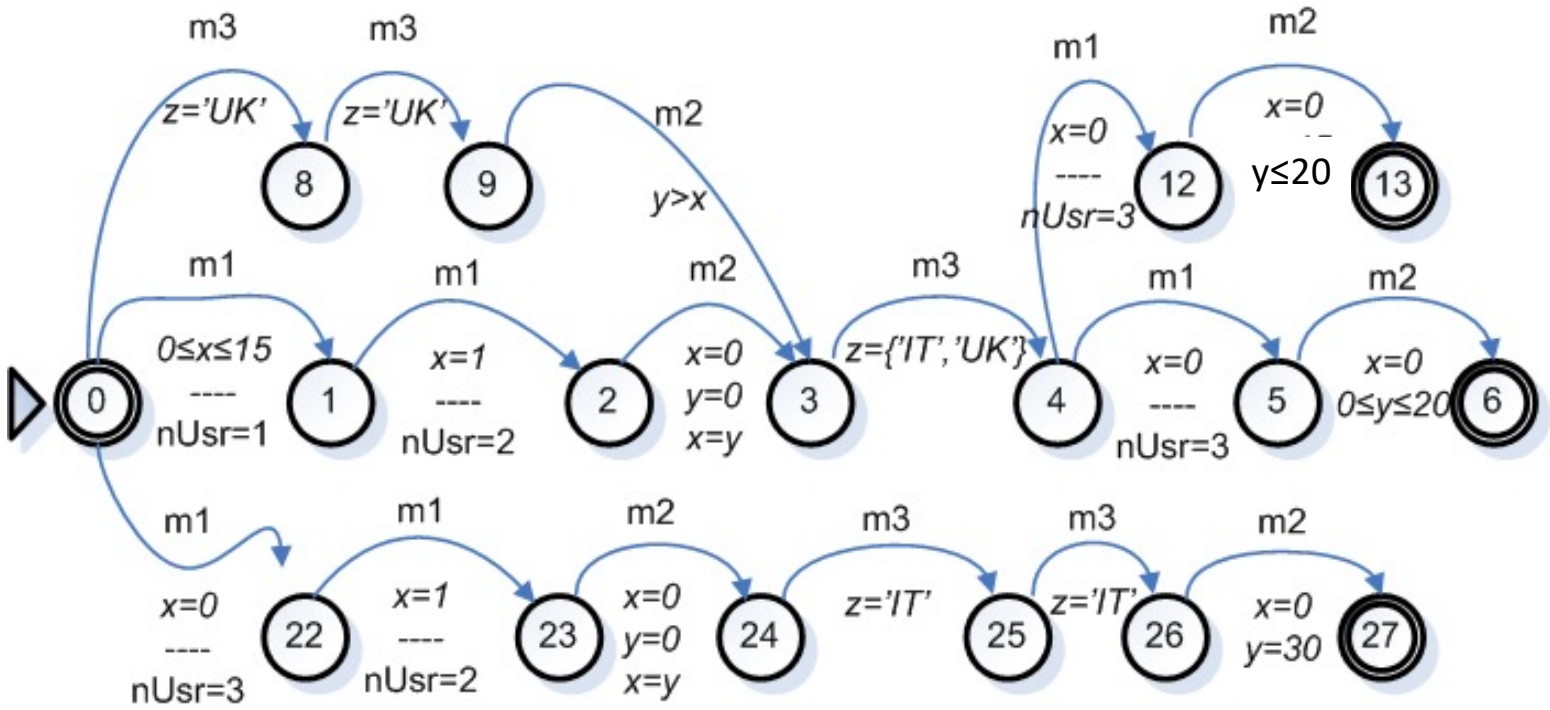


GKTail: Example

2-weak-subsumption

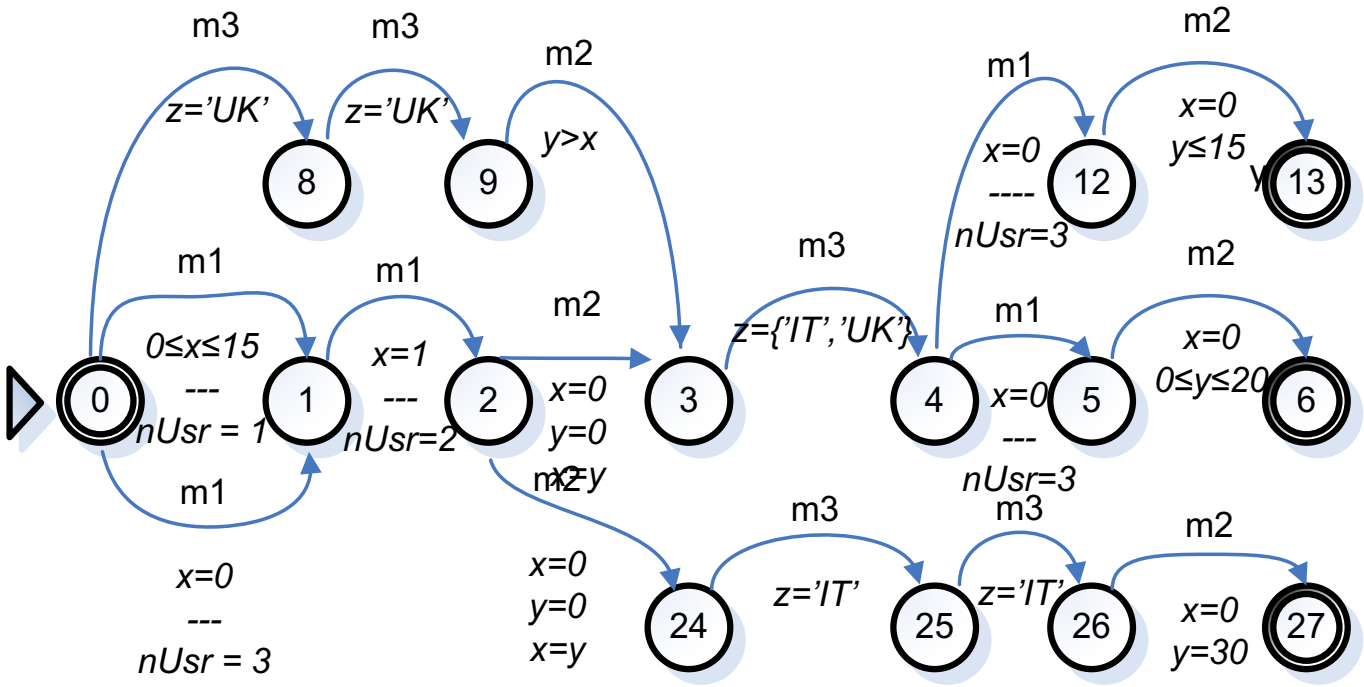


GKTail: Example



GKTail: Example

Result



KLFA: Rationale

Concrete values do not matter. How values repeat across events matters!!

Event name	**Thread id**	**Lock id**
takeLock	**28145**	0xd42e9a78
takeLock	**28145**	0xd11b33b1
relLock	**28145**	0xd11b33b1
relLock	**28145**	0xd42e9a78
takeLock	12130	0xd11b33b1
takeLock	12130	0xd42e9a78
…		

KLFA: Rationale

Concrete values do not matter. How values repeat across events matters!!

Event name	**Thread id**	**Lock id**
takeLock	28145	**0xd42e9a78**
takeLock	28145	0xd11b33b1
relLock	28145	0xd11b33b1
relLock	28145	**0xd42e9a78**
takeLock	12130	0xd11b33b1
takeLock	12130	**0xd42e9a78**
…		

