SOFTWARE AND SYSTEMS ENGINEERING research group

Model-based testing under uncertainty

L1: Theoretical aspects & practical applications

Matteo Camilli
matteo.camilli@unibz.it
https://matteocamilli.github.io

Formal Methods at Work @
Gran Sasso Science Institute (GSSI)
A.Y. 2019/20

Outline

- Markov Decision Process
  - Structure
  - Rewards
  - Policy, best policy, value iteration
- Model-based testing (MBT)
  - Offline vs online approaches
  - Conformance relation
  - Probabilistic alternating simulation and refinement
- Online MBT under uncertainty
  - Problem statement
  - Uncertain model paramenters
  - Bayesian inference
  - Framework and test case generation strategies

Markov decision processes

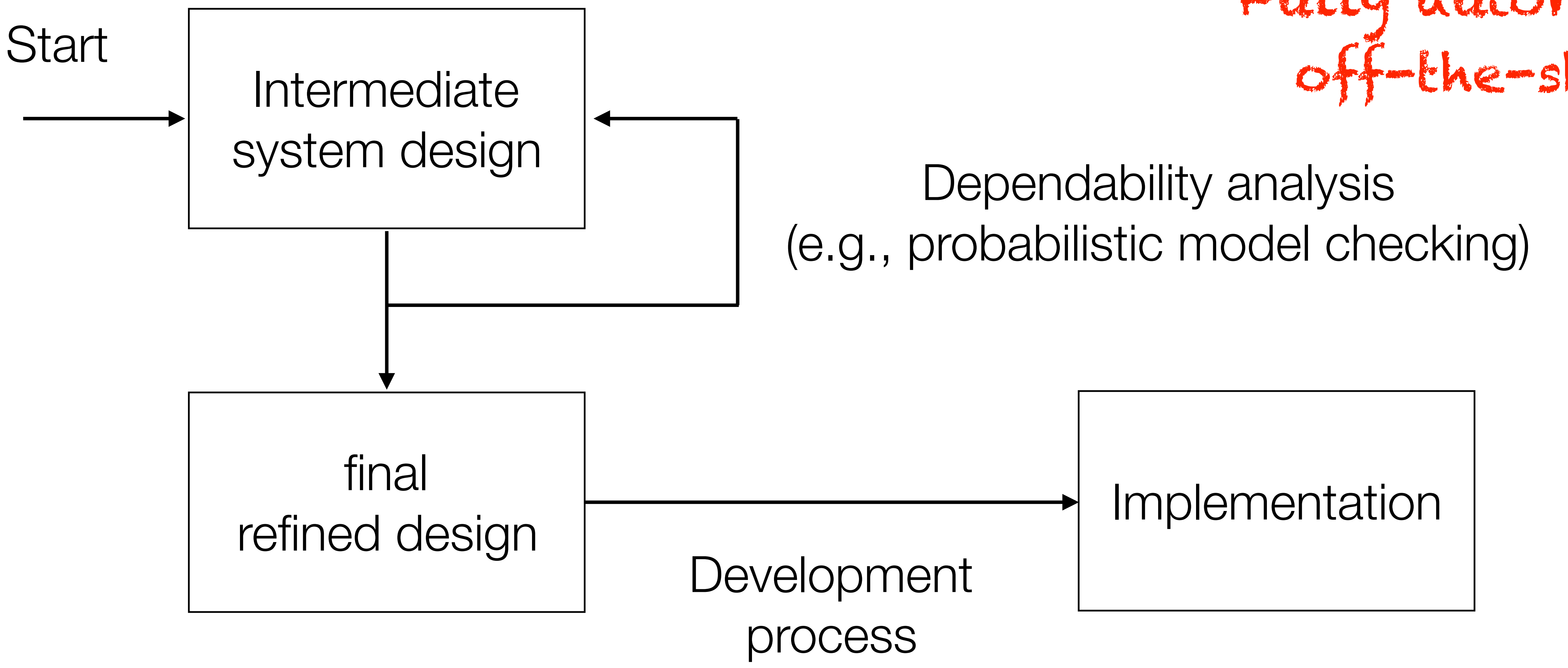
- Structure
- Examples
- Rewards
- Policy, best policy
- Value iteration algorithms

Markov Models

- Basic notions
  - The **behavior** of the target system (or phenomenon) of interest is **partially/fully stochastic**
  - Formal framework for **performance** and **dependability** (reliability, availability, safety) analysis
  - **Dependability modeling** (upfront) at design time improves the quality of the system eventually produced
  - **Assumption** — the modeled system meets the **Markov property**[1,2] (memoryless)

Fully automated using off-the-shelf tools

Engineering perspective

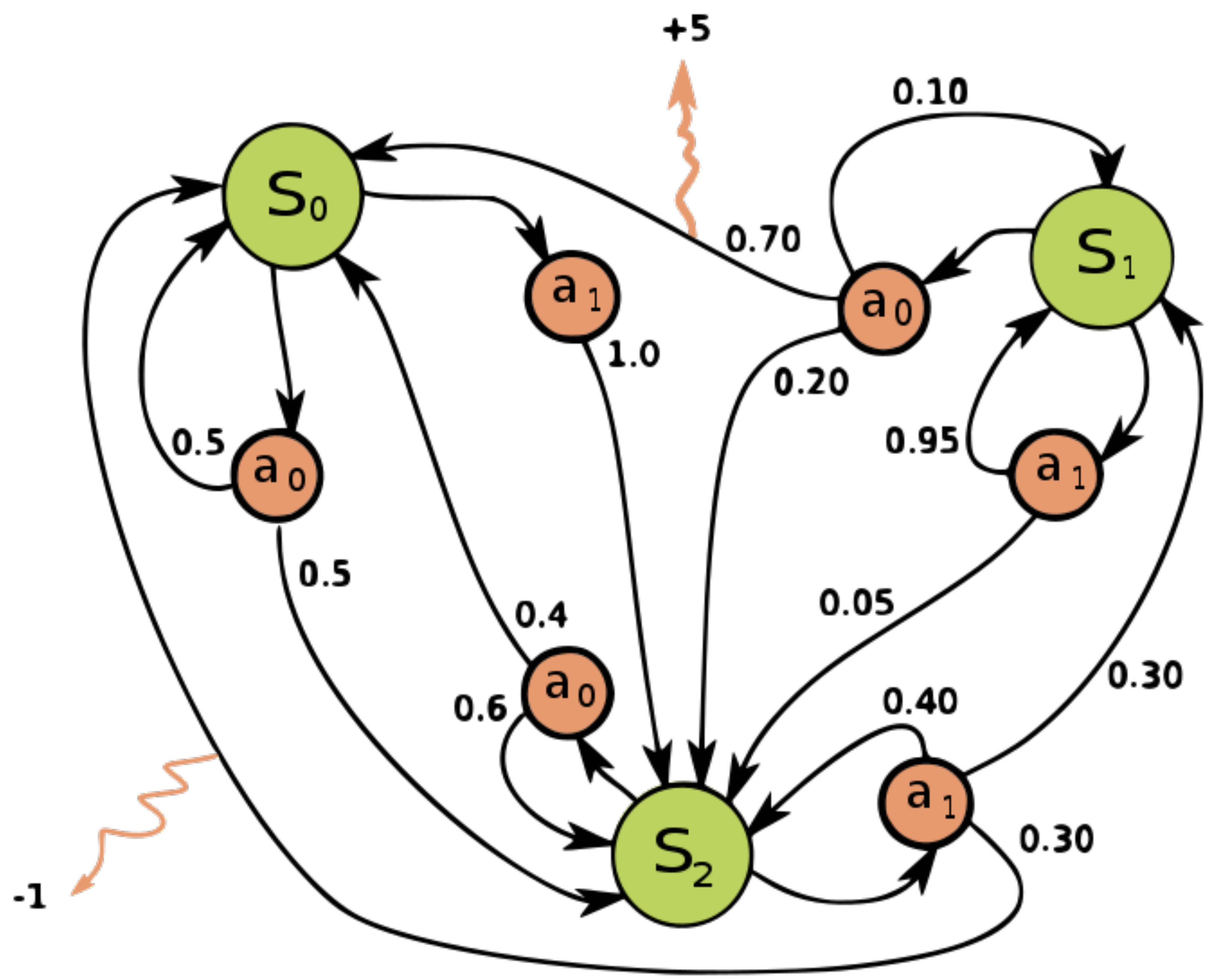


1. The the probability of moving to the next state only depends on the current state, not on the history that lead to that state.
2. R. C. Cheung, A user-oriented software reliability model, IEEE TSE, no. 2, pp. 118–125, 1980

