Living with change and uncertainty, and the quest for incrementality

27th CREST/DAASE Open Workshop
Dynamic Adaptive Automated Search Based Software Engineering
London, April 2013

Carlo Ghezzi
Politecnico di Milano
Deep-SE Group @ DEIB

Acknowledgements

- The work discussed here has been mostly developed thanks to a funding from the European Research Council (Advanced Grant IDEAS-ERC, Project 227977---SMScom)
- ...and thanks to

The vision

- **Software everywhere**
  - world fully populated by computationally rich devices (disappearing computer)
    - appliances, sensors/actuators, ... "things"
- **Cyber-physical systems**
  - built from and depending upon synergy of computational and physical components
- **Mobility and situation-awareness**
  - new behaviors emerge dynamically in a situation-dependent manner
- **Continuously running systems**
  - need to evolve while they offer service

The challenge

- **Continuous change, uncertainty**
  - in the requirements
  - in the environment
  - in the platform
    - cloud and service infrastructures
- **Dependability**
  - high assurance

★ Change and flexibility adversary of dependability

★ Can they be reconciled through a disciplined approach?

The questions

- Are the traditional software paradigms still valid?
- What is new or different?

… and the answers

- The way software is developed and run has to change quite radically
- We made development time agile/iterative, but this is not enough
- The traditional separation between development time and run time must be broken

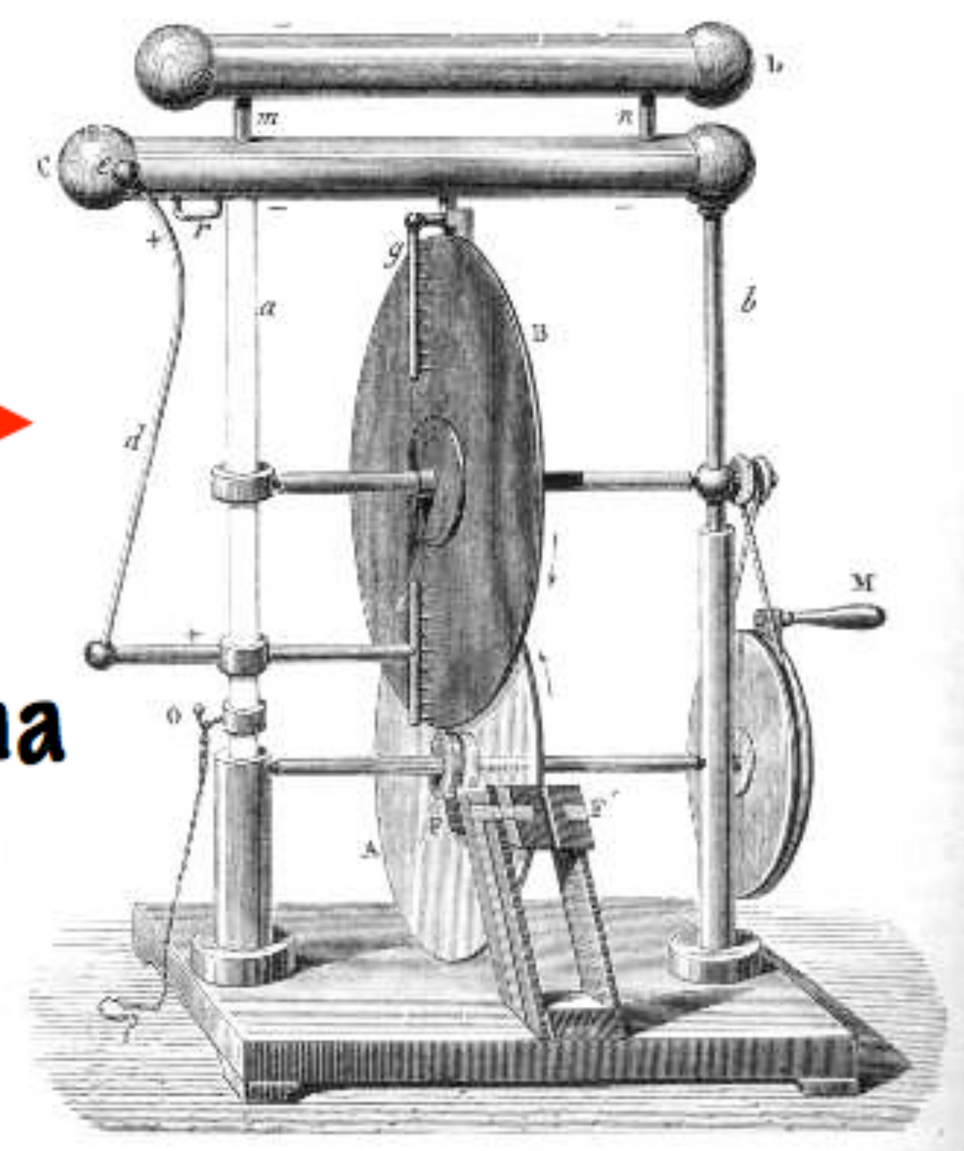
Understanding change and uncertainty

The global picture:
the *machine* and the *world*

World (the environment)

Shared phenomena

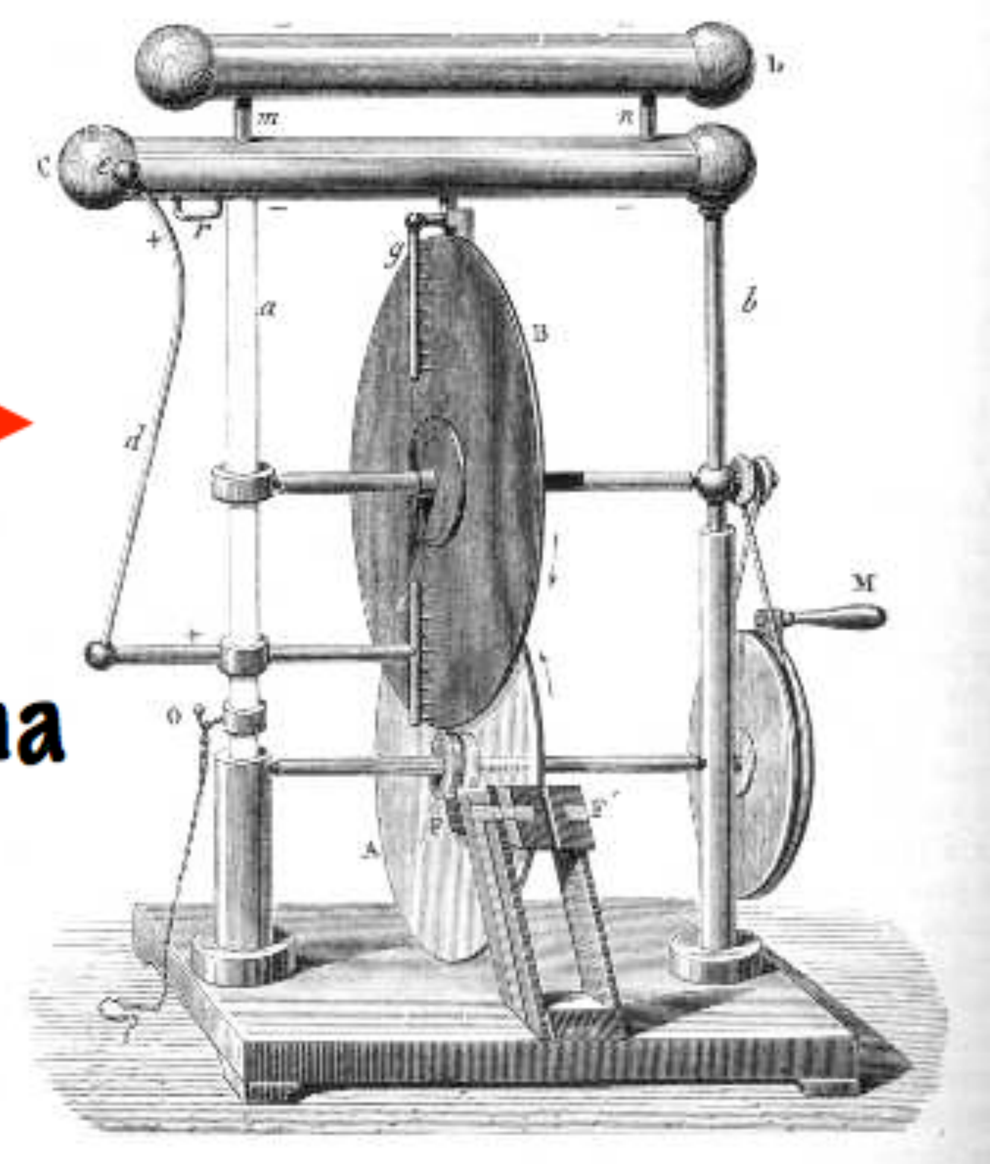
Machine



The global picture:
the *machine* and the *world*

World (the environment)

Machine



Shared phenomena

Goals
Requirements

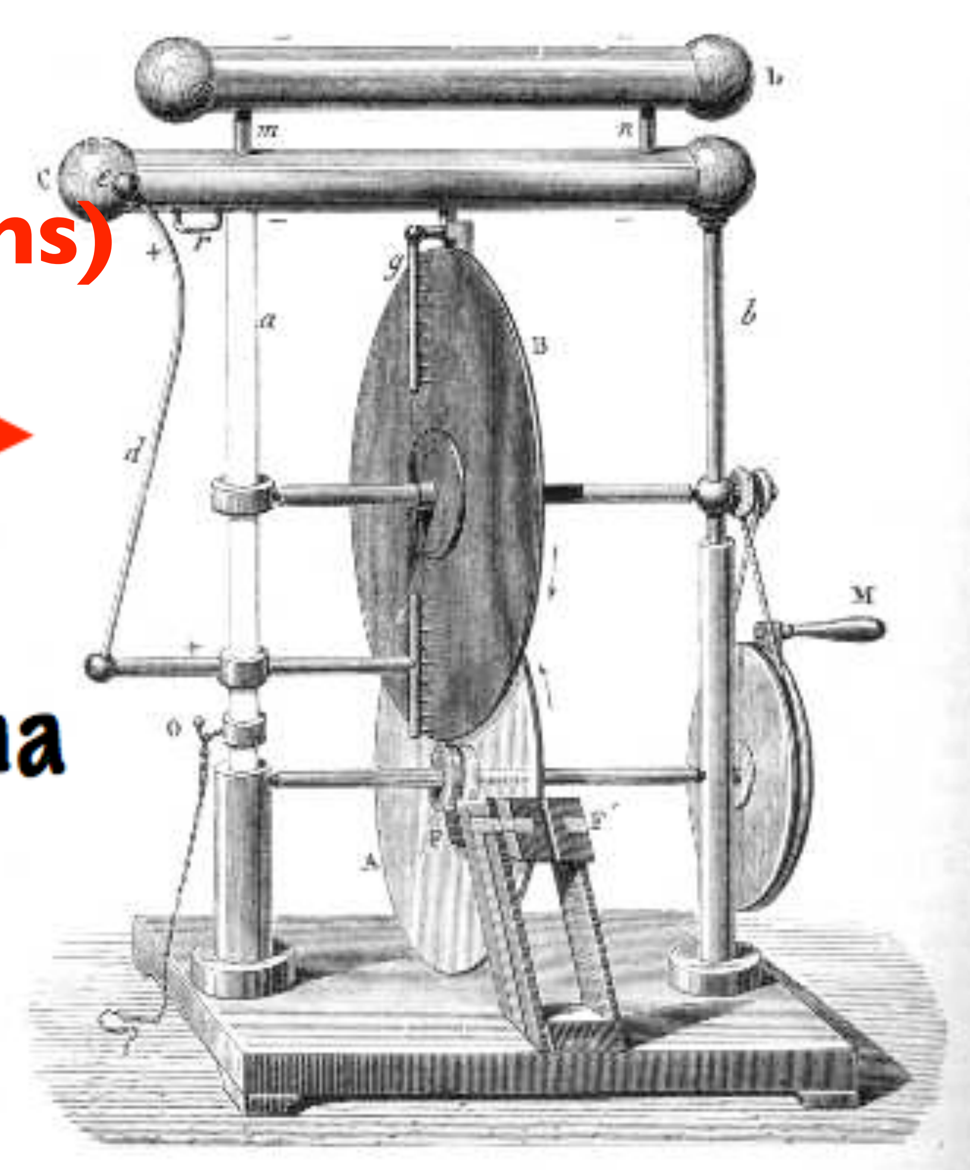
The global picture:
the *machine* and the *world*

World (the environment)

Machine

Domain properties (assumptions)

Shared phenomena



Goals
Requirements

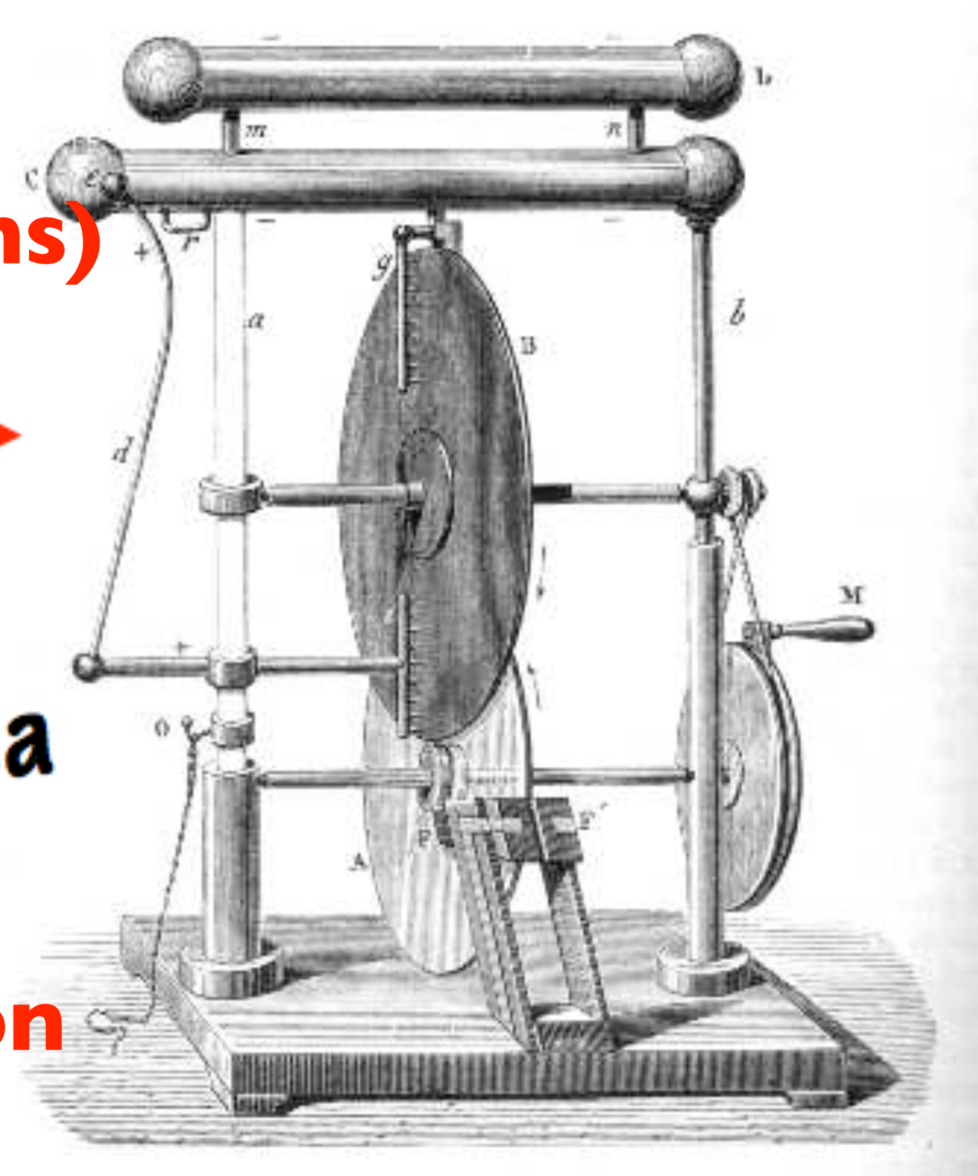
The global picture:
the *machine* and the *world*

Self Managing Situated Computing

World (the environment)

Machine

Domain properties (assumptions)



Shared phenomena

Goals
Requirements

Specification

Domain assumptions

May concern

- usage profiles
- users' responsiveness
- remote servers response time
- network latency
- sensors/actuators behaviors
- . . .