Idea: rewrite event names taking recurrence into account

Event name	Thread id	Lock id
takeLock	**28145**	0xd42e9a78
takeLock	**28145**	0xd11b33b1
relLock	**28145**	0xd11b33b1
relLock	**28145**	0xd42e9a78
takeLock	12130	0xd11b33b1
takeLock	12130	0xd42e9a78
…		

takeLock_A_C
takeLock_A_D
relLock_A_D
relLock_A_C
takeLock_B_D
takeLock_B_C
…

How KLFA works

- Implement **several rewriting strategies** to capture different cases
- Implement an algorithm to **detect the best rewriting strategy** that must be applied to each chunk of trace file
- Reuse algorithms for regular inference (the publicly available implementation uses kBehavior)

[Mariani, Pastore. Automatic Identification of Failure Causes in System Logs, ISSRE, 2008]

Extended FSA models: Features

- GKTail = k-Tail + guards, KLFA = kBehavior + recurrences
- See [Lo, Mariani, Santoro, Learning extended FSA from Software: An Empirical Assessment. JSS, 2012] for an empirical comparison of four models (k-Tail, kBehavior, GKTail, KLFA) on 10 applications
- GKTail and KLFA will soon be available again through the LTA web page http://www.lta.disco.unimib.it/

What kind of model should we use?

The big picture

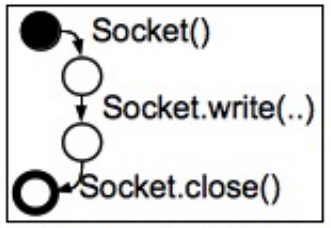
- Models of events, total order
  - Procedural
    - Trace-based mining
      - State-based merging: **k-Tail**
      - Behavior-based merging: **kBehavior**
    - State-based mining: **ADABU, ReAjax, Revolution**
  - Declarative: **InvariMint**
- Models of events, partial order: **Perracotta, Texada**
- Models of data: **Daikon**
- Combined models
  - Constrained: **k-Tail with steering, Synoptic**
  - Extended: **GKTail, KLFA**

Different models for different aspects

```
void sendData(Iterator data) {
  Properties = Properties.load( new File("config.xml") );

  Socket socket = new Socket(properties.get("port") );
  while( data.hasNext() ){
          socket.write( data.next() );
  }

  socket.close( )
}
```

```
Socket(int port)::ENTER
  port > 0

Socket(int port)::EXIT
  this.open = true;
```

Properties for Socket(int)