Markov Decision Process

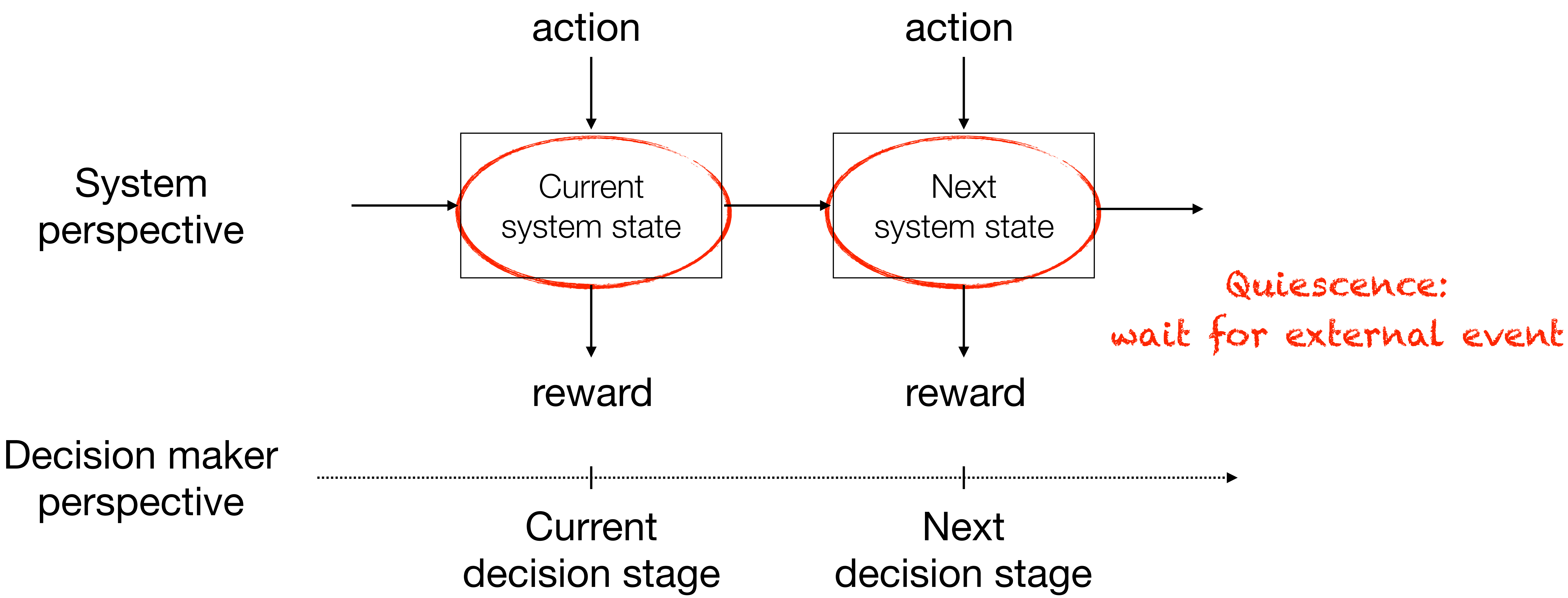
- Mathematical framework for modeling systems whose behavior is partially
  - Nondeterministic — actions (external stimuli) under the control of a decision maker
  - Stochastic — random outcome out of an executed action

- Formal structure
  - S: set of states (finite/infinite)
  - s_0: initial state
  - A: set of actions (alphabet)
  - P: S x A x S —> [0,1], $P(s, a, s') = P(s_{t+1} = s' \mid s_t = s, a_t = a)$
  - R: S x A x S —> $\mathbb{R}$, $R(s, a, s')$ reward for $(s_{t+1} = s', s_t = s, a_t = a)$



MDP behavior

- How does the model operate
  - The system must be in **one** of the **states** (finite countable set) at a time
  - The system makes a transition **s —> s'** when **one** of the available **actions** is selected



MDP behavior (2)

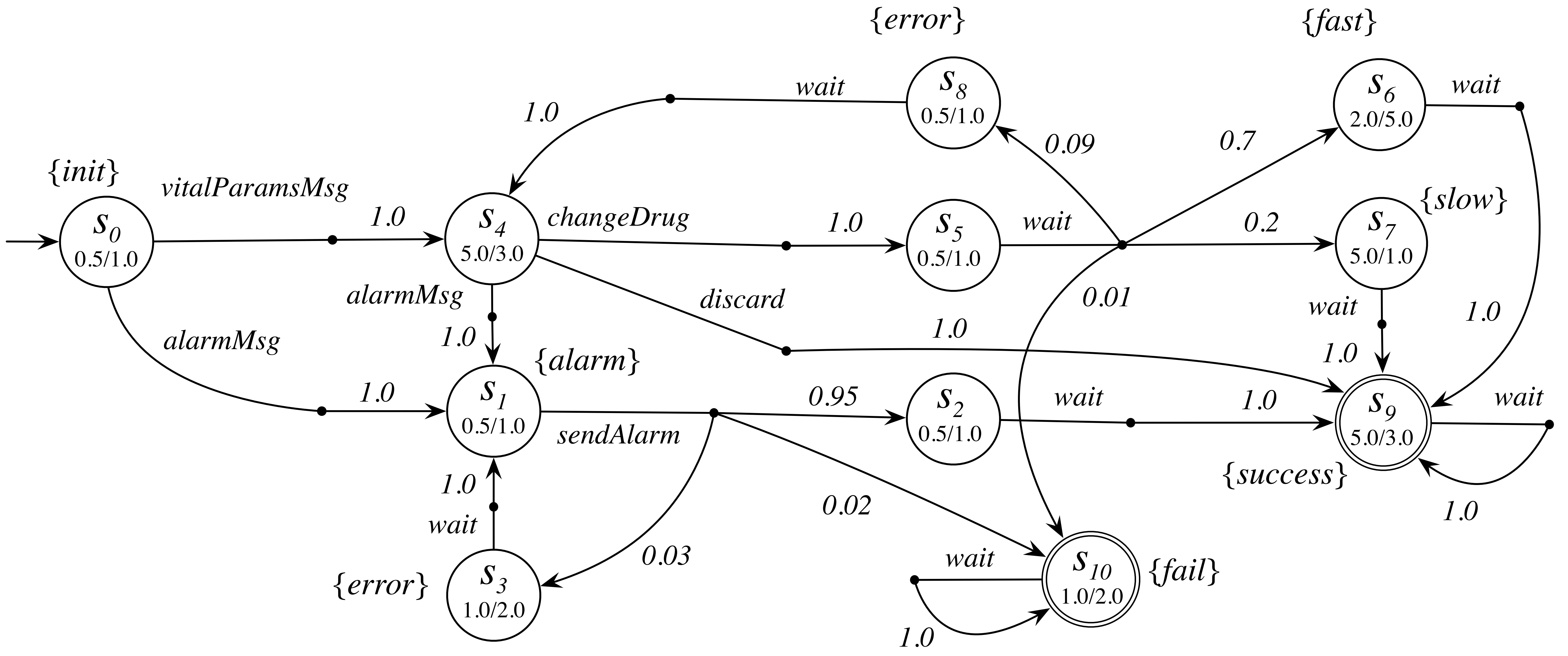
- States
  - System configurations or operational status of components
  - Instances of the system where
    - Components are operational or failed (e.g., enumeration of working/failed components)
    - Experienced specific sequences of events (e.g., events observed so far)
    - Operating in a fully-functioning mode, degraded mode, faulty, etc.
    - Undergoing recover/repair
- Actions
  - Possible inputs or external events
- Transitions
  - Define whether is possible to go from one state to another
  - Transition probability —> governs the likelihood of observing the transition

MDP examples

- Possible scenarios
  - Service-based systems
  - Web applications
  - Mobile applications
  - Cyber physical systems (CPSs)
  - Control policies in robotics
  - Security protocols
  - etc.
- Selected examples
  - Tele assistant system (TAS) — example of service-based system
  - SafeHome — example of CPS