"Domain assumptions bridge the gap between requirements and specifications"
(M. Jackson & P. Zave)

Change and uncertainty

Dependability arguments

- Assume you have a formal representation for
  - R = requirements
  - S = specification
  - D = domain assumptions

if S and D are both satisfied and consistent, it is necessary to prove

- S, D |= R

The role of (formal) models

- The formal representations of D and S are often given in terms of models (e.g., state machines)
- Dependability arguments are based on proofs that the models satisfy R
- For example, model checking may be used at **design time** to assess dependability

Cyber-physical systems

- Network of computational elements interacting with environment
- Changes/uncertainty in goals/requirements
- Changes/uncertainty in domain assumptions
  - concerning the physical environment
  - other computational elements
- Some changes can be anticipated, others cannot
- Changes lead to *software evolution*
- *Adaptation* (as a special case of evolution)
  - the system can self-organize its reaction to anticipated changes (in environment)

Changes may affect dependability

$S, D \models R$

- Changes may concern
  - R
  - D (our focus here)
- We can decompose D into D_f and D_c
  - – D_f is the fixed/stable part
  - – D_c is the changeable part

We need to **detect changes to D_c**

and **make changes to S** (and to the implementation) to keep satisfying **R**

Terminology

- **Changes** are the perceived difference between the expected behavior and the materialized behavior
- They cause an **evolution** in the software to keep the requirements satisfied
- **(Self-)adaptation**: evolution is self-managed by the software (also, on-line adaptation)
- It may require human intervention (**maintenance**) to evolve the software off-line
- It may still be possible to install off-line modifications while the application is running

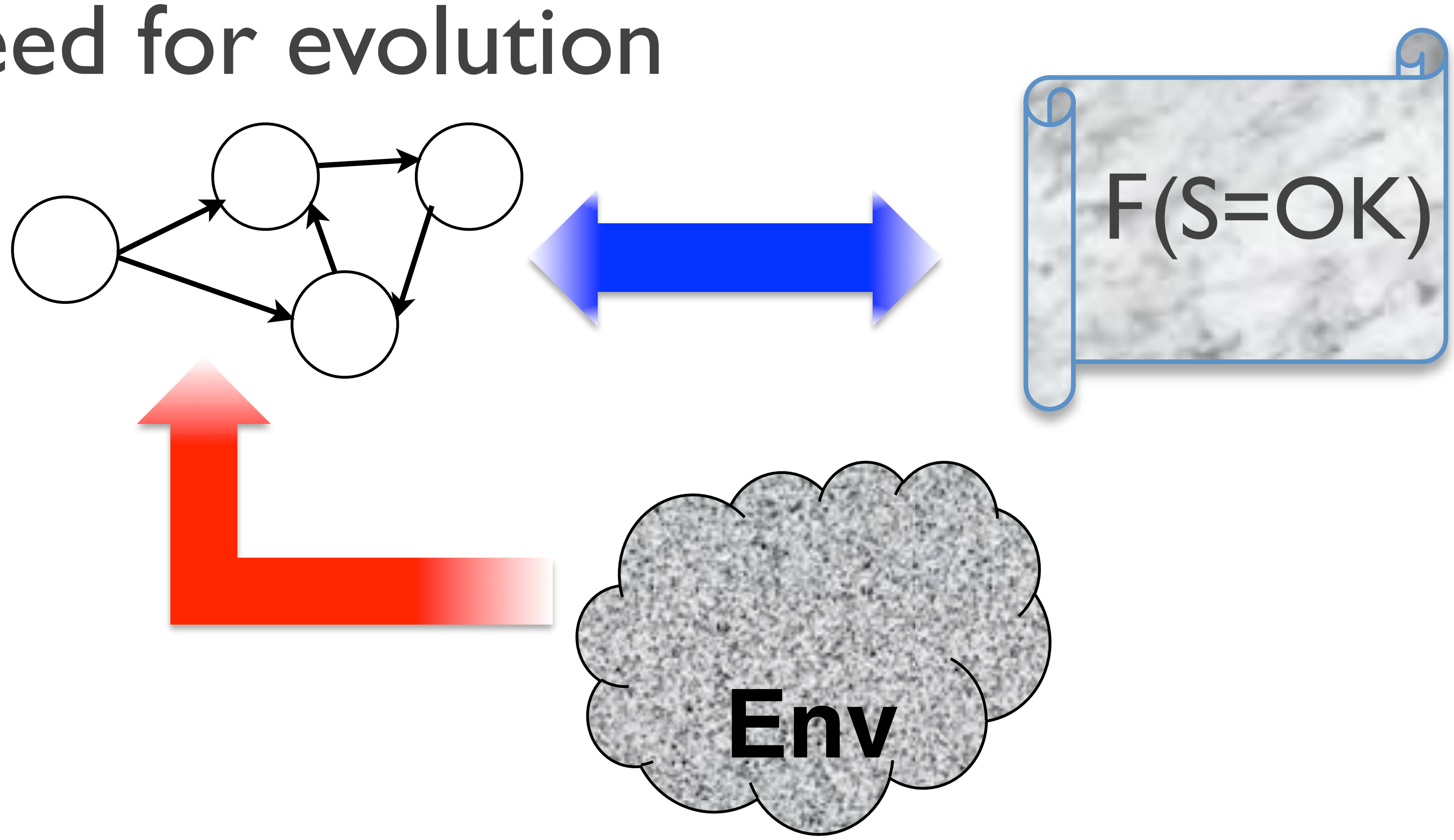
Software evolution revisited

- Software evolution recognized as a crucial problem since the 1970's (work by L. Belady and M. Lehman)
  - they proposed "laws" of sw evolution
- Viewed as a problem to be managed off-line and mainly as a software development problem
- What is new here
  – the unprecedented intensity of change/uncertainty
  – the request that software responds to changes while the system is running, possibly in a self-managed manner

Software paradigm shift

Self Managing Situated Computing

- Conventional separation between development time and run time is blurring
- Models + requirements need to be kept + updated at run time (systems need to be *reflective*)
- Continuous verification must be performed to detect the need for evolution



R. Calinescu, C. Ghezzi, M. Kwiatkokwska, R. Mirandola, “Self-adaptive software needs quantitative verification at runtime”, Comm. ACM, Sept. 2012

Paradigm shift: rethink run-time environments

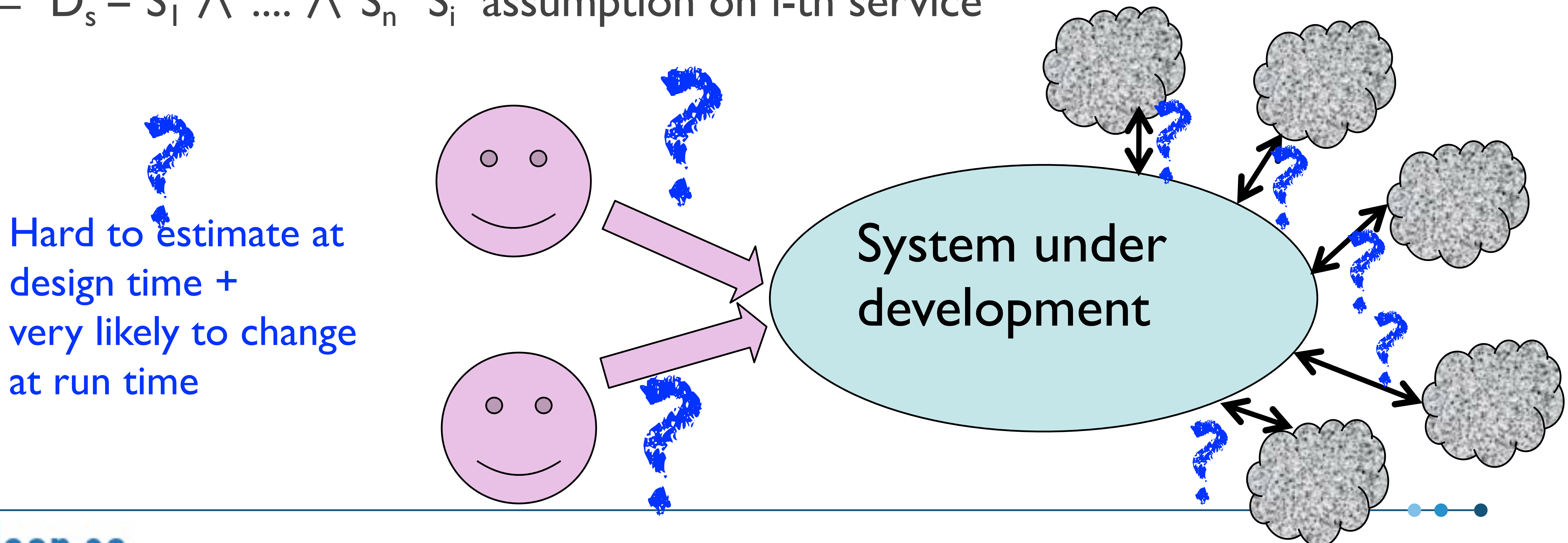
- Traditionally software engineering mostly concerned with development time, clearly separated from run time
- The result is **code** that simply needs to be **run**
- *(Self-)adaptive software requires much more*
  - *the deployed software must be reflective, able to reason about itself and the environment*
    - ✓ *models*
    - ✓ *goals and requirements*
    - ✓ *strategies*
  - ***agility becomes a run-time objective***

KAMI: a framework for self-adaptation

- [ICSE 2009] I. Epifani, C. Ghezzi, R. Mirandola, G. Tamburrelli, "Model Evolution by Run-Time Parameter Adaptation"
- [RE 2009] C. Ghezzi, G. Tamburrelli, "Reasoning on Non Functional Requirements for Integrated Services"
- [FSE 2010] I. Epifani, C. Ghezzi, G. Tamburrelli, "Change-Point Detection for Black-Box Services"

Specific focus

- **Non-functional** requirements
  - reliability, performance, energy consumption, cost, …
- Quantitatively stated in **probabilistic** terms
- D_c decomposed into D_u , D_s
  - D_u = usage profile
  - $D_s = S_1 \wedge .... \wedge S_n$ S_i assumption on i-th service