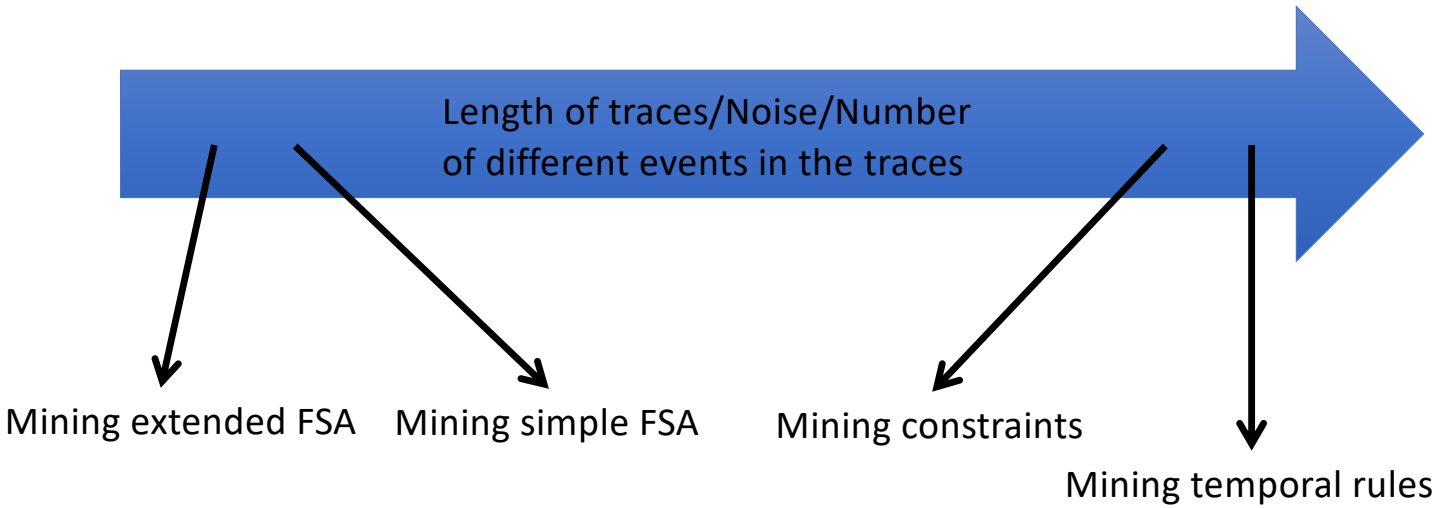


FSA for Socket class

```
new File ("config.xml") <alwaysPrecedes> new Socket
Socket.write()  <alwaysFollows> data.hasNext()
```

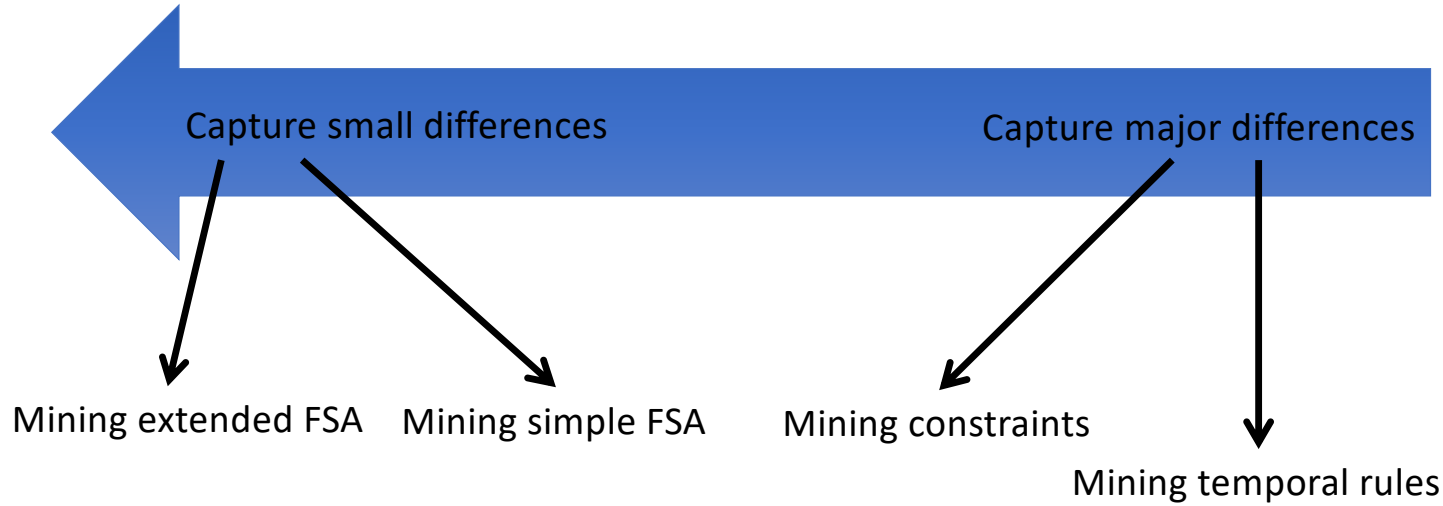
Temporal Rules

Empirical studies: Complexity



[Lo, Mariani, Santoro, Learning extended FSA from Software: An Empirical Assessment.
JSS, 2012]
[Yang, Evans, Bhardwaj, Bhat, Das. Perracotta: mining temporal API Rules from Imperfect Traces. ICSE. 2006]
[Nugyen, Marchetto, Tonella. Automated Oracles: An Empirical Study on Cost and Effectiveness, ESEC/FSE, 2013]

Empirical studies: Sensitivity



FSA good to analytically capture the behavior of small units (e.g., components)

Temporal rules and constraints good to capture some behaviors in relatively big applications

Take home

- Think to your research area
- If you need models and specifications…
- …and you do not have any,
- but you have a way of executing your software
- Specification Mining could be an option!