MDP examples — TAS

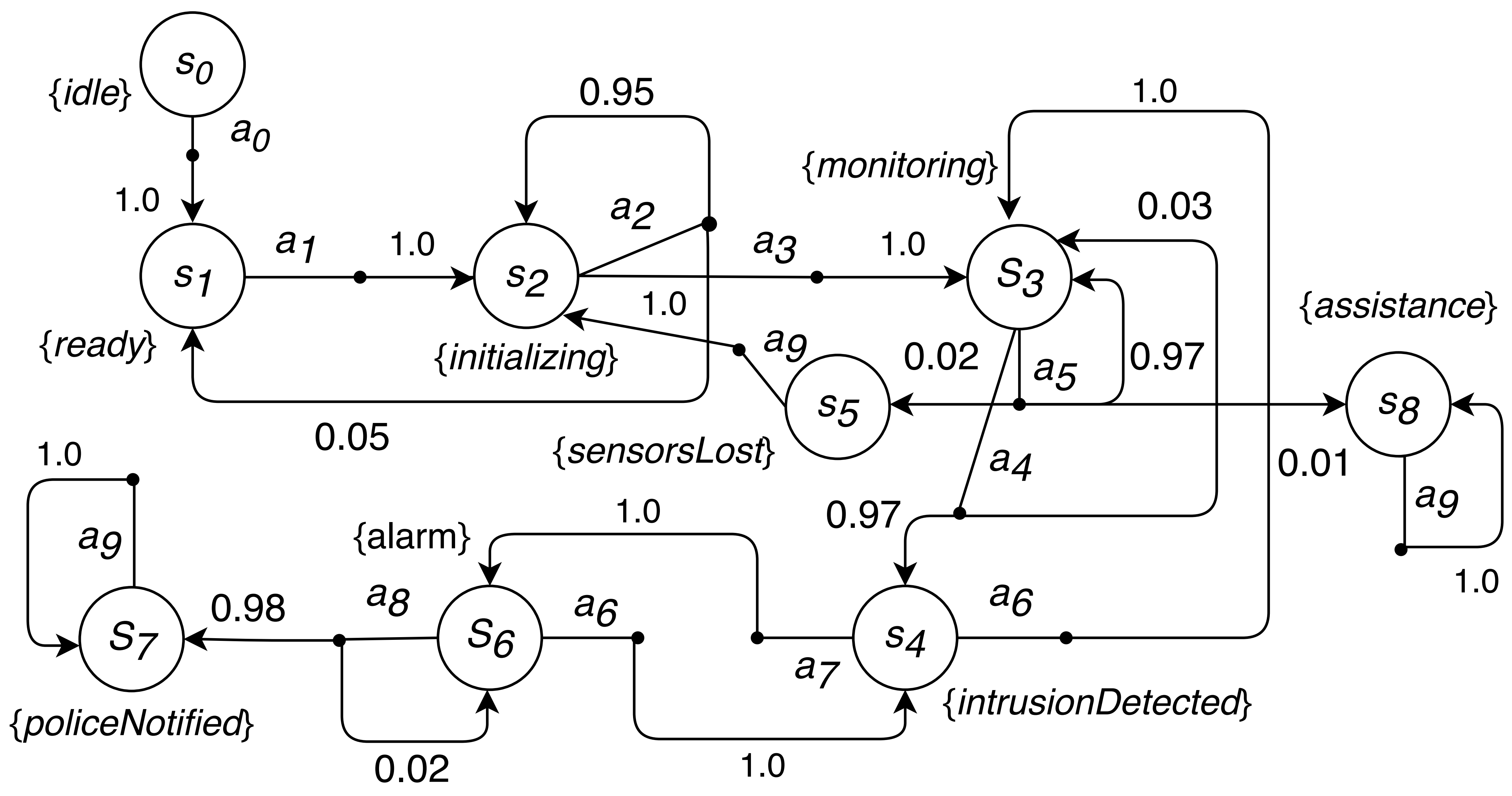
- TAS[1,2]
  - SBS providing health support to chronic condition patients at their homes
  - **wearable devices** (track vital parameters) + **remote services** (healthcare, pharmacy and emergency units)



1. D. Weyns et al., Tele Assistance: A Self-Adaptive Service-Based System Examplar, SEAMS 2015, IEEE
2. M. Camilli et al., Online Model-Based Testing under Uncertainty, ISSRE 2018, IEEE

MDP examples — SafeHome

- SafeHome[1,2]
  - Configure and control alarms along with related sensors that implement security and safety features
  - Here is the description of a part in charge of monitoring and intrusion detection
    - Modeled **phases**: sensors **initialization**, **monitoring**, **detection**, **notification**



Actions
a0: activate
a1: startInit
a2: initSensors
a3: startMonitoring
a4: intrusionOccurred
a5: sensorsCheck
a6: cancel
a7: turnAlarmOn
a8: notify
a9: wait

1. Roger S Pressman. 2005. Software engineering: a practitioner's approach. Palgrave Macmillan
2. Man Zhang, at al., Uncertainty-wise test case generation and minimization for Cyber-Physical Systems. 2019, JSS 153

MDP with rewards

- An MDP model can be augmented with multiple **reward structures**
- Reward structure
  - $R: S \times A \times S \longrightarrow \mathbb{R}$, $R(s, a, s')$ reward for $(s_{t+1} = s', s_t = s, a_t = a)$
  - Describe **nonfunctional** aspects (e.g., energy consumption, computational cost, response time, …)

Mental note

This is the usual interpretation of a reward structure.
We'll see how to leverage this notion in a "unconventional" way to drive testing.

MDP policy

- The notion of policy π refers to the way a Decision Maker (DM) solves nondeterminism of a MDP
- Deterministic policy[1,2]
  - π: S —> A, prescribes the action to take given a state
  - DM objective: choose π which maximizes the expected cumulated reward over an infinite horizon
    - This is called best policy π*
- Definition of the best deterministic policy
  - Given R(s, a), i.e., the one-step expected reward
  - We can compute the value function V(s) for each state

Bellman's equation

Best policy

$$R(s,a) = \sum_{s' \in S} p_{s,a,s'} \, r_{s,a,s'} \qquad V(s) = \max_{a \in A} \{R(s,a) + \gamma \sum_{s' \in S} p_{s,a,s'} V(s')\} \qquad \pi^*(s) = \arg\max_{a \in A} \{R(s,a) + \gamma \sum_{s' \in S} p_{s,a,s'} V(s')\}$$

1. Given a deterministic policy, the MDP reduces to a Discrete Time Markov Chain (DTMC).
2. Martin L. Puterman. 1994. Markov Decision Processes: Discrete Stochastic Dynamic Programming (1st. ed.). John Wiley & Sons, Inc., USA

MDP policy — Value iteration

```
Procedure ValueIteration(S,A,P,R,θ)
    Inputs
        S set of all states
        A set of all actions
        P transition function P(s,a,s')
        R reward function R(s,a,s')
        θ a threshold, θ>0
    Output
        π[S] optimal policy
        V[S] value function
    Local
        real array Vk[S] is a sequence of value functions
        action array π[S]
    assign V0[S] arbitrarily
    k ←0
    repeat
        k ←k+1
        for each state s do
            Vk[s] = maxa ∑s' P(s,a,s') (R(s,a,s')+ γVk-1[s'])
            π[S] = a
    until ∀s |Vk[s]−Vk-1[s]| < θ
    return π,Vk
```

- The ValueIteration procedure uses dynamic programming
  - Memoization + recursion (or iteration)
- This procedure converges no matter what is the initial value function V_0

(current problem)

(subproblem)

Update V_k based on V_{k-1}

Model-based testing of probabilistic systems

- Offline vs online approaches
- Conformance relation
- Probabilistic alternating simulation and refinement

Model-based testing

- Basic idea
  - A formal model of the required behavior of the System Under Test (SUT) is used as baseline of
    - test case generation
    - Construction of the oracle
  - Test suites are automatically extracted from models and then executed
- Formal verification vs Model-based testing
  - Formal verification — prove that the model (i.e., formal specification) satisfies requirements
  - MBT — show that the SUT behaves as defined in the (verified) model
    - Limitation: testing is not complete (i.e., *"testing can only show the presence of errors, not their absence"*)

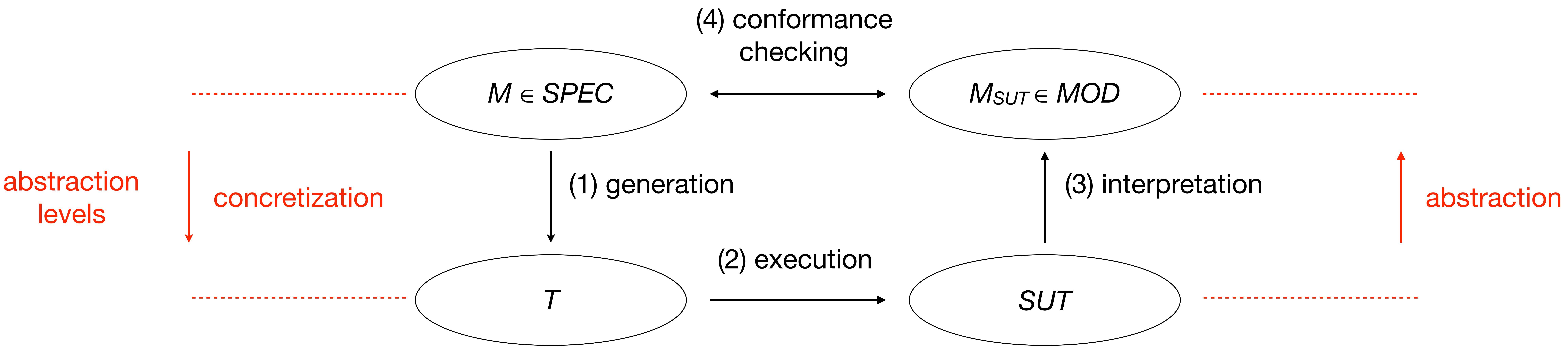
MBT — terminology

- Implementation or System Under Test
  - Piece of hardware/software, a software system, an embedded system, a CPS, etc.
  - The SUT is viewed as a black-box (secret internal structure)
  - The tester controls and observes the SUT via its interfaces (e.g., APIs)
- Specification
  - Describes what the SUT should do using a formal notation (or language)
  - *SPEC* — set of all valid models in a formal notation
    - A specification is $M \in SPEC$
- Conformance
  - Formalizes the notion of correct behavior of a SUT w.r.t. $M \in SPEC$
  - Problem: we'd like to define a relation between elements of different domains
    - *M* (formal entity) <—> SUT (not a formal entity)