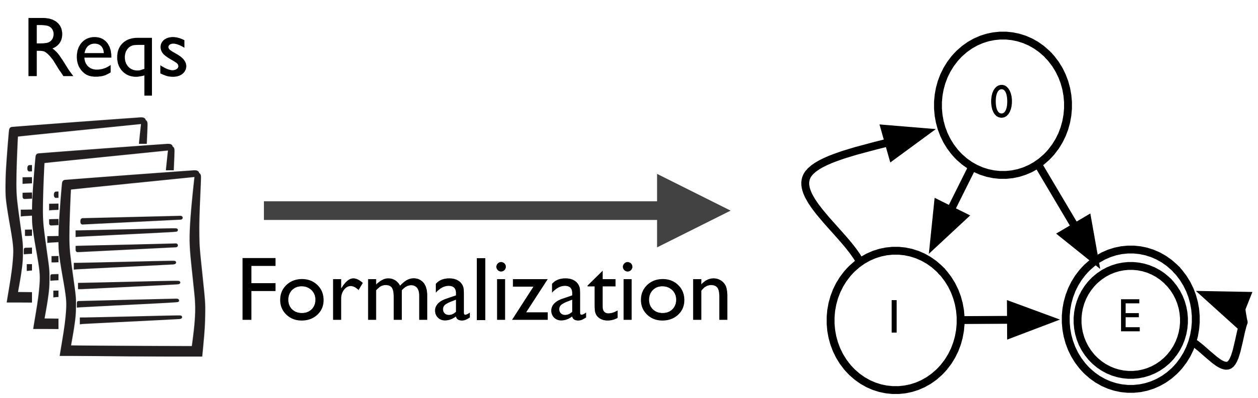


Our approach in a nutshell

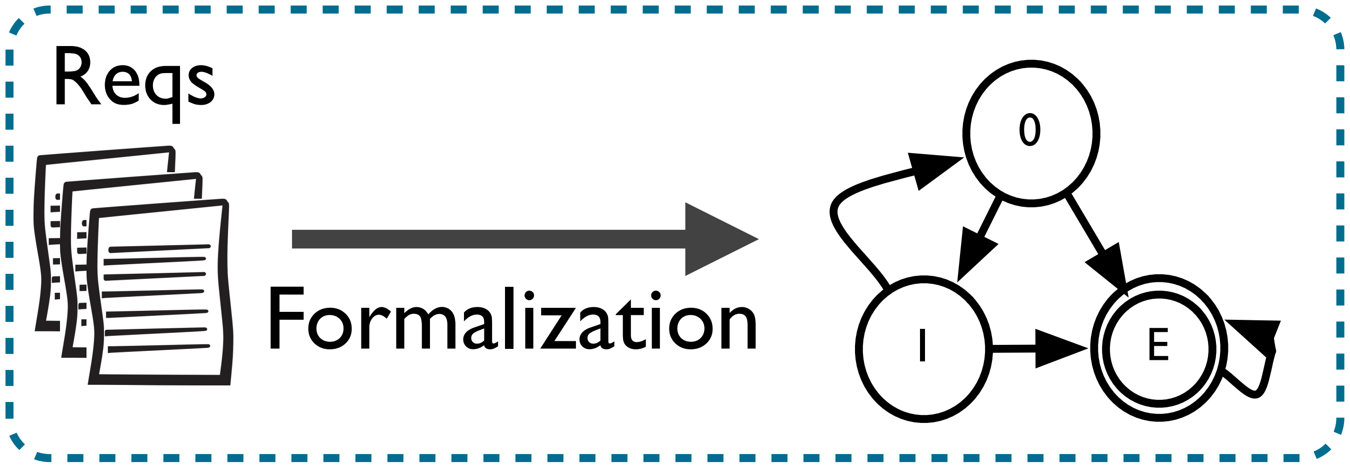
Our approach in a nutshell

Reqs

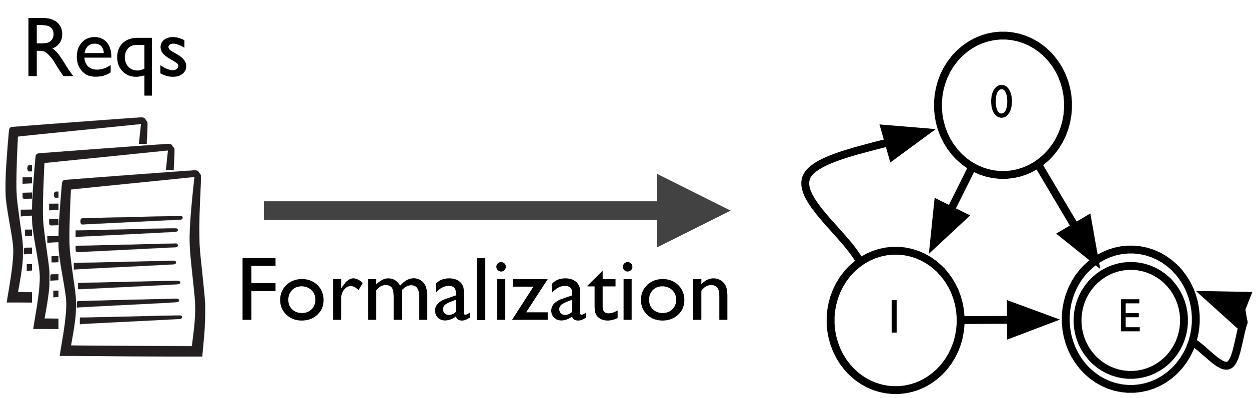
Our approach in a nutshell



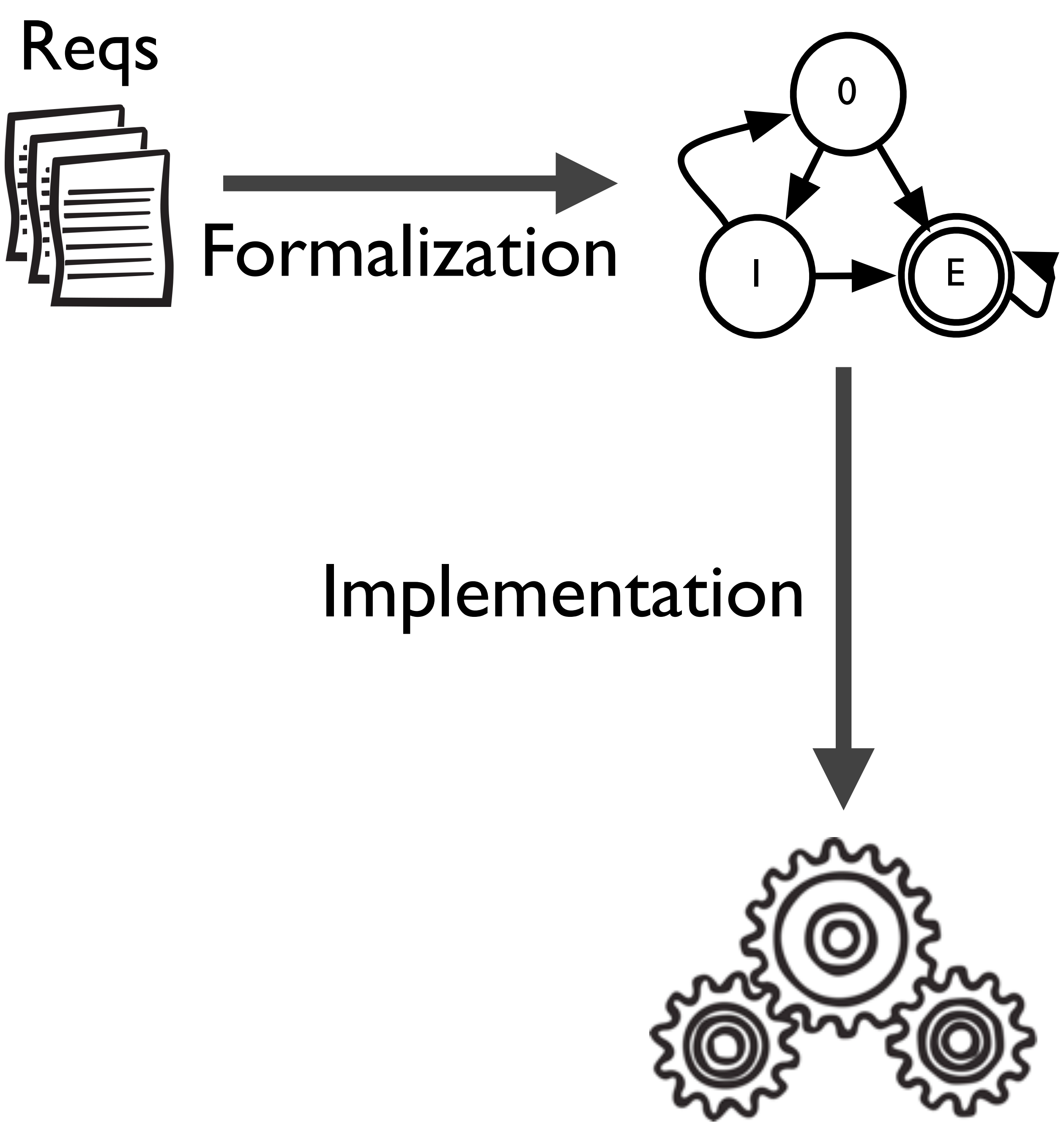
Our approach in a nutshell



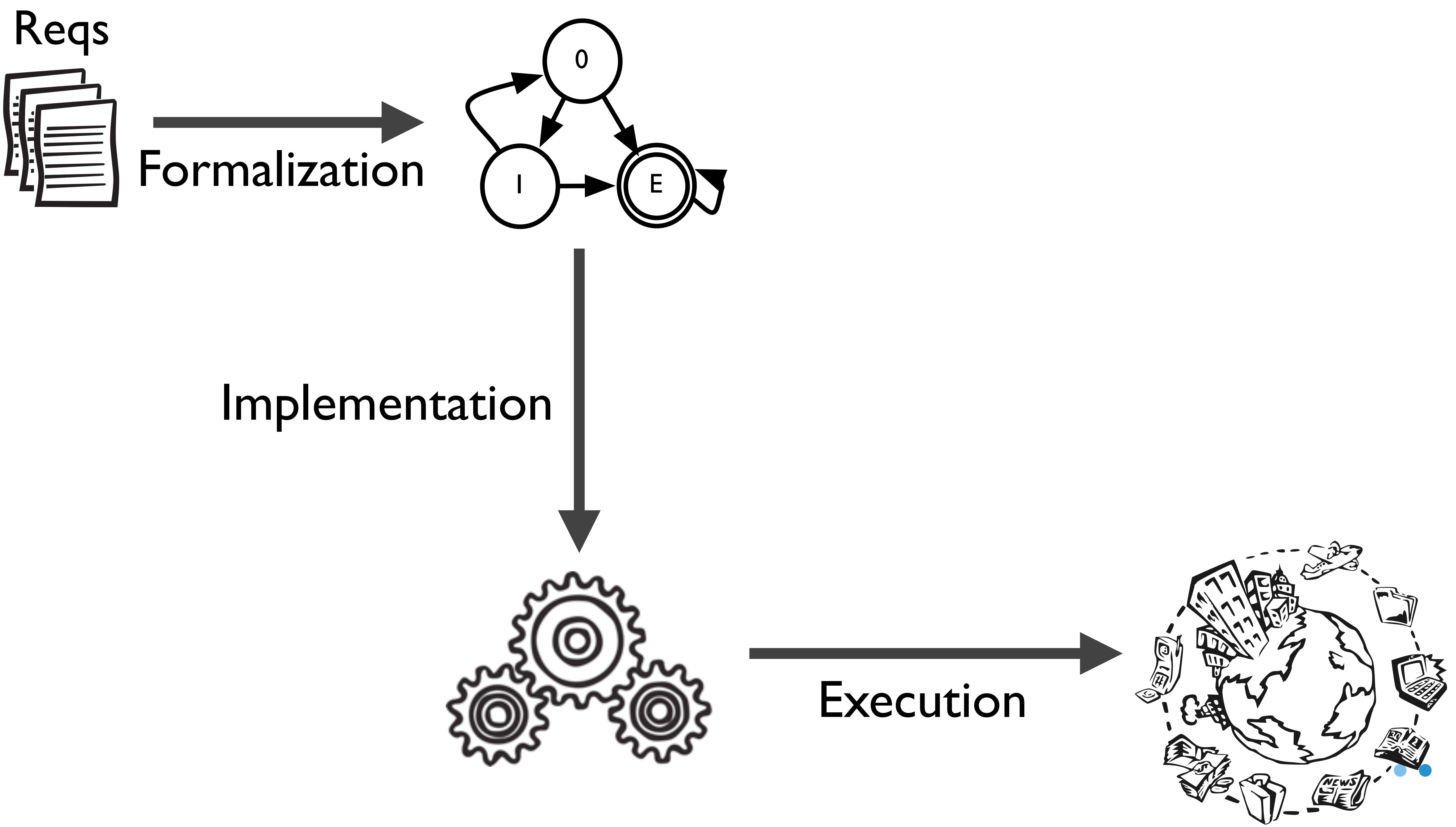
Our approach in a nutshell



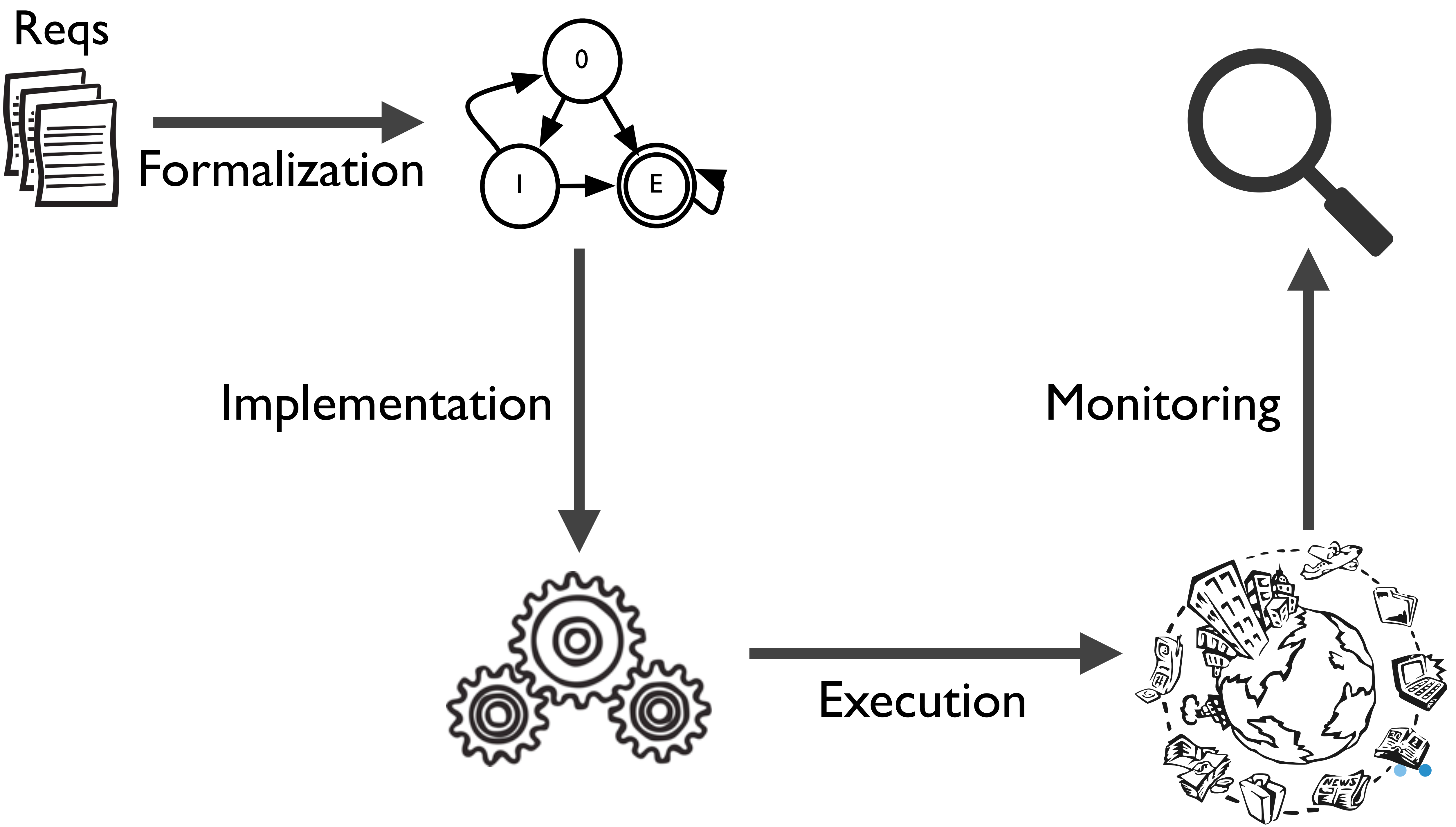
Our approach in a nutshell



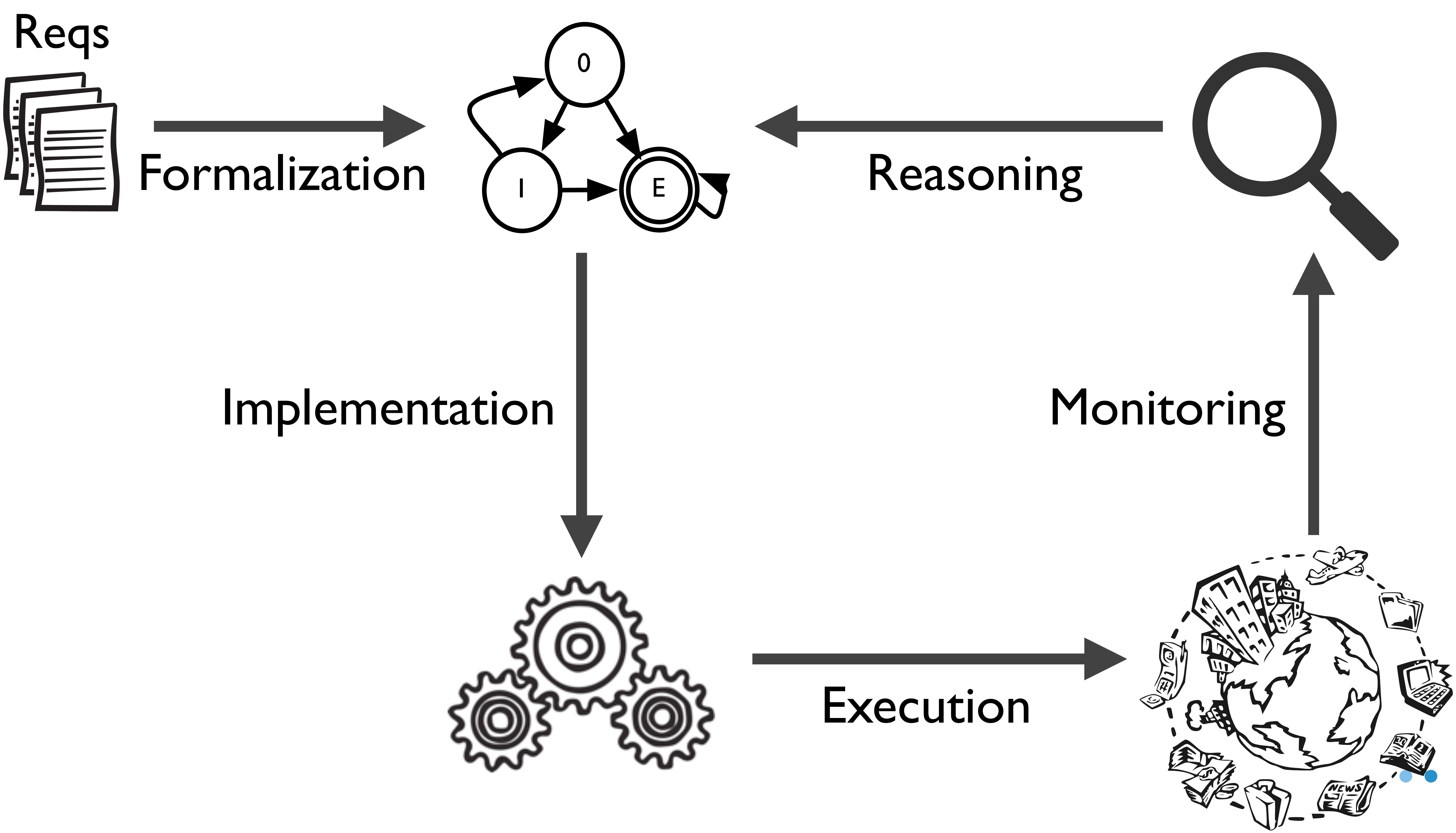
Our approach in a nutshell



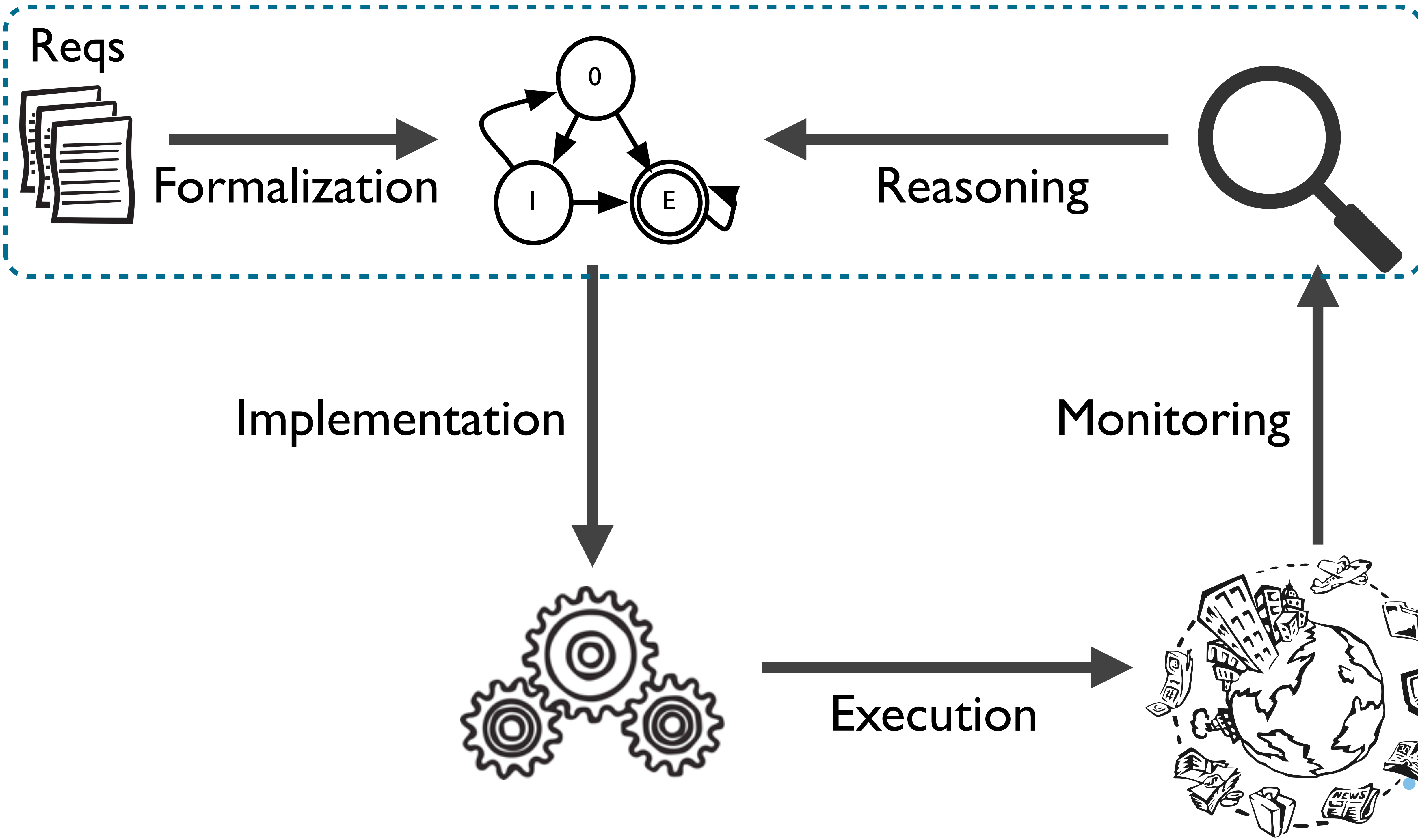
Our approach in a nutshell



Our approach in a nutshell



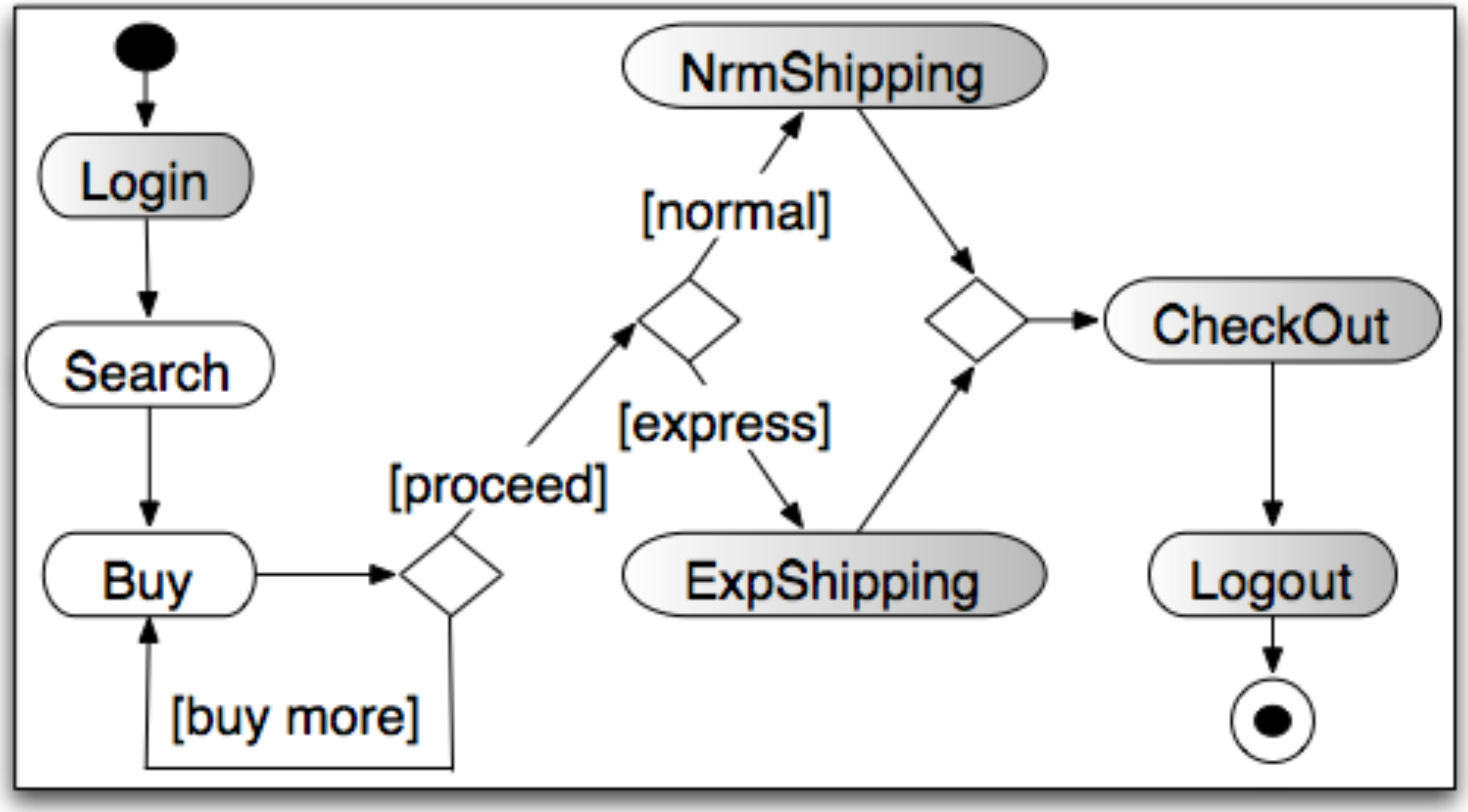
Our approach in a nutshell



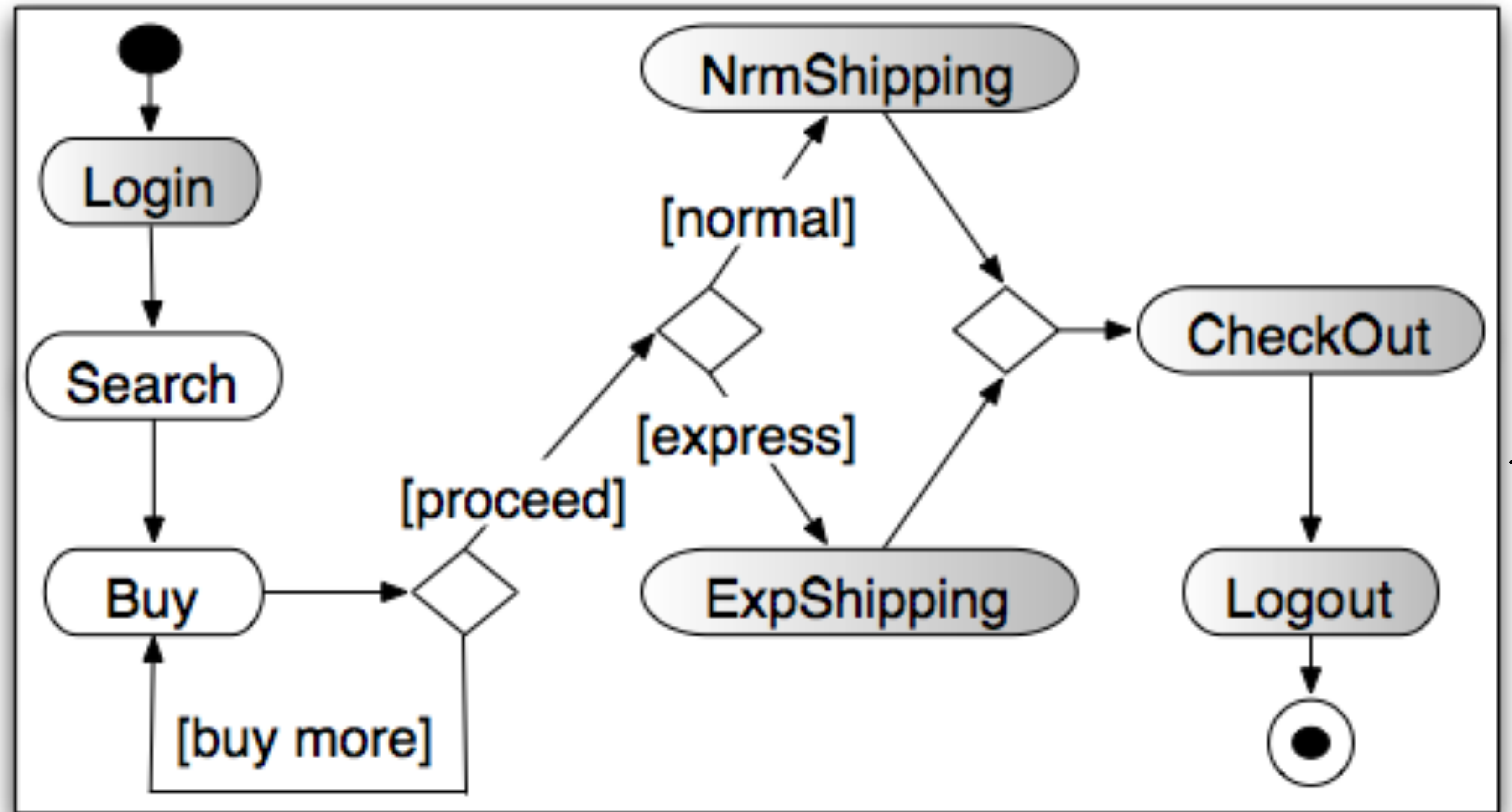
Models

- Different models provide different **viewpoints** from which a system can be analyzed
- Focus on **non-functional** properties and quantitative ways to deal with uncertainty
- Use of **Markov models**
  - DTMCs for reliability
  - CTMCs for performance
  - Reward DTMCs for energy/cost/...

Using verification for change detection and adaptation

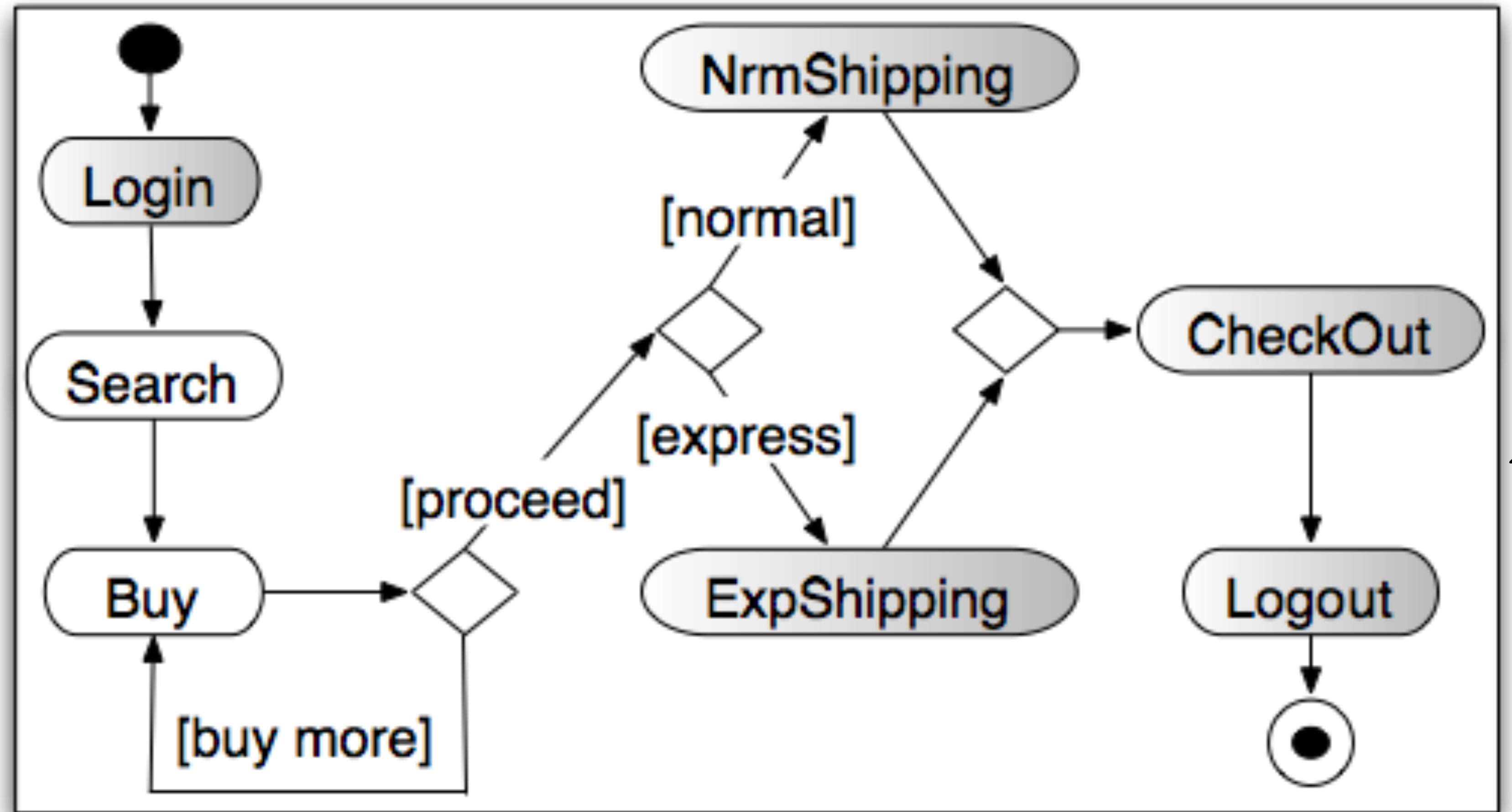


Using verification for change detection and adaptation



Users classified as BigSpender or SmallSpender based on their usage profile.

Using verification for change detection and adaptation



Users classified as BigSpender or SmallSpender based on their usage profile.

3 probabilistic requirements:

R1: "Probability of success is > 0.8"

R2: "Probability of a ExpShipping failure for a user recognized as BigSpender < 0.035"

R3: "Probability of an authentication failure is less then < 0.06"

Assumptions

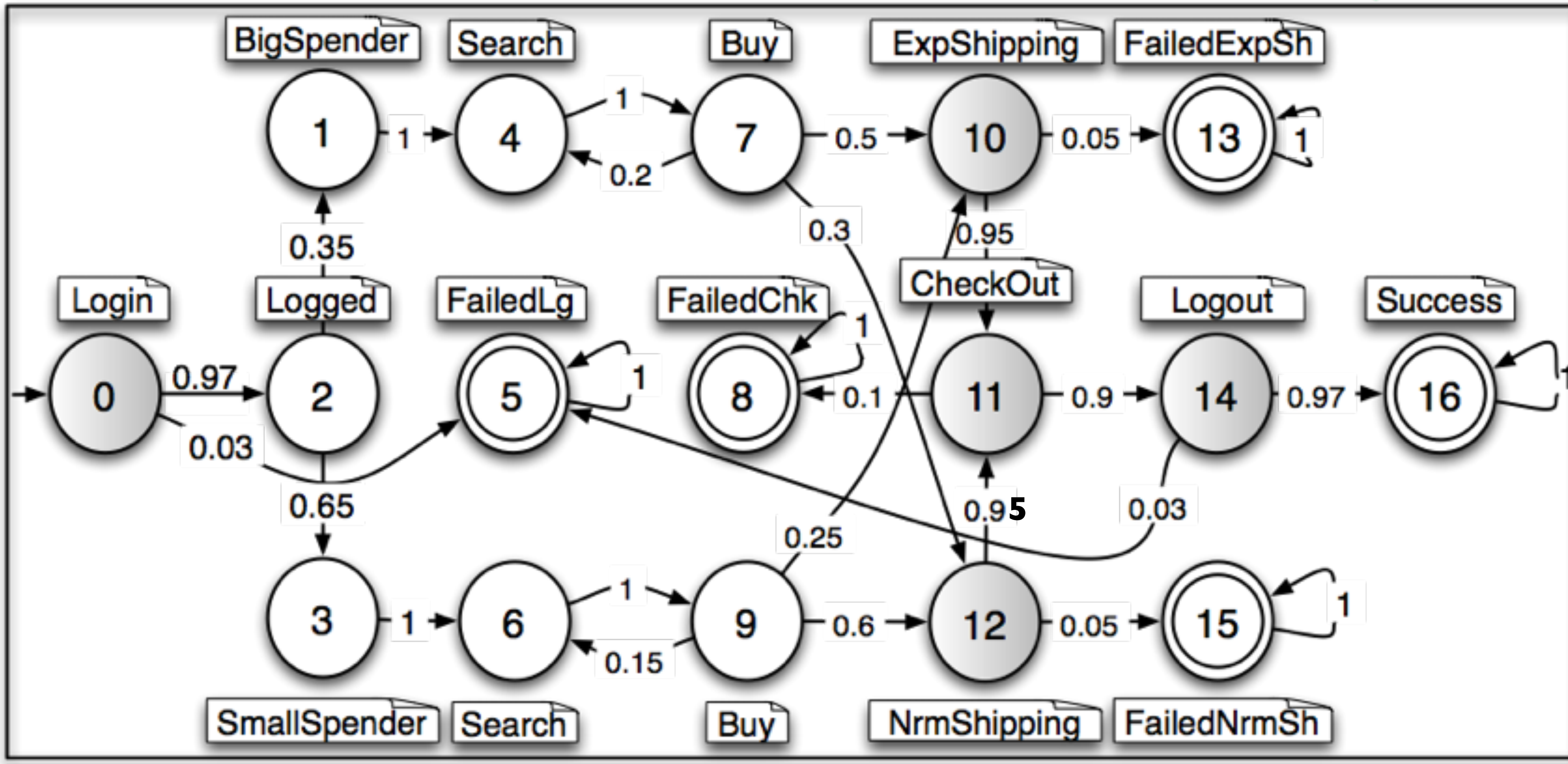
User profile domain knowledge

$D_{u,n}$	Description	Value
$D_{u,1}$	*P(User is a BS)*	0.35
$D_{u,2}$	*P(BS chooses express shipping)*	0.5
$D_{u,3}$	*P(SS chooses express shipping)*	0.25
$D_{u,4}$	*P(BS searches again after a buy operation)*	0.2
$D_{u,5}$	*P(SS searches again after a buy operation)*	0.15

External service assumptions (reliability)