MBT — terminology (2)

- **Conformance Problem:** *M* (formal domain) <—> SUT (not formal domain)
  - Trick — test assumption
    - The **SUT behavior** can be interpreted using the **same level of abstraction** of *M*
    - The SUT behavior is a model $M_{SUT} \in MOD \subseteq SPEC$
      - *MOD* — universe of implementation models
      - M_{SUT} not a-priori known
- Conformance (under the test assumption)
  - Can be expressed as a formal relation between *MOD* and *SPEC* elements
  - $conf \subseteq MOD \times SPEC$
  - M_{SUT} is correct w.r.t. *M* if M_{SUT} *conf M*
- Conformance checking
  - Assess by testing whether M_{SUT} *conf M*
  - *Create T (test suite) s.t.* M_{SUT} conf *M* => M_{SUT} *passes T* (sound but not complete)

MBT process

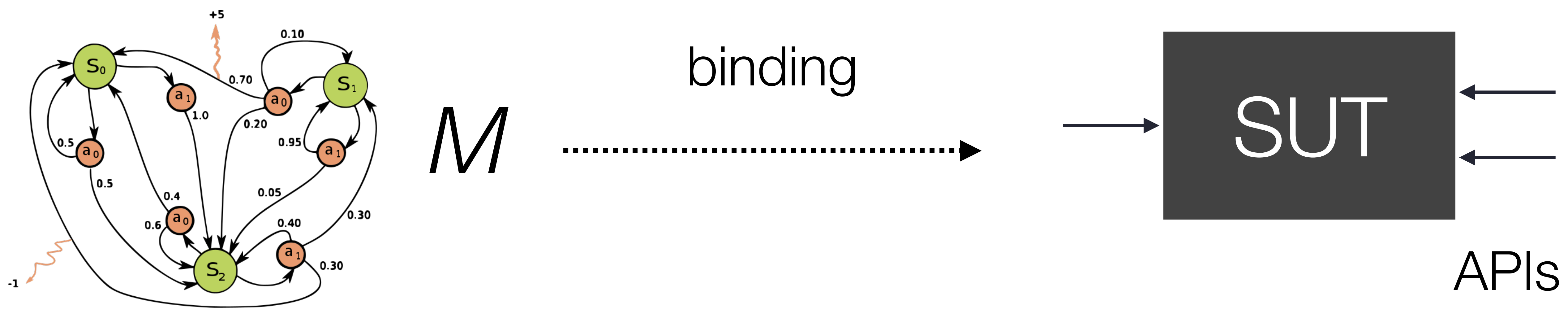


- Offline vs online [1]
  - Offline — steps 1-4 are **separated**
  - Online (or on-the-fly) — steps 1-4 are **merged** into a one-iteration step
    - Test cases are created dynamically and take advantage of the knowledge gained by exploring *M*
    - Iterative approach —> 2-players game: **controller + observer**

1. Utting, Mark, and Bruno Legeard. Practical model-based testing: a tools approach. Elsevier, 2010

Online MBT with MDPs

- Binding (concretization)
  - Defines a **mapping** between the **MDP model** spec and **the SUT behavior**



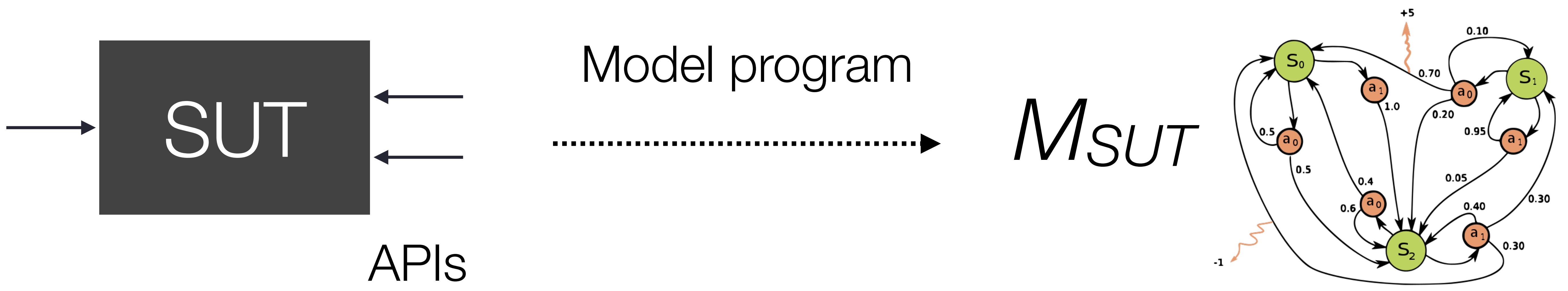
- Formal definition
  - Given a **MDP** M = (S, s_0, A, P) and a **SUT**, i.e., a set of **exported services H** (having signature and arguments), a **binding** is a **tuple of partial functions (h, i, post)** s.t.
    - h(s, a), a ∈ A(s) identifies a service ∈ H
    - i(s, a), a ∈ A(s) identifies a vector v_{in} for the service h(s, a)
    - post(s, a, s'), a ∈ A(s) maps to a **post-condition** that must hold for v_{out} resulting from the execution of the service h(s, a) on input v_{in}

Controllable SUT components

Observable SUT behavior

Online MBT with MDPs (2)

- Model program (abstraction)
  - Defines the abstract **interpretation** of the **SUT behavior** in terms of **MDP model**



- Formal definition
  - Given a MDP $M = (S, s_0, A, P)$ and a binding (h, i, post) the model program $M_{SUT} = (S', s_0', A', P')$ is a MDP model s.t.
    - $S' \subseteq S$ All observable SUT states exist in M
    - $A \subseteq A'$ All controllable actions in M are feasible in SUT
    - $s_0 = s_0'$
    - $P'(s, a, s') > 0$ iff there exists $v_{out} = h(s, a)(v_{in})$ s.t. post(s, a, s') holds for v_{out}

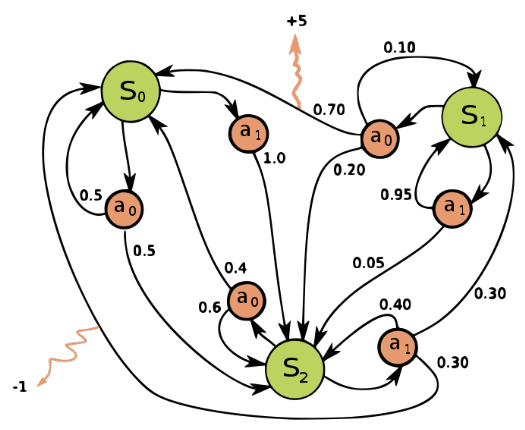
M_{SUT} transitions are defined in terms of SUT behavior

Online MBT with MDPs (3)

- Conformance checking
  - Needs the definition of conformance relation = probabilistic alternating simulation + refinement

- Probabilistic alternating simulation
  - between *M* and M_{SUT} is a binary relation $\sigma \subseteq S \times S'$, s.t. for all $(s, s') \in \sigma$
    - $A(s) \subseteq A'(s')$
    - For each $t \in S : P(s, a, t) > 0$, there exists $t' \in S' : P'(s', a, t') > 0$ and $(t, t') \in \sigma$

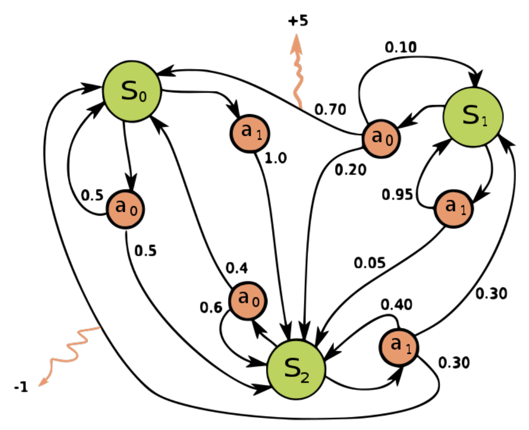
M



$$\begin{tikzcd} s \arrow[r, dashed, no head, "{(s, s') \in \sigma}"] \arrow[d, "a"] & s' \arrow[d, "a"] \\ t \arrow[r, dashed, no head, "{(t, t') \in \sigma}"] \arrow[d, "a'"] & t' \arrow[d, "a'"] \\ {} & {} \end{tikzcd}$$

...