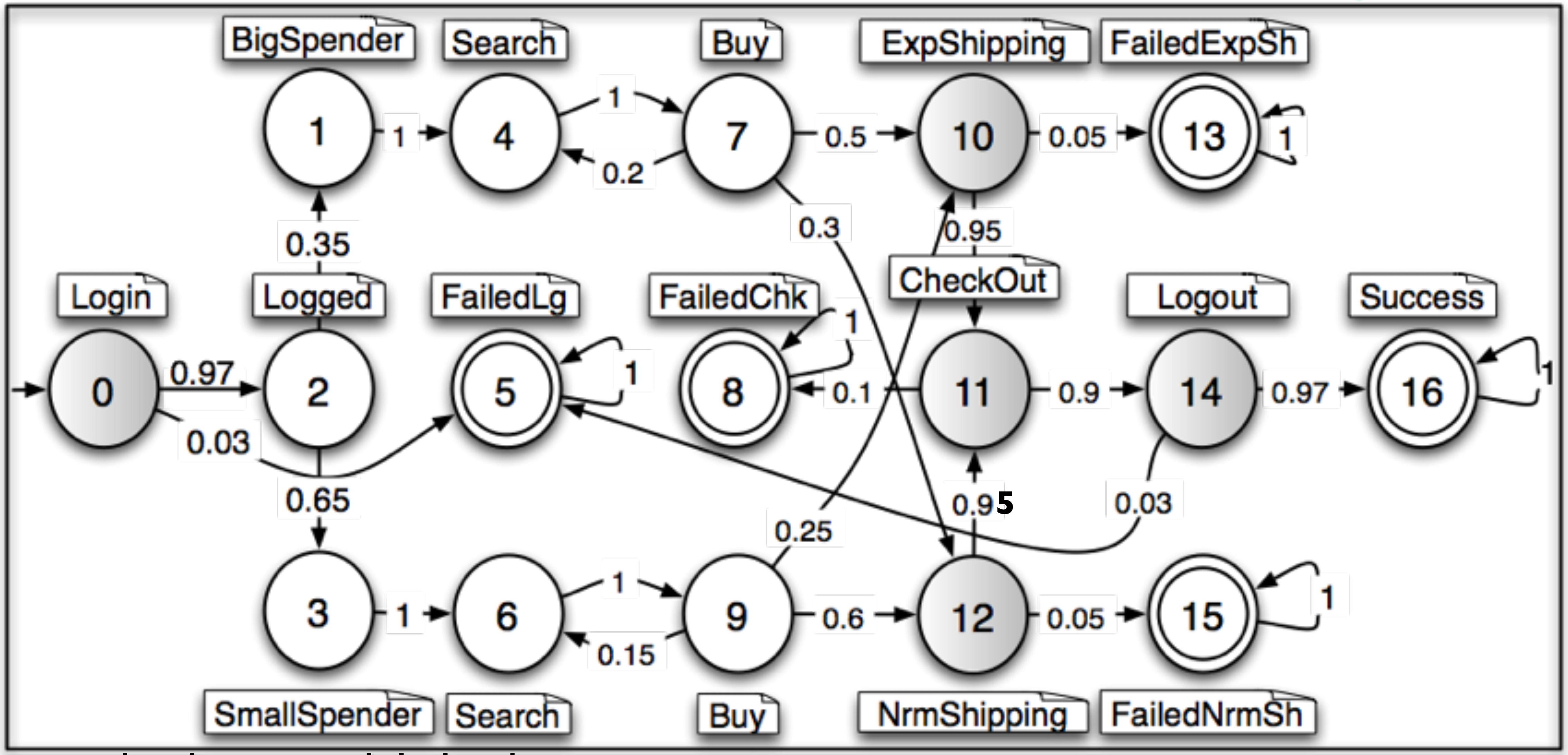
$D_{s,n}$	Description	Value
$D_{s,1}$	*P(Login)*	0.03
$D_{s,2}$	*P(Logout)*	0.03
$D_{s,3}$	*P(NrmShipping)*	0.05
$D_{s,4}$	*P(ExpShipping)*	0.05
$D_{s,5}$	*P(CheckOut)*	0.1

DTMC model


BigSpender
Search
Buy
ExpShipping
FailedExpSh
Login
Logged
FailedLg
FailedChk
CheckOut
Logout
Success
SmallSpender
Search
Buy
NrmShipping
FailedNrmSh
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
0.97
0.03
0.35
0.65
1
0.2
0.5
0.05
0.3
0.95
0.1
0.9
0.25
0.95
0.03
0.15
0.6

Self Managing Situated Computing

DTMC model



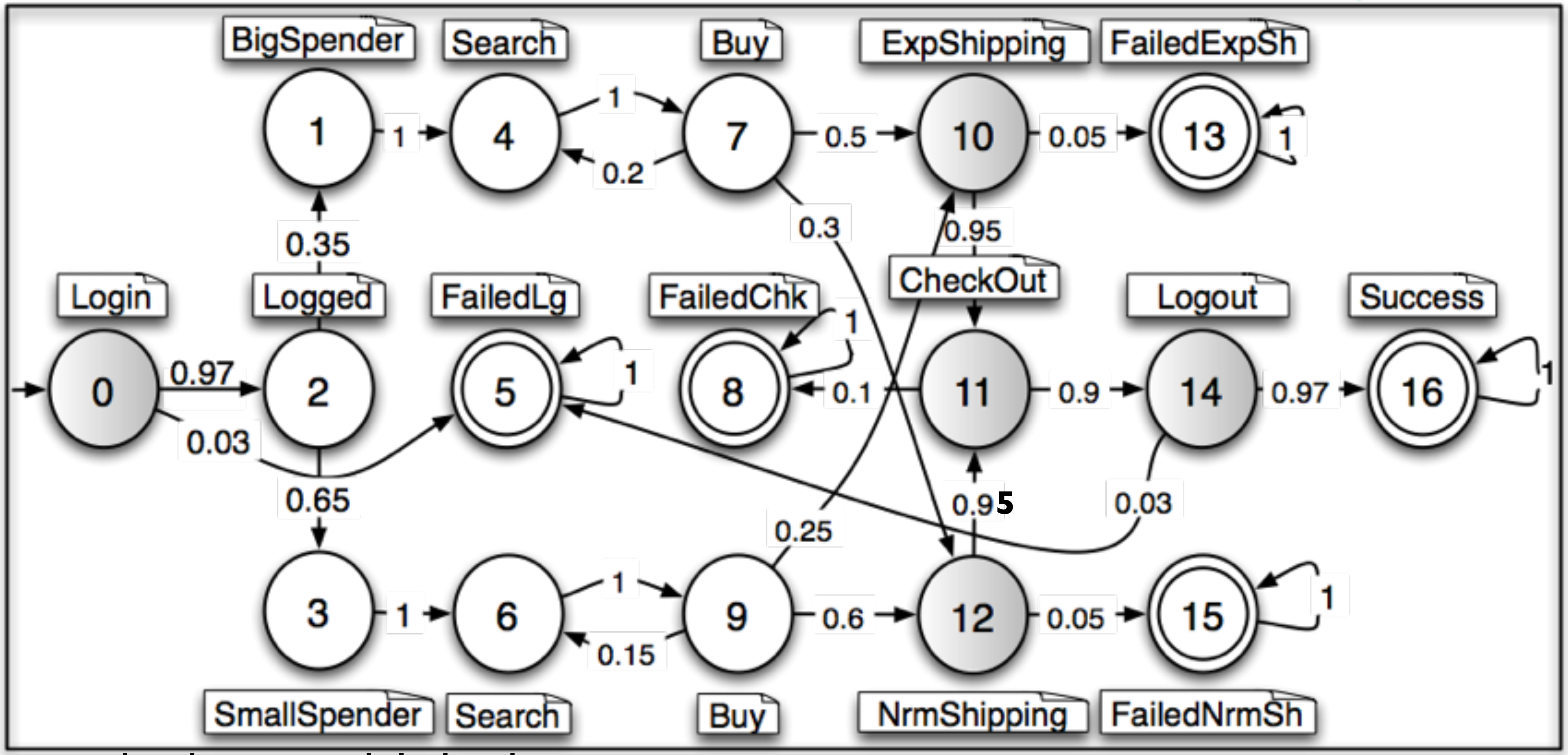
Property check via model checking

R1: “Probability of success is > 0.8”

R2:“Probability of a ExpShipping failure for a user recognized as BigSpender < 0.035”

R3:“Probability of an authentication failure is less then < 0.06”

DTMC model



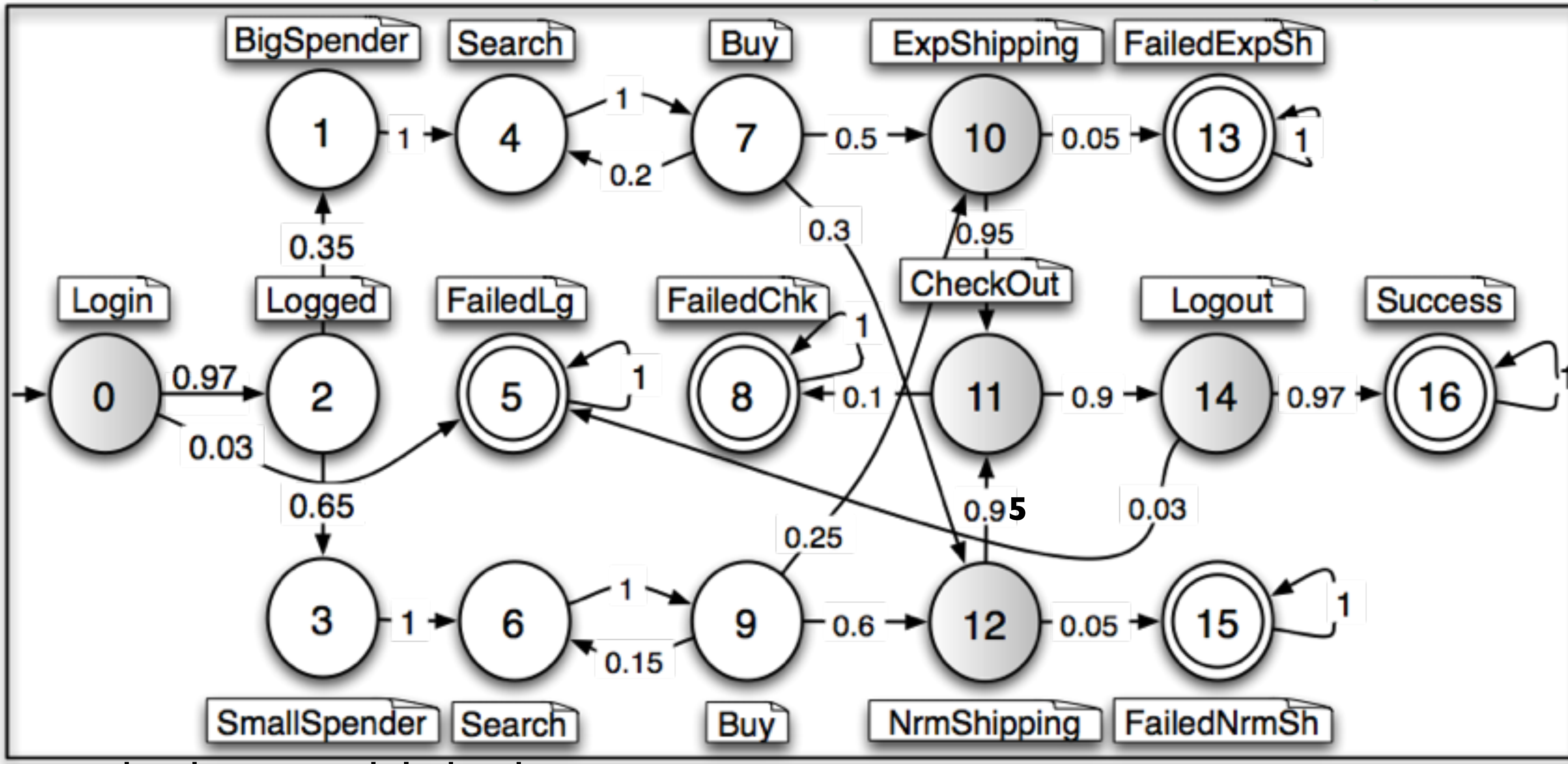
Property check via model checking

R1: “Probability of success is > 0.8” 0.84

R2:“Probability of a ExpShipping failure for a user recognized as BigSpender < 0.035”

R3:“Probability of an authentication failure is less then < 0.06”

DTMC model



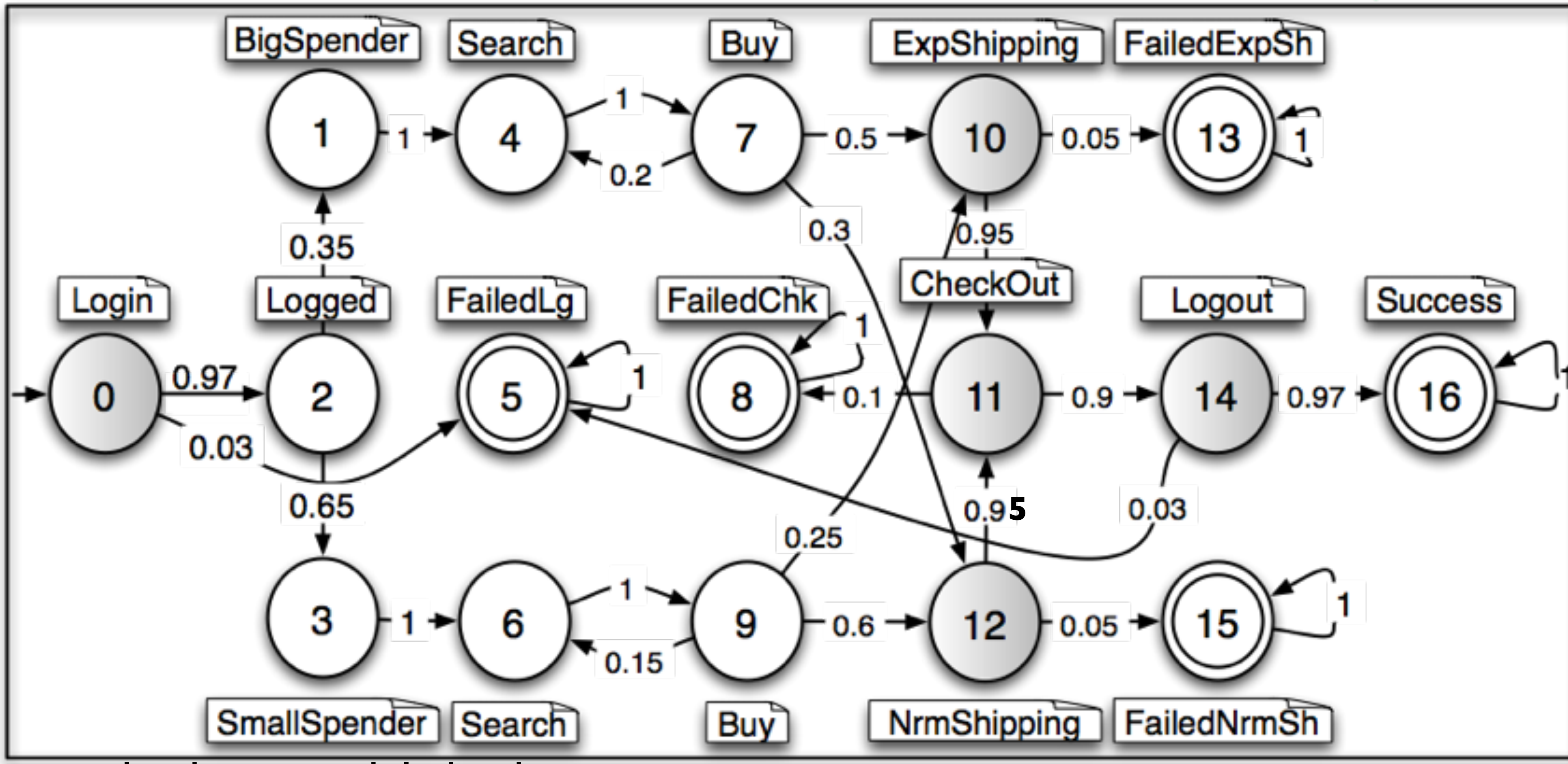
Property check via model checking

R1: "Probability of success is > 0.8"   0.84

R2: "Probability of a ExpShipping failure for a user recognized as BigSpender < 0.035"   0.031

R3: "Probability of an authentication failure is less then < 0.06"

DTMC model



Property check via model checking

R1: "Probability of success is > 0.8" 0.84

R2: "Probability of a ExpShipping failure for a user recognized as BigSpender < 0.035" 0.031

R3: "Probability of an authentication failure is less then < 0.06" 0.056

What happens at run time?

- We monitor the actual behavior
- A statistical (Bayesian) approach estimates the updated DTMC matrix (posterior) given run time traces and prior transitions
- Boils down to the following updating rule

What happens at run time?

- We monitor the actual behavior
- A statistical (Bayesian) approach estimates the updated DTMC matrix (posterior) given run time traces and prior transitions
- Boils down to the following updating rule

$$\boxed{m_{i,j}^{(N_i)} = \frac{c_i^{(0)}}{c_i^{(0)} + N_i} \times m_{i,j}^{(0)} + \frac{N_i}{c_i^{(0)} + N_i} \times \frac{\sum_{h=1}^{d} N_{i,j}^{(h)}}{N_i}}$$

What happens at run time?

- We monitor the actual behavior
- A statistical (Bayesian) approach estimates the updated DTMC matrix (posterior) given run time traces and prior transitions
- Boils down to the following updating rule

$$\boxed{m_{i,j}^{(N_i)} = \frac{c_i^{(0)}}{c_i^{(0)} + N_i} \times m_{i,j}^{(0)} + \frac{N_i}{c_i^{(0)} + N_i} \times \frac{\sum_{h=1}^{d} N_{i,j}^{(h)}}{N_i}}$$

A-priori Knowledge

Self Managing Situated Computing

What happens at run time?