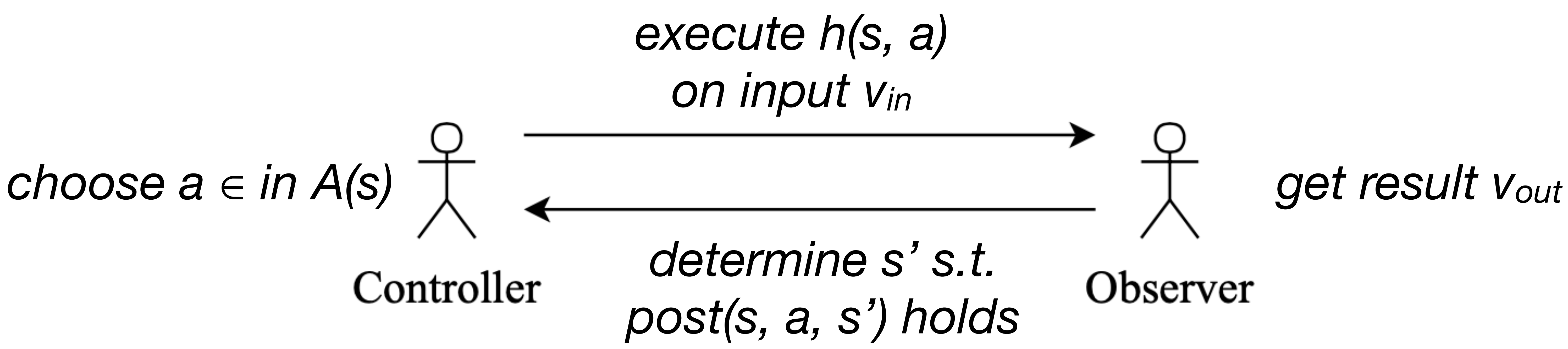
M_{SUT}



Online MBT with MDPs (4)

- Refinement
  - M_{SUT} refines M iff there exists a probabilistic alternating simulation σ s.t. $(s_0, s_0') \in \sigma$

- Conformance game
  - The notion of refinement is verified in practice by means of a conformance game between
    - Controller —> chooses actions based on a given test case generation strategy
    - Observer —> verifies the result out of a test execution



Online MBT under Uncertainty

- Problem statement
- Uncertain model paramenters
- Bayesian inference
- Framework and test case generation strategies

Problem statement

$$M \in SPEC \xrightarrow{\text{dev process}} SUT$$

Is it complete?
or is it based on partial knowledge
of the phenomenon of interest?

How to perform MBT if part of
M is uncertain?

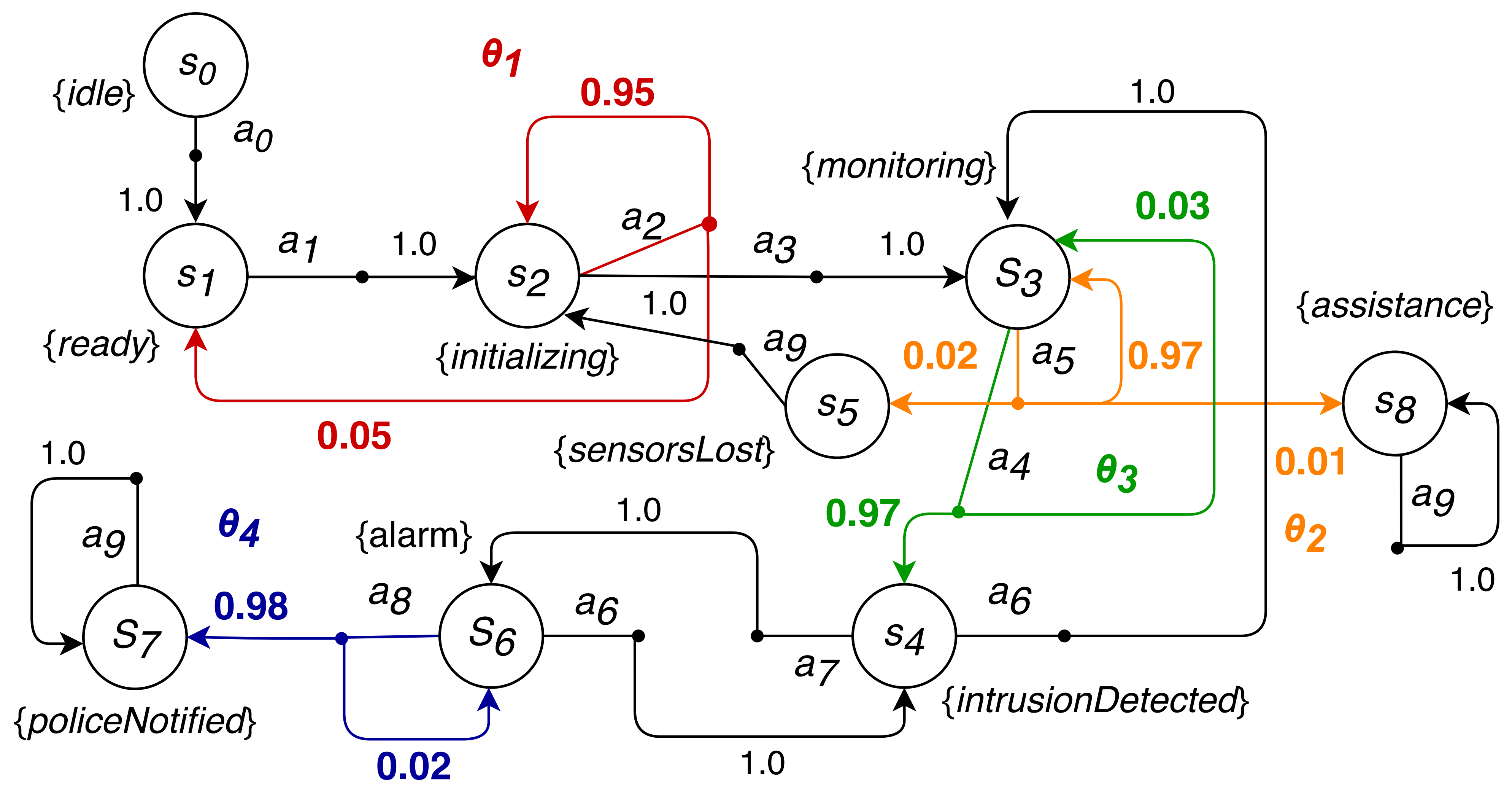
- Problem
  - Design-time models are imperfect and include **assumptions**
  - Assumptions are affected by sources of **uncertainty**
    - Uncertain **system** properties (e.g., algorithmic/structural uncertainty, performance)
    - Uncertain **environment** properties (e.g., usage profiles, failure rate of 3rd-party components, latency)

The very idea

- Objective
  - Reduce the discrepancy between design-time assumptions (uncertain mathematical models) and real-world entities (runtime evidence) by applying Inverse Uncertainty Quantification (IUQ)
- Assumption
  - sources of uncertainty affect model parameters (i.e., uncertain transition probability values in the MDP)
- How
  - Calibration of uncertain model parameters during integration/system testing by combining
    - Online Model-based Testing
    - Bayesian inference

Uncertain model parameters — SafeHome

- Sources of uncertainty (in CPSs) [1]
  - Application level — uncertain events/data generated from software running upon physical units (e.g., θ_3)
  - Infrastructure level — uncertain reliability of networking and/or cloud infrastructure (e.g., θ_4)
  - Integration level — uncertain outcomes from interacting physical units (e.g., θ_1, θ_2)



Examples
θ_2: uncertain sensing capability from the monitoring state
θ_4: uncertain failure rate of police notification from the alarm state

1. Man Zhang, et al., Uncertainty-wise cyber-physical system test modeling. Software & Systems Modeling (2017), 1–40

Uncertain regions

- Uncertain regions
  - Uncertain transition probabilities θ_i grouped by <src-state, action>
  - Values in θ_i are uncertain parameters of a Categorical distribution
    - $\theta_i \sim Cat(p_1, \ldots, p_k)$

region	state-action	affected level	target states	probability values
θ_1	s_2-a_2	integration	s_2, s_1	0.95, 0.05
θ_2	s_3-a_4	integration	s_3, s_4	0.03, 0.97
θ_3	s_3-a_5	application	s_3, s_5, s_8	0.01, 0.97, 0.02
θ_4	s_6-a_8	infrastructure	s_6, s_7	0.02, 0.98