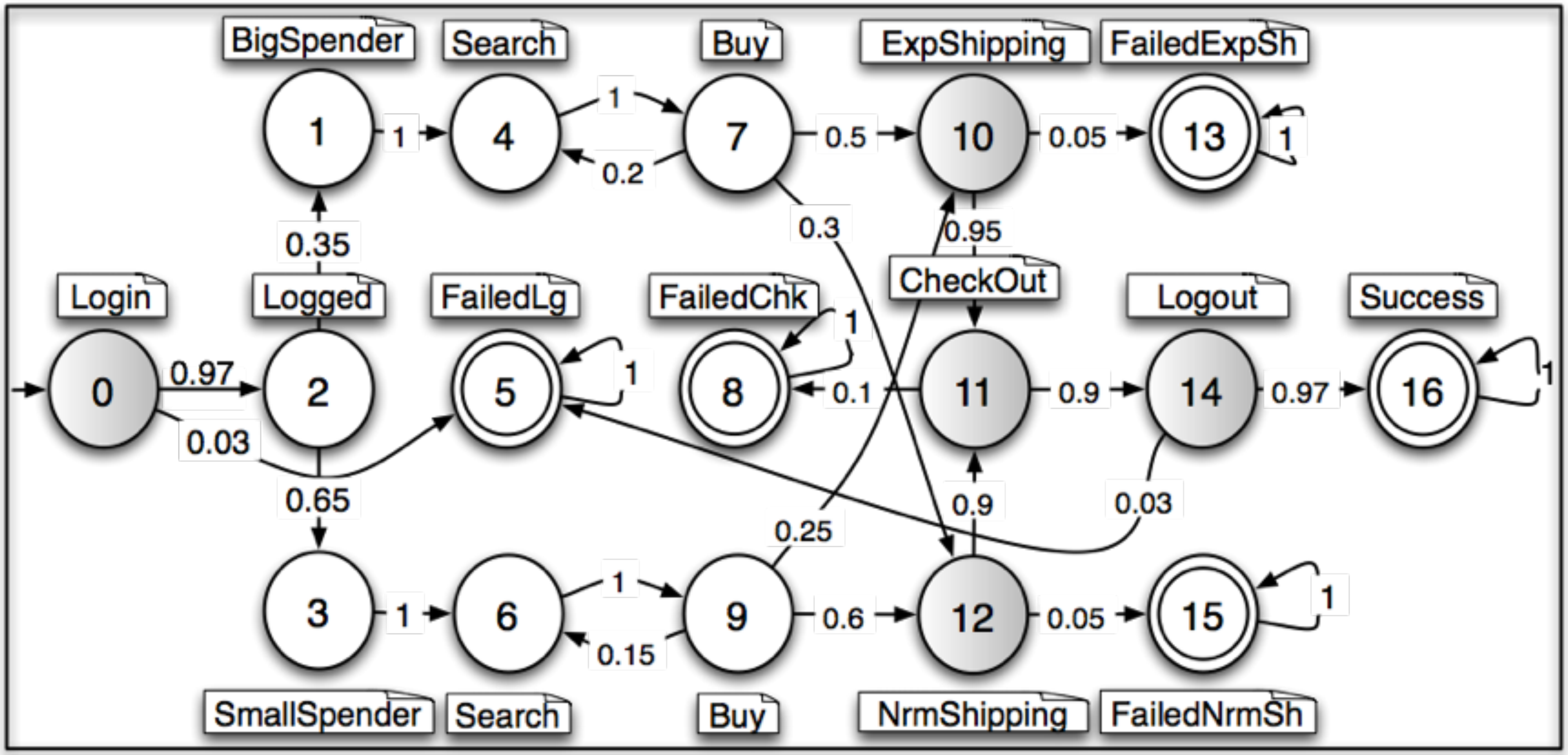
- We monitor the actual behavior
- A statistical (Bayesian) approach estimates the updated DTMC matrix (posterior) given run time traces and prior transitions
- Boils down to the following updating rule

$$m_{i,j}^{(N_i)} = \frac{c_i^{(0)}}{c_i^{(0)} + N_i} \times m_{i,j}^{(0)} + \frac{N_i}{c_i^{(0)} + N_i} \times \frac{\sum_{h=1}^{d} N_{i,j}^{(h)}}{N_i}$$

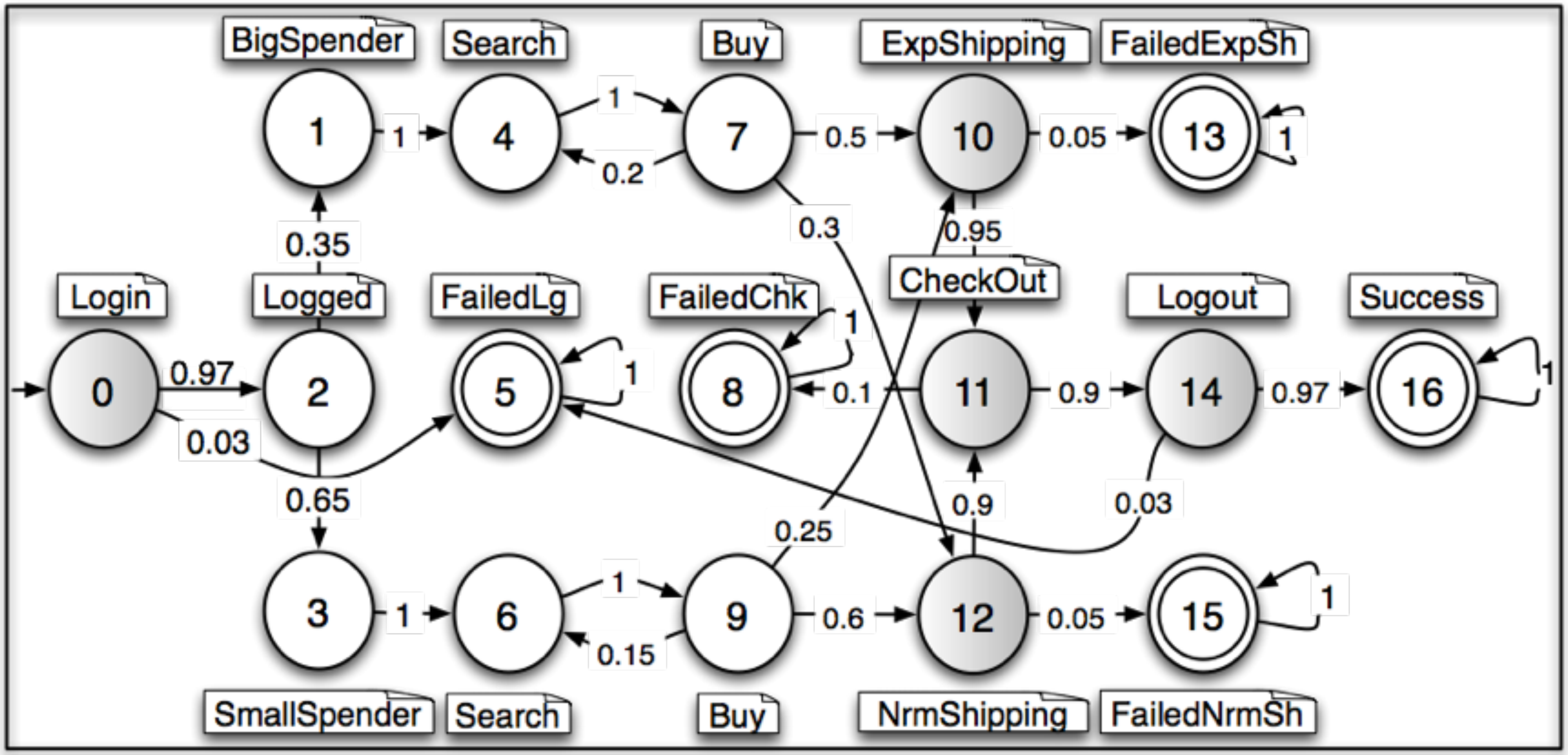
A-priori Knowledge

A-posteriori Knowledge

In our example

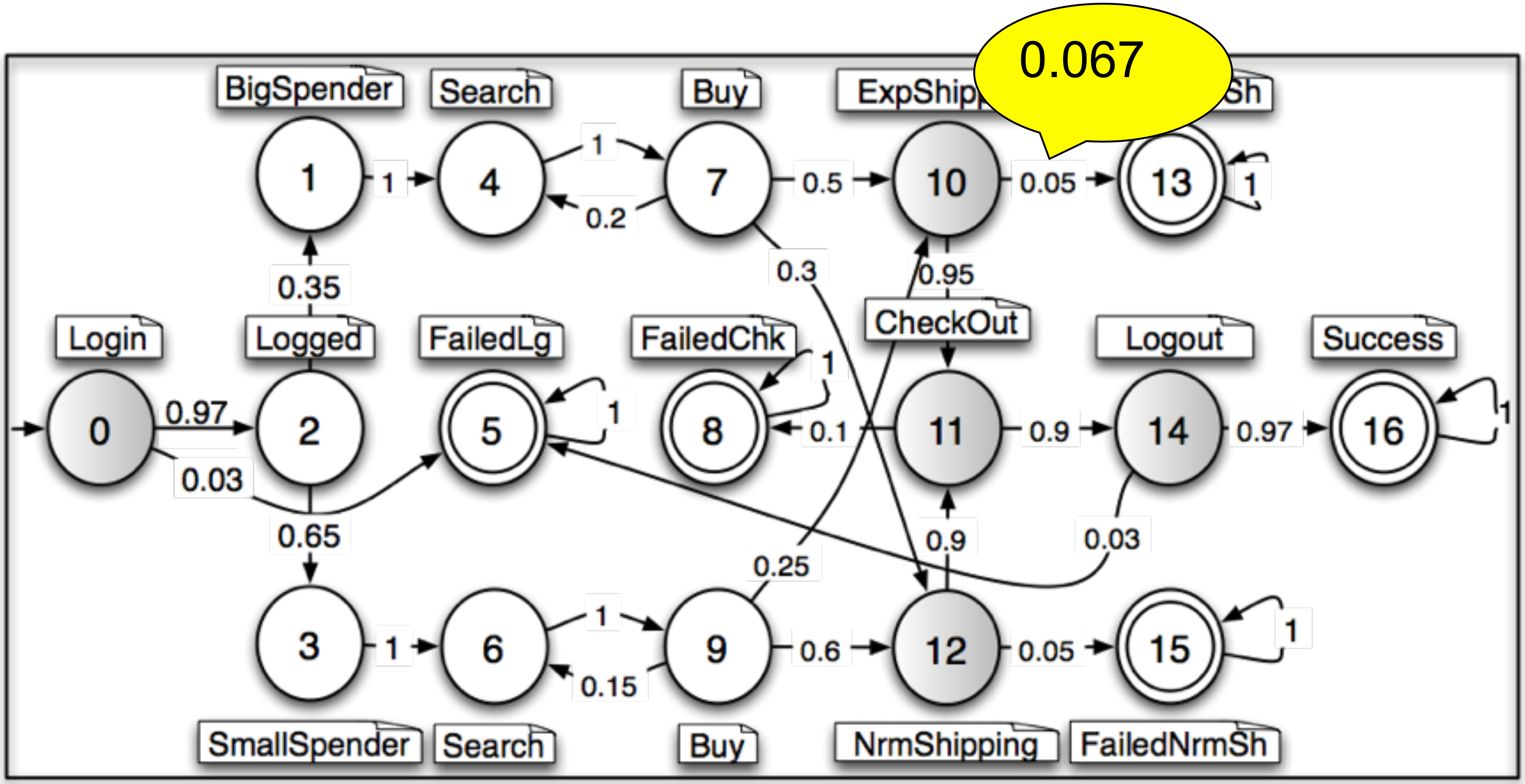


In our example



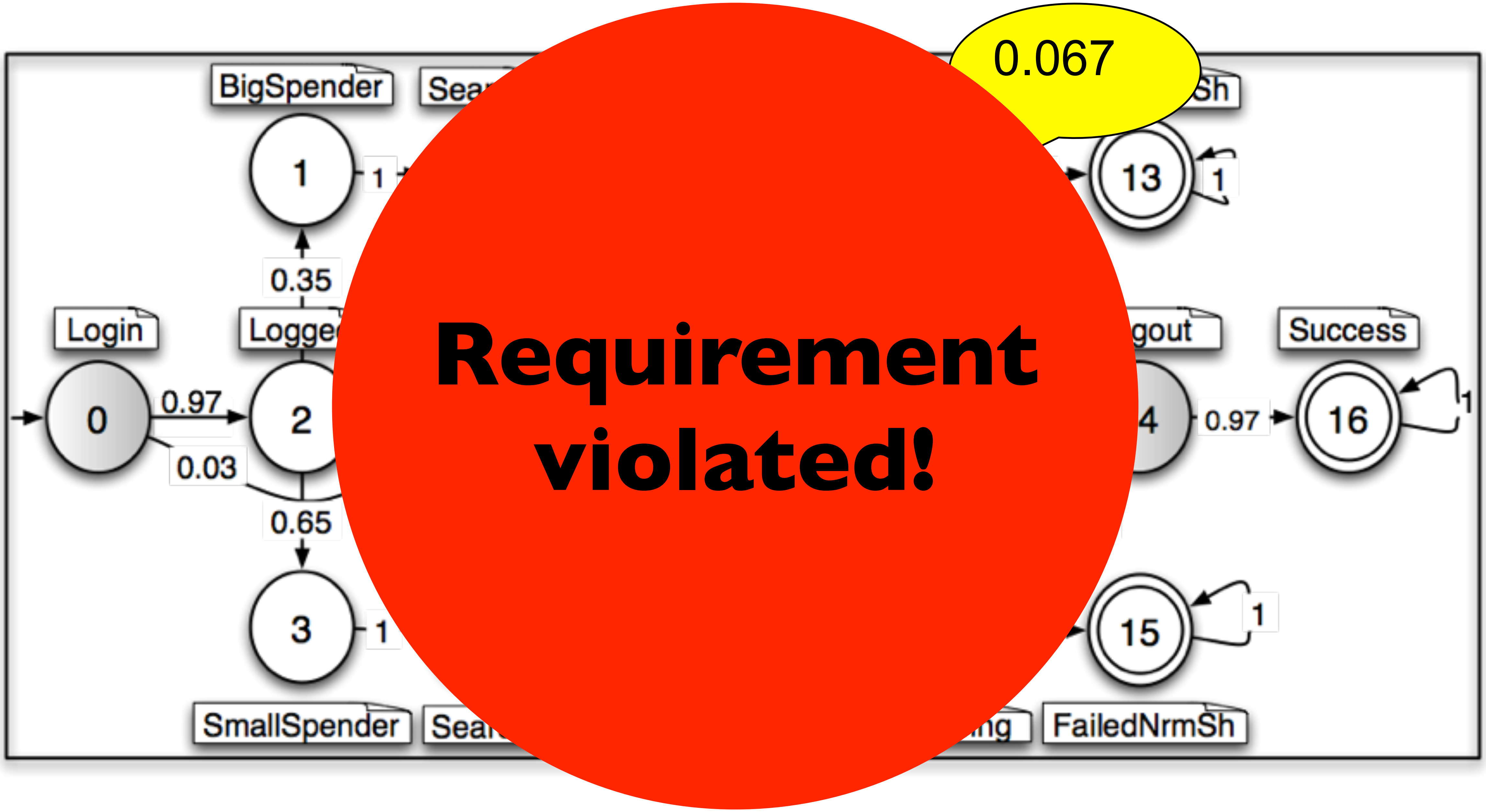
R2:“Probability of an ExpShipping failure for a user recognized as BigSpender < 0.035”

In our example



R2:"Probability of an ExpShipping failure for a user recognized as BigSpender < 0.035"

In our example



R2: "Probability of an ExpShipping failure for a user recognized as BigSpender < 0.035"

The problem

- Verification subject to (application-dependent) hard real-time requirements
- Running the model checker after any change impractical in most realistic cases
- But changes are often local, they do not disrupt the entire specification
- Can they be handled in an **incremental** fashion?
- This requires revisiting verification procedures!

Run-time agility, incrementality

- Agility taken to extremes
  - time boundaries shrink
    - ✓ constrained by real-time requirements
- Verification approaches must be re-visited
  - they must be **incremental**

Given S system (model), P property to verify for S
Change = new pair S', P'

Incremental verification reuses part of the proof of S against P to verify S' against P'

Incrementality by parameterization

- Requires anticipation of changing parameters
- The model is partly numeric and partly symbolic
- Evaluation of the verification condition requires *partial evaluation* (mixed numerical/symbolic processing)
- Result is a formula (polynomial for reachability on DTMCs)
- Evaluation at run time substitutes actual values to symbolic parameters

Incrementality by parameterization

- Requires anticipation of changing parameters
- The model is partly numeric and partly symbolic
- Evaluation of the verification condition requires *partial evaluation* (mixed numerical/symbolic processing)
- Result is a formula (polynomial for reachability on DTMCs)
- Evaluation at run time substitutes actual values to symbolic parameters

Working mom paradigm

Cook first

Warm-up later

Working mom paradigm

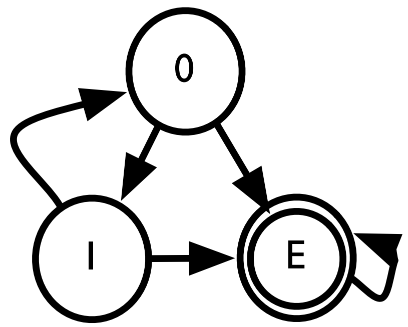
Design-Time
(offline)

Run-Time
(online)

Analyzable properties: reliability, costs (e.g., energy consumption)

Working mom paradigm

Design-Time
(offline)

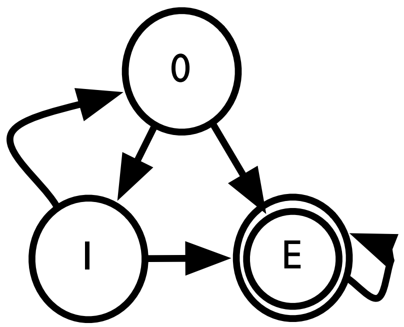


Run-Time
(online)

Analyzable properties: reliability, costs (e.g., energy consumption)

Working mom paradigm

Design-Time
(offline)



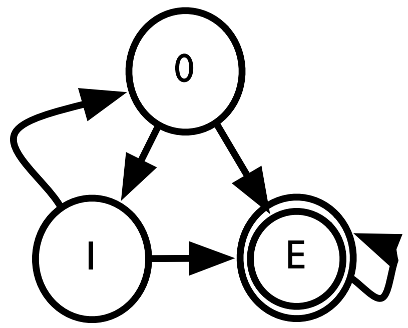
Partial
evaluation

Run-Time
(online)

Analyzable properties: reliability, costs (e.g., energy consumption)

Working mom paradigm

Design-Time (offline)



Partial evaluation

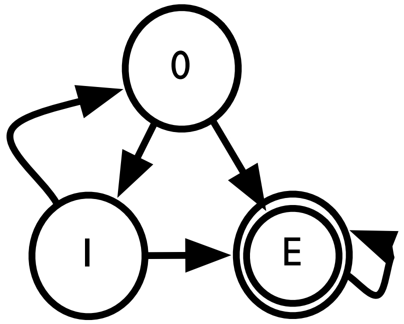
Run-Time (online)

Trace	
p11	0.34
p12	0.21
p31	0.12
p43	0.71
p31	0.23
p32	0.54

Analyzable properties: reliability, costs (e.g., energy consumption)

Working mom paradigm

Design-Time (offline)



Partial evaluation

Run-Time (online)

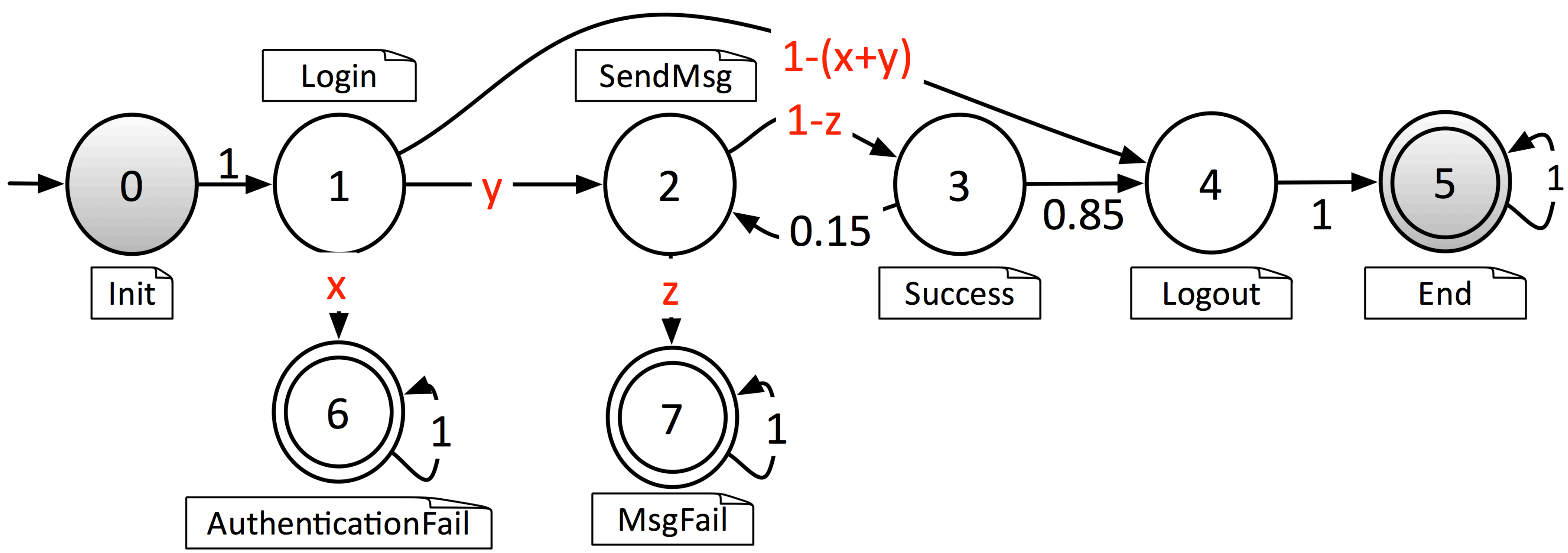
Trace	
p11	0.34
p12	0.21
p31	0.12
p43	0.71
p31	0.23
p32	0.54

Parameter values

Analyzable properties: reliability, costs (e.g., energy consumption)

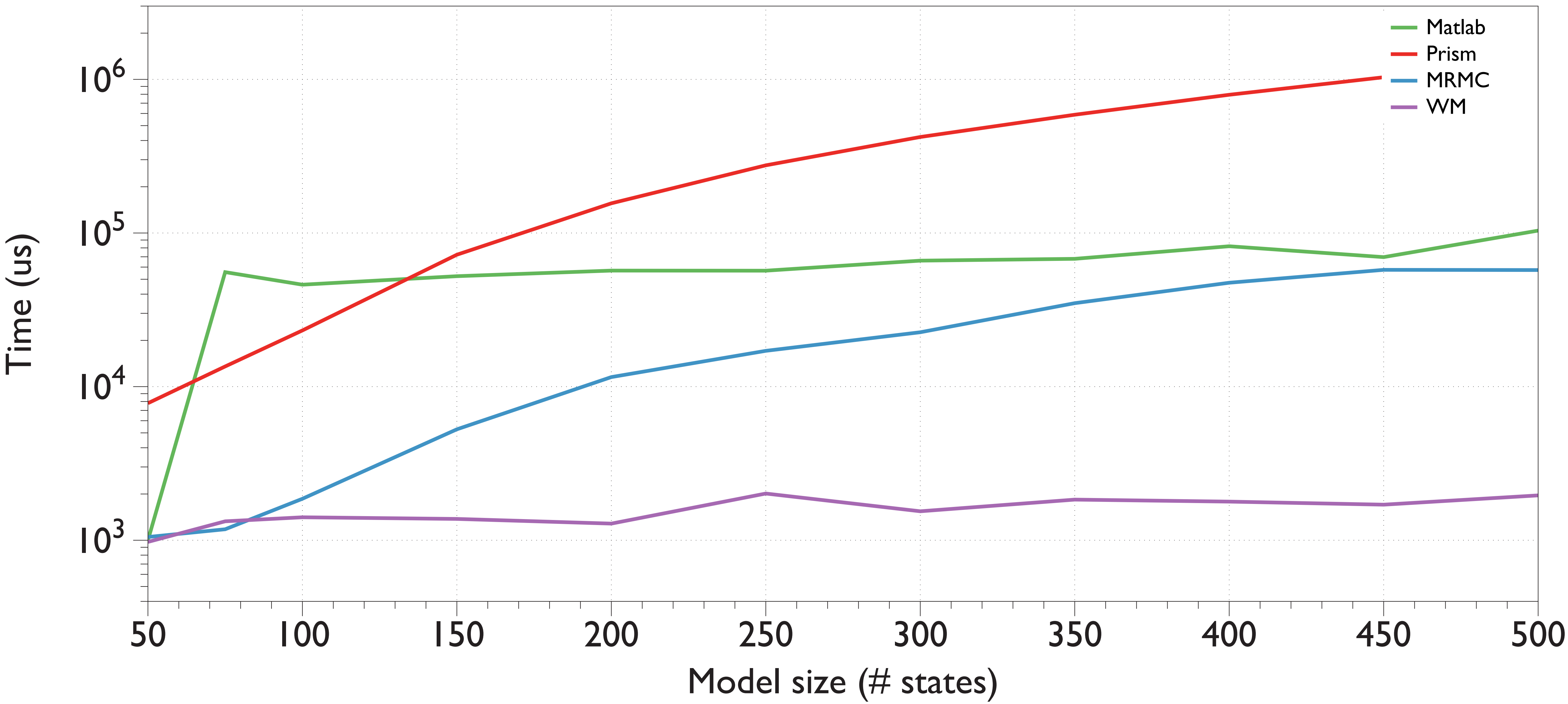
Back to the example

$$r = \Pr(\Diamond s = 5) > \bar{r}$$



$$r = \frac{0.85 - 0.85 \cdot x + 0.15 \cdot z - 0.15 \cdot x \cdot z - y \cdot x}{0.85 + 0.15 \cdot z}$$

Run-time verification



The WM approach

- Assumes that the Markov model contains absorbing states, and that they are reachable
- Works by symbolic/numeric matrix manipulation
- Resulting formula for reachability properties is polynomial
- All of (R) PCTL covered
- Expensive design-time partial evaluation, fast run-time verification
  - symbolic matrix multiplications, but very sparse and normally only few variables

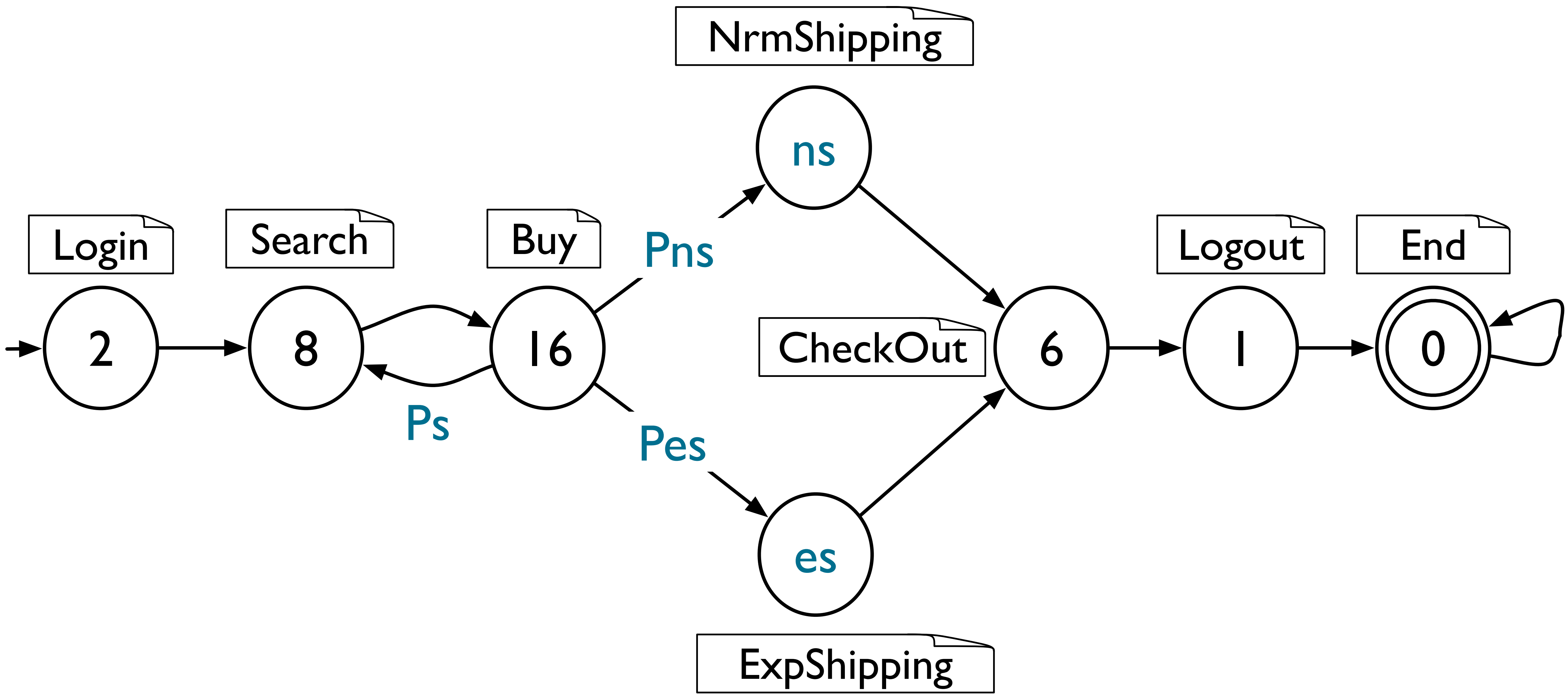
Further advantage of WM

- Because reachability properties can be expressed via polynomial functions, it is also possible to compute their (partial) derivative and perform sensitivity analysis
  - Which parameters affect most the global quality in the current operation point?
- Similar approach can deal also with rewards
  - Energy consumption, Average Execution time, Outsourcing cost, CPU time, Bandwidth

More on example

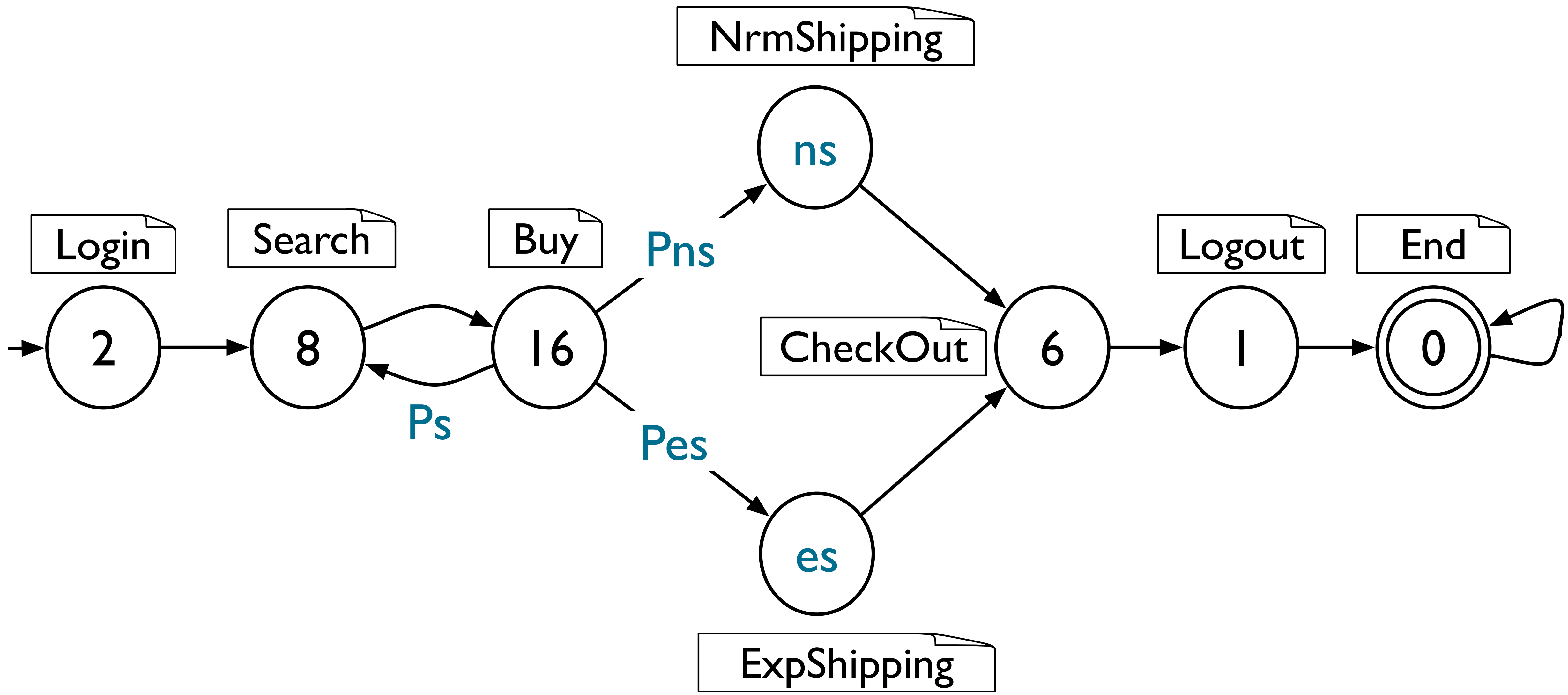
Cost units: $10^{-4}\$$

More on example



Cost units: $10^{-4}\$$

More on example



What's the average cost of a session?