Region

Uncertain parameters of a Categorical distribution

—

hypothesis

- Intuition
  - Mitigate the uncertainty over θ regions by observing (multiple times) the SUT
  - Observation provides evidence to increase the confidence on transition probabilities

Bayesian inference

- Method used to **update the probability** for a **hypothesis** as more **evidence** becomes available
- Formulation [1]
  - To learn θ (phenomenon of interest) we collect a sample $y = (y_1, \ldots, y_n)$
  - *Posterior* ∝ *likelihood* x *Prior*
    - Prior $f(\theta)$ — *hypothesis on* θ
    - Likelihood $f(y \mid \theta)$ — *compatibility of the evidence with the given hypothesis*
    - Posterior $f(\theta \mid y)$ — ***best knowledge*** *on the hypothesis given the evidence*
- In our context
  - The natural conjugate **Prior** of the **Categorical** distribution is the **Dirichlet** distribution
  - $\text{Prior}_{\theta i} \sim Dir(\alpha_1, \ldots, \alpha_K)$
    - e.g., uninformative $\text{Prior}_{\theta 3} \sim Dir(0.5, 0.5, 0.5)$
    - e.g., informative $\text{Prior}_{\theta 3} \sim Dir(1.0, 97.0, 2.0)$
      100 observations = 1 s_3, 97 s_5, 2 s_8

region	state-action	affected level	target states	probability values
θ_1	s_2-a_2	integration	s_2, s_1	0.95, 0.05
θ_2	s_3-a_4	integration	s_3, s_4	0.03, 0.97
θ_3	s_3-a_5	application	s_3, s_5, s_8	0.01, 0.97, 0.02
θ_4	s_6-a_8	infrastructure	s_6, s_7	0.02, 0.98

1. Robert, Christian. The Bayesian choice: from decision-theoretic foundations to computational implementation. Springer Science & Business Media, 2007

Bayesian inference (2)

- In our context
  - Updating rule Prior$_{\theta i}$ ~ *Dir*(α$_1$, ..., α$_K$) —> Post$_{\theta i}$ ~ *Dir*(α$_1$ + n$_1$, ..., α$_K$ + n$_k$)
    - e.g., *Dir*(1.0, 97.0, 2.0) —> Post$_{\theta i}$ ~ *Dir*(1.0 + 35, 97.0 + 955, 2.0 + 10)
      1000 observations = 35 s$_3$, 955 s$_5$, 10 s$_8$
- Summarization
  - Prior/Posterior knowledge can be summarized by using
    - Mean — transition probability values $p_i = \alpha_i / \sum_{j=1}^{k} \alpha_j$
    - HPD region — degree of confidence $C = \{p : f(\cdot) \geq 0.95\}$

region	state-action	affected level	target states	probability values
θ_1	s_2-a_2	integration	s_2, s_1	0.95, 0.05
θ_2	s_3-a_4	integration	s_3, s_4	0.03, 0.97
θ_3	s_3-a_5	application	s_3, s_5, s_8	0.01, 0.97, 0.02
θ_4	s_6-a_8	infrastructure	s_6, s_7	0.02, 0.98

Prior mean

inference

Posterior mean

0.033, 0.956, 0.011

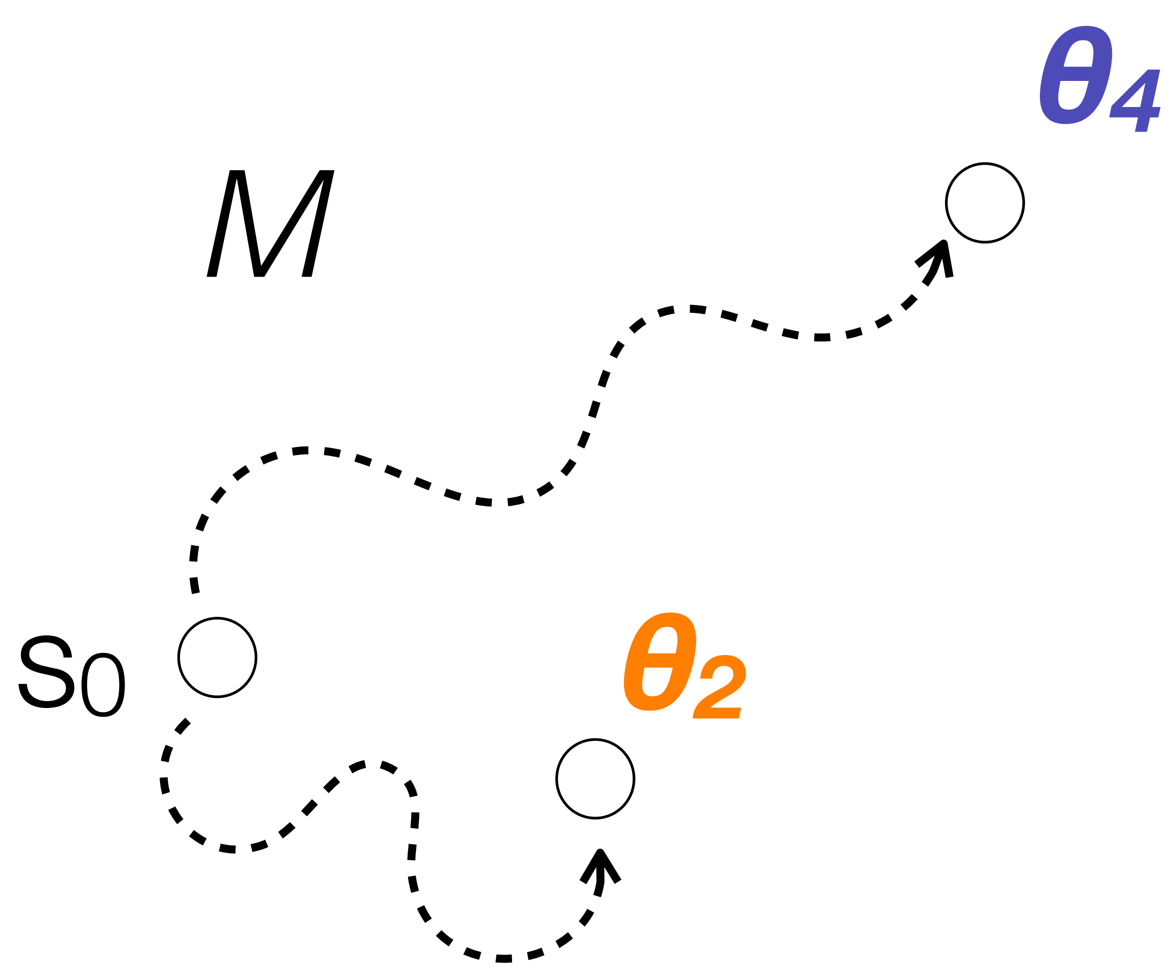
Prior HPD region = { [0.009, 0.019], [0.936, 0.996], [0.001, 0.047] }
Prior HPD width = 0.136

Posterior HPD region = { [0.009, 0.019], [0.936, 0.996], [0.001, 0.047] }
Posterior HPD width = 0.056

Lower value -> higher confidence

Online MBT + Bayesian inference

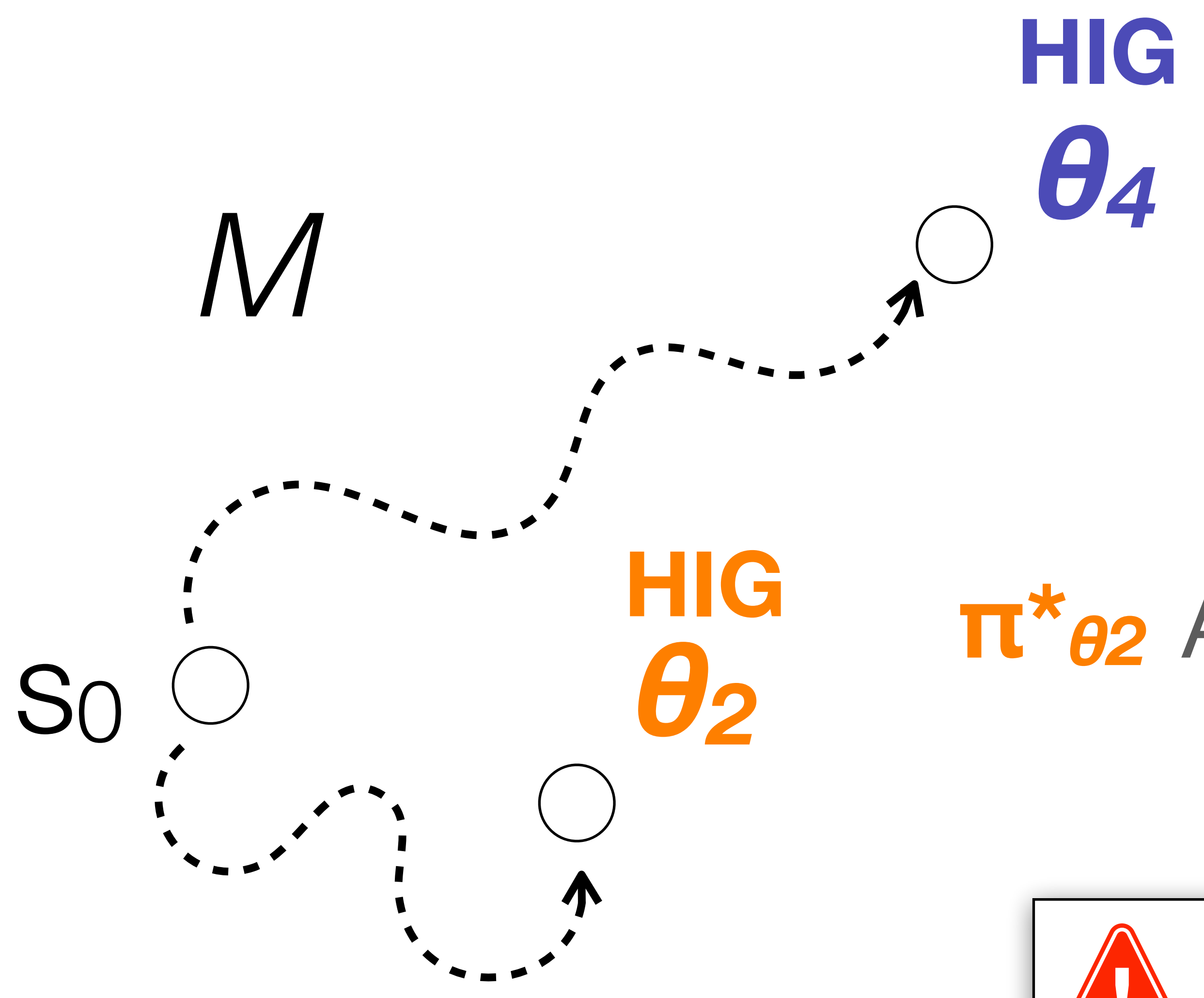
- Objective (reminder)
  - design-time assumptions <— gap reduction —> runtime evidence
- How
  - Perform a controlled exploration using online MBT to stress the uncertain components
  - Gather evidence and run bayesian inference to reduce the uncertainty



- Uncertainty-aware MBT strategy
  - Explore by maximize the probability of reaching θ regions
  - Reduces to an optimization problem:
    - Find out the actions a decision maker should take to maximize the exploration of θ regions

Uncertainty-aware strategy

- Computation of the best policies
  - For each θ_i
    - construct a **reward structure** that assigns **HIG** reward to θ_i transitions, **LOW** elsewhere
    - Compute the **best policy** $\pi^*_{\theta_i}$ (value iteration)
      - For each state, it selects the action that maximizes the probability to reach θ_i



$\pi^*_{\theta_4}$ Actions maximizing the probability to reach θ_4

$\pi^*_{\theta_2}$ Actions maximizing the probability to reach θ_2

Back to our mental note

We'll see how to leverage rewards in a "unconventional" way to drive testing.

Uncertainty-aware strategy (2)

- How to combine the best policies $\pi^*_{\theta i}$?
  - Simple scenario —> there exists just a single θ region
  - Otherwise —> different exploration strategies may be constructed/adopted
    - Strategies represent decision makers (i.e., testers) that use a probabilistic function

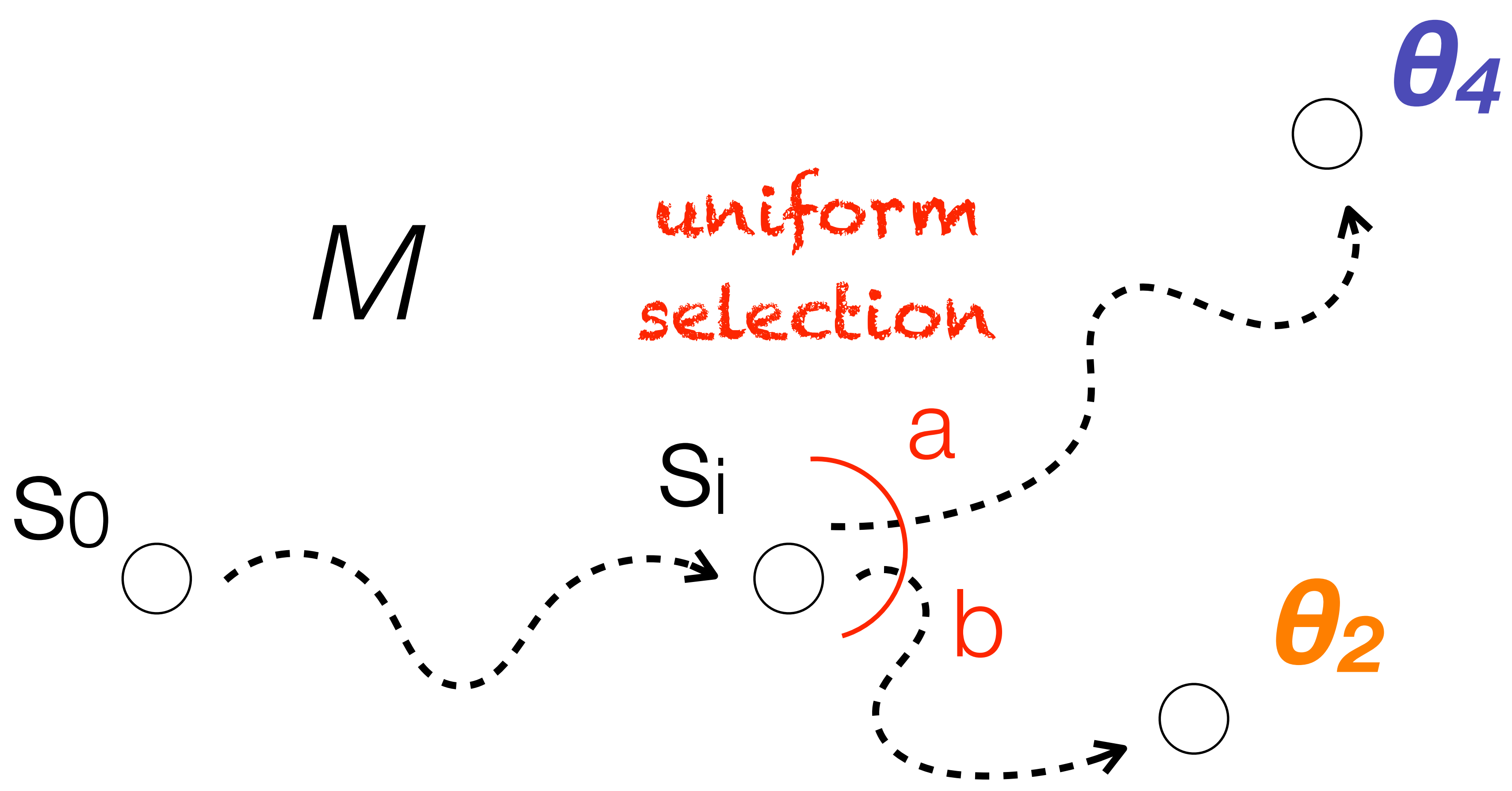
$$\mathcal{P}(s,a) = \begin{cases} 0 & \omega(s,a) = 0 \\ \omega(s,a) / \sum_{a' \in A(s)} \omega(s,a') & \textit{otherwise} \end{cases}$$

- The ω weight selectively increase/decrease the probability of choosing a specific action *a* from state *s*

Uncertainty-aware strategy (3)

- Flat strategy
  - Actions selected by different policies $\pi^*_{\theta i}$ have equal probability
  - **Uniform random sampling** of the available policies