Cost units: $10^{-4}\$$

WM for Reward-DTMC

WM for Reward-DTMC

Cook first:

WM for Reward-DTMC

Cook first: Partial evaluation

$$x_{\text{login}} = \frac{26 - 2 \cdot \text{Ps} + \text{Pns} \cdot \text{ns} + 7 \cdot \text{Pes} + 7 \cdot \text{Pns} + \text{Pes} \cdot \text{es}}{1 - \text{Ps}}$$

WM for Reward-DTMC

Cook first: Partial evaluation

$$x_{\mathrm{login}} = \frac{26 - 2 \cdot \mathrm{Ps} + \mathrm{Pns} \cdot \mathrm{ns} + 7 \cdot \mathrm{Pes} + 7 \cdot \mathrm{Pns} + \mathrm{Pes} \cdot \mathrm{es}}{1 - \mathrm{Ps}}$$

Warm-up later:

WM for Reward-DTMC

Cook first: Partial evaluation

$$x_{\mathrm{login}} = \frac{26 - 2 \cdot \mathrm{Ps} + \mathrm{Pns} \cdot \mathrm{ns} + 7 \cdot \mathrm{Pes} + 7 \cdot \mathrm{Pns} + \mathrm{Pes} \cdot \mathrm{es}}{1 - \mathrm{Ps}}$$

Warm-up later:

$$\{\mathrm{Ps} = 0.2, \mathrm{Pns} = 0.54, \mathrm{Pes} = 0.14, \mathrm{ns} = 20, \mathrm{es} = 50\}$$

WM for Reward-DTMC

Cook first: Partial evaluation

$$x_{\text{login}} = \frac{26 - 2 \cdot \text{Ps} + \text{Pns} \cdot \text{ns} + 7 \cdot \text{Pes} + 7 \cdot \text{Pns} + \text{Pes} \cdot \text{es}}{1 - \text{Ps}}$$

Warm-up later: Evaluate rational expression

$$\{\text{Ps} = 0.2, \text{Pns} = 0.54, \text{Pes} = 0.14, \text{ns} = 20, \text{es} = 50\}$$

$$\implies x_{\text{login}} = 60.875$$

More on incremental verification

- A must to move verification at run-time
- Also key to support development processes that are agile/iterative
  - they normally reject formal specification/verification because they are viewed as responsible for rigid, inflexible development
- Through incrementality, formal specification/verification can be reconciled with agility

A general traditional approach: assume-guarantee

- Resorts on modularity
- Applies assume-guarantee reasoning
  - System is viewed as a parallel composition of modules, each of which has to guarantee a certain property (its contract)
    - ✓ we show that module M1 guarantees properties P1 on the assumption that module M2 guaranties P2 , and vice versa for M2 , and then claim that the system composed of M1 and M2 (i.e., both running and interacting together) guarantees P1 and P2 unconditionally

More on incremental verification

A. Molzam Sharifloo, P. Spoletini,
LOVER: Light-Weight fOrmal Verification of adaptivE Systems at Run Time,
Formal Aspects of Component Software, LNCS, 2013

C. Ghezzi, C. Menghi, A. Molzam Sharifloo, P. Spoletini
On Requirements Verification for Model Refinements, accepted RE 2013

Incremental verification of state machines against functional properties

- Partial LTSs against CTL
  - a state (in general, a subgraph) can be un-specified
    - ★ we can compute the property P^-- the constraint that needs to be satisfied by the un-specified fragment so that the entire LTS satisfies a property P

- The un-specified portion is the one that may change; constraints are the pre-computed property that needs to be verified on the portion

How to make verification incremental

Syntax-driven incrementality

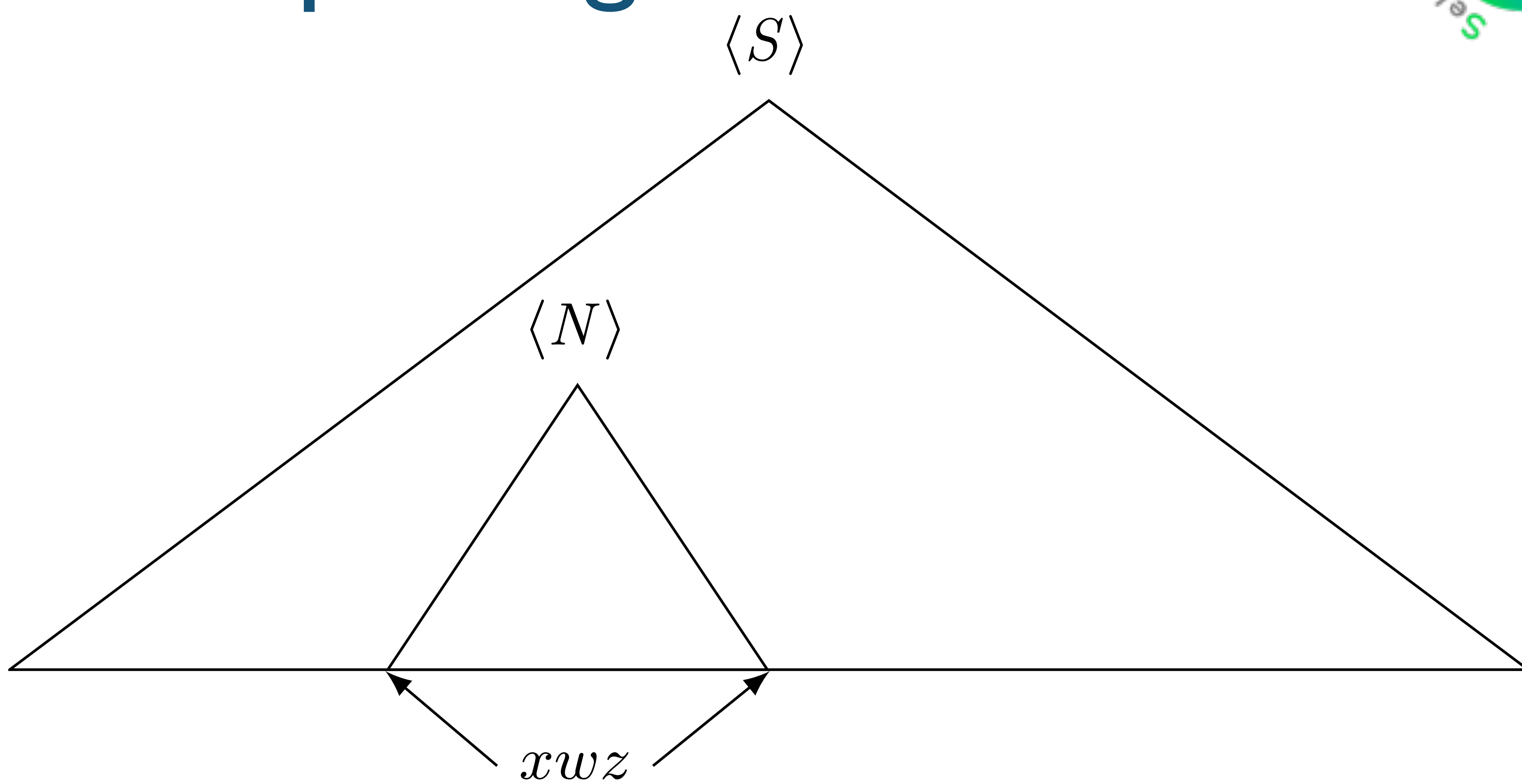
- Assumes that artifact to analyze has a syntactic structure expressible as a formal grammar
- Verification is expressed via attributes (à la Knuth)
- Changes can be of any kind

Intuition

Syntax-driven incrementality

- Incremental parsing strategy finds boundary for artifact re-analysis
- Knuth proved that attributes can be only synthesized (computed bottom-up) and thus only need to be recomputed for the changed portion + propagated to the root node

Incremental parsing: intuition



- Assume *w* is the modified portion
- Ideally, re-analyze only a sub-tree “covering” w, rooted in <N>, and “plug-it-in” the unmodified portion of tree
- The technique works if the sub-tree is small, and complexity of re-analysis is the same as complexity of “main” algorithm

Incremental parsing: past and new results

- Past work on incremental parsing
  - C. Ghezzi, D. Mandrioli, Incremental parsing, ACM Trans. Program. Lang. Systems, 1979
  - C. Ghezzi, D. Mandrioli, "Augmenting parsers to support incrementality", Journal of the ACM, 1980
    - ✓ Saves the maximum possible portion of the syntax tree, but the re-analyzed portion can still be large in certain cases
- Recent work resurrected Floyd's operator precedence grammars
  - R.W. Floyd. Syntactic analysis and operator precedence, Journal of the ACM, 1963
  - S. Crespi Reghizzi and D. Mandrioli. Operator-precedence and the visibly pushdown property, J. Comput. Syst. Sci., to appear.
  - Floyd's grammars cannot generate all deterministic CF languages
  - but in practice any programming language can be described by a Floyd grammar
  - parsing can be started from any arbitrary point of the artifact to be analyzed; it can work in parallel

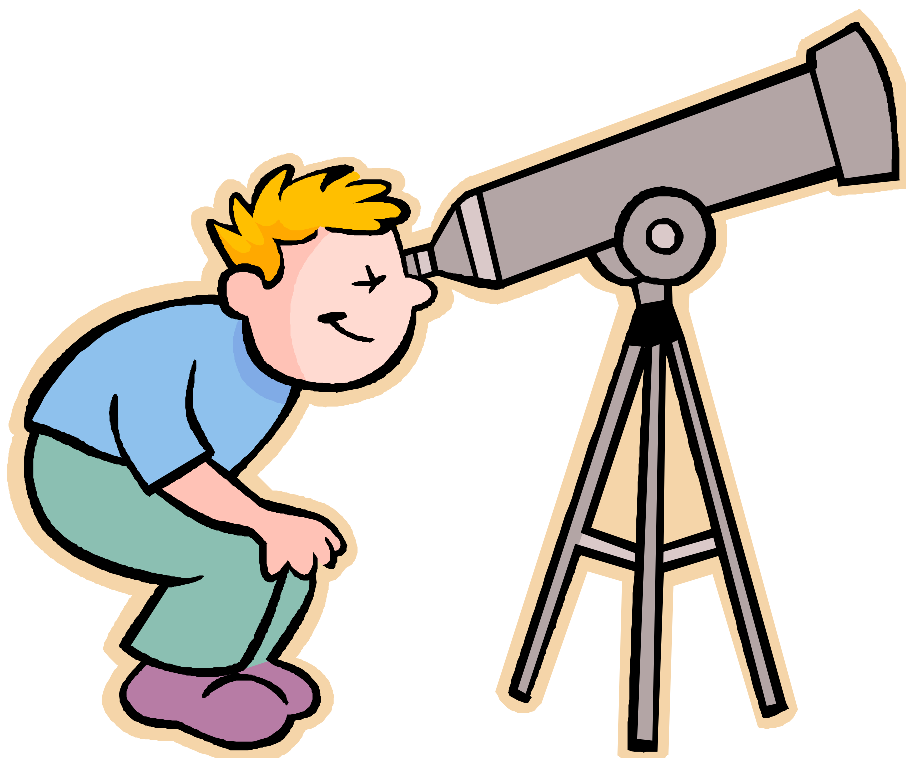
Initial validation of the approach

- Case 1: reliability (QoS) analysis of composite workflows
  - a (BPEL) workflow integrates external Web services having given reliability and we wish to assess reliability of composition
  - if reliability of an external service changes, does our property about reliability of composition change?
    - ✓ our previous work framed this into probabilistic model checking
    - ✓ here we can deal with unrestricted changes, also in the workflow in a very efficient way

Initial validation of the approach

- Case 2: reachability analysis as supported by program model checking
  - given a program and a safety property, is there an execution of the program that leads to a violation of the property?
  - if the program changes, how does our property change?
    - ✓ similar problem faced by Henzinger et al.
    - T.A. Henzinger, R. Jhala, R. Majumdar, and M.A. Sanvido. Extreme model checking. In Verification: Theory and Practice, volume 2772 of LNCS, 2004.

Other research directions



- Adaptation via control theory

A. Filieri, C. Ghezzi, A. Leva, M. Maggio, Self-Adaptive Software Meets Control Theory: A preliminary Approach Supporting Reliability Requirements, ASE 2011

A. Filieri, C. Ghezzi, A. Leva, M. Maggio
Reliability-driven dynamic binding via feedback control, SEAMS 2012

- Dynamic software update

X, Ma, L, Baresi, C, Ghezzi, V. Panzica La Manna, and J. Lu, Version-consistent Dynamic Reconfiguration of Component-based Distributed Systems, ESEC/FSE 2011

C. Ghezzi, J. Greenyer, V. Panzica La Manna, Synthesizing Dynamically Updating Controllers from Changes in Scenario-based Specifications, SEAMS 2012

V. Panzica La Manna, , J. Greenyer, C. Ghezzi, C. Brenner, Formalizing Correctness Criteria of Dynamic Updates Derived from Specification Changes, SEAMS 2013

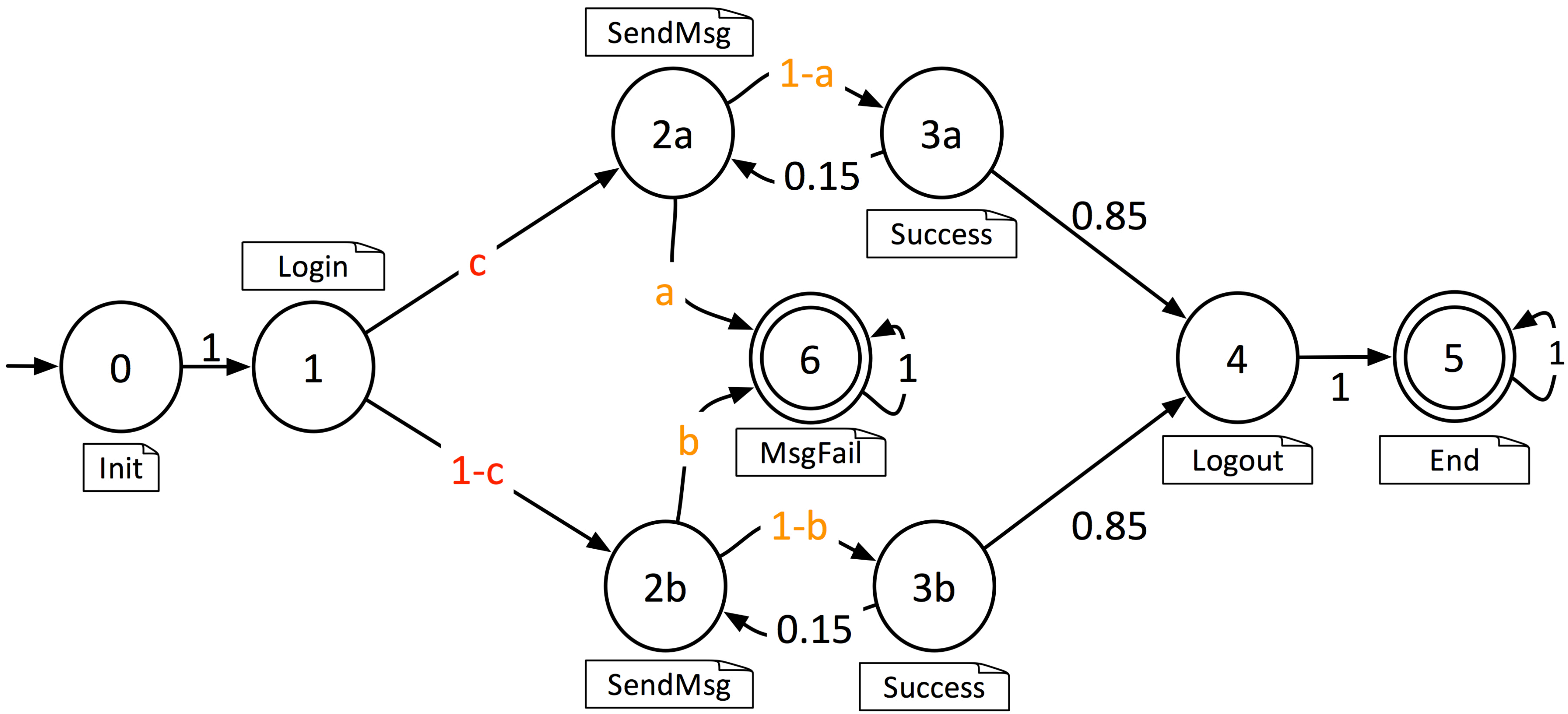
- Model inference techniques

C. Ghezzi, A. Mocci, M. Monga
Synthesizing Intensional Behavior Models by Graph Transformation (ICSE 2009)

C. Ghezzi, A. Mocci, G. Salvaneschi
Automatic Cross Validation of Multiple Specifications: A Case Study (FASE 2010)

C. Ghezzi, A. Mocci: Behavioral validation of JFSL specifications through model synthesis (ICSE 2012)

Tunable DTMC model



3 kinds of transitions: fixed, observable, controllable

Questions?