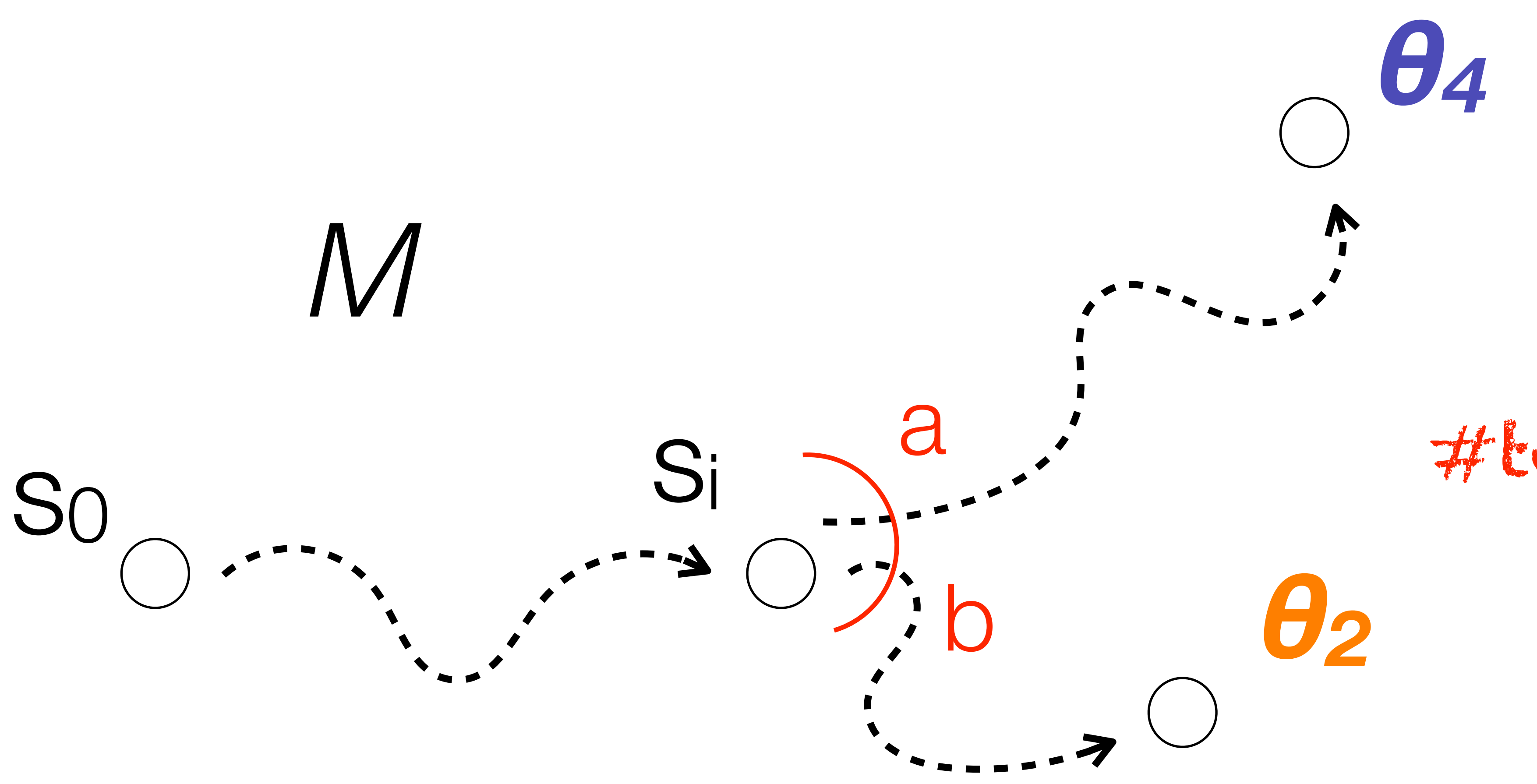
$$\omega^{RT}(s,a) = \begin{cases} 1 & \exists i : \pi^*_i(s) = a \\ 0 & otherwise \end{cases}$$



Uncertainty-aware strategy (4)

- History-based strategy
  - Tries to **keep balanced** the number of times θ regions are tested
  - We leverage decrementing weights **inversely proportional to #selections** of state-action pairs

$$\omega^{HT}(s,a) = \begin{cases} 1/\#(s,a) & \exists i : \pi_i^*(s) = a \\ 0 & otherwise \end{cases}$$



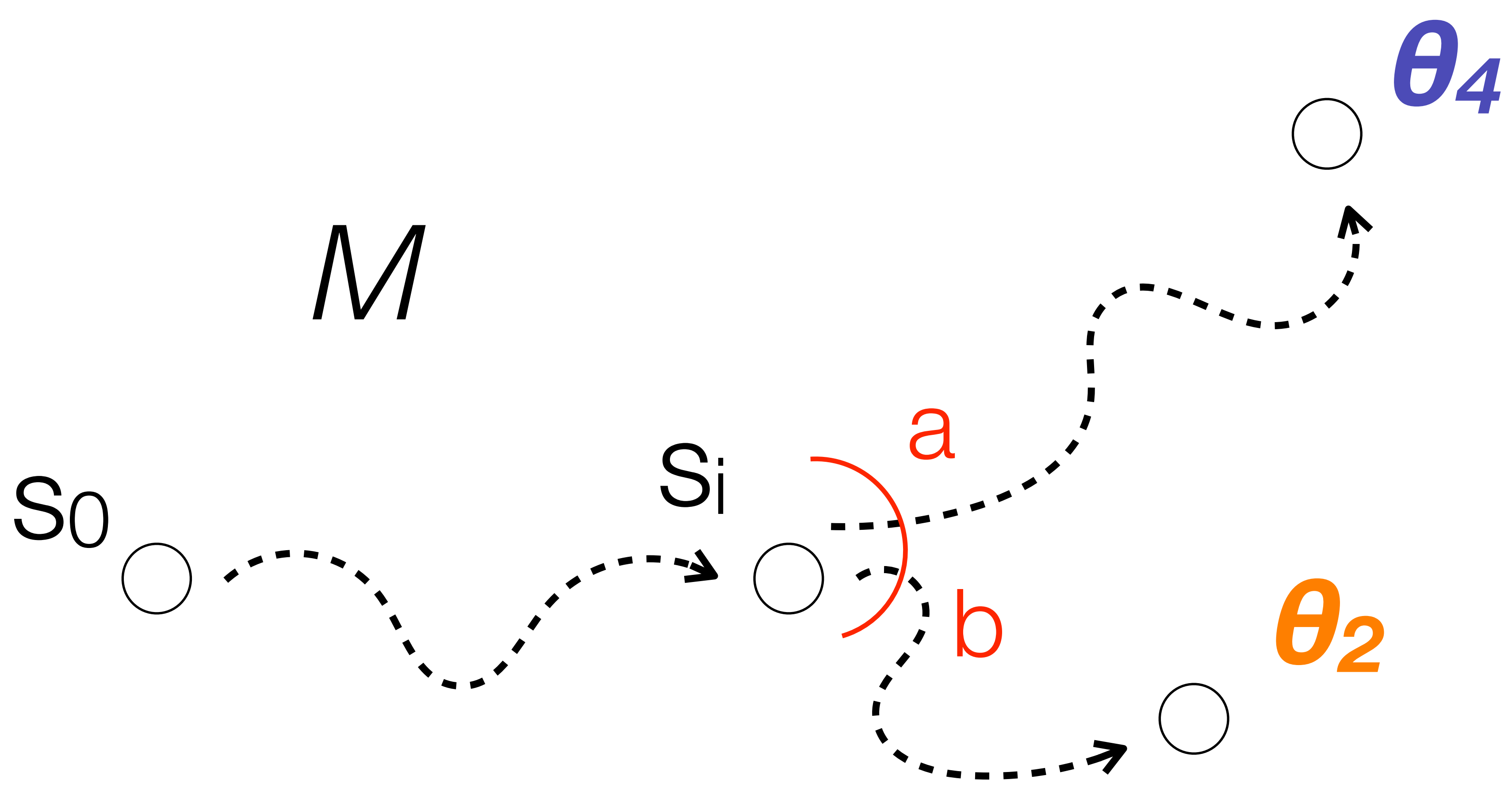
#tests θ_4 > #tests θ_2

the higher the #selections of a,
the lower the likelihood of selecting it again

Uncertainty-aware strategy (5)

- Distance strategy
  - Tries to deliver **balanced degree of confidence** on θ regions
  - The weight is **proportional to the HPD width** of θ regions

$$\omega^{DT}(s,a) = \begin{cases} |\mathrm{HPD}_{\theta_i}| & \exists i : \pi_i^*(s) = a \\ 0 & otherwise \end{cases}$$



confidence θ_2 > confidence θ_4

the larger the HPD width of the target θ, the higher the likelihood of selecting it

Termination condition

- Limit on the effort
  - Traditional termination condition based on #tests limit
- Bayes factor
  - Tries to recognize when the inference process converges

$$\mathcal{F} = \frac{f(y|\theta)}{f(y|\theta')}$$

$$\mathcal{F} \in [10^0, 10^{1/2}]$$

likelihood that data y are produced under different assumptions θ and θ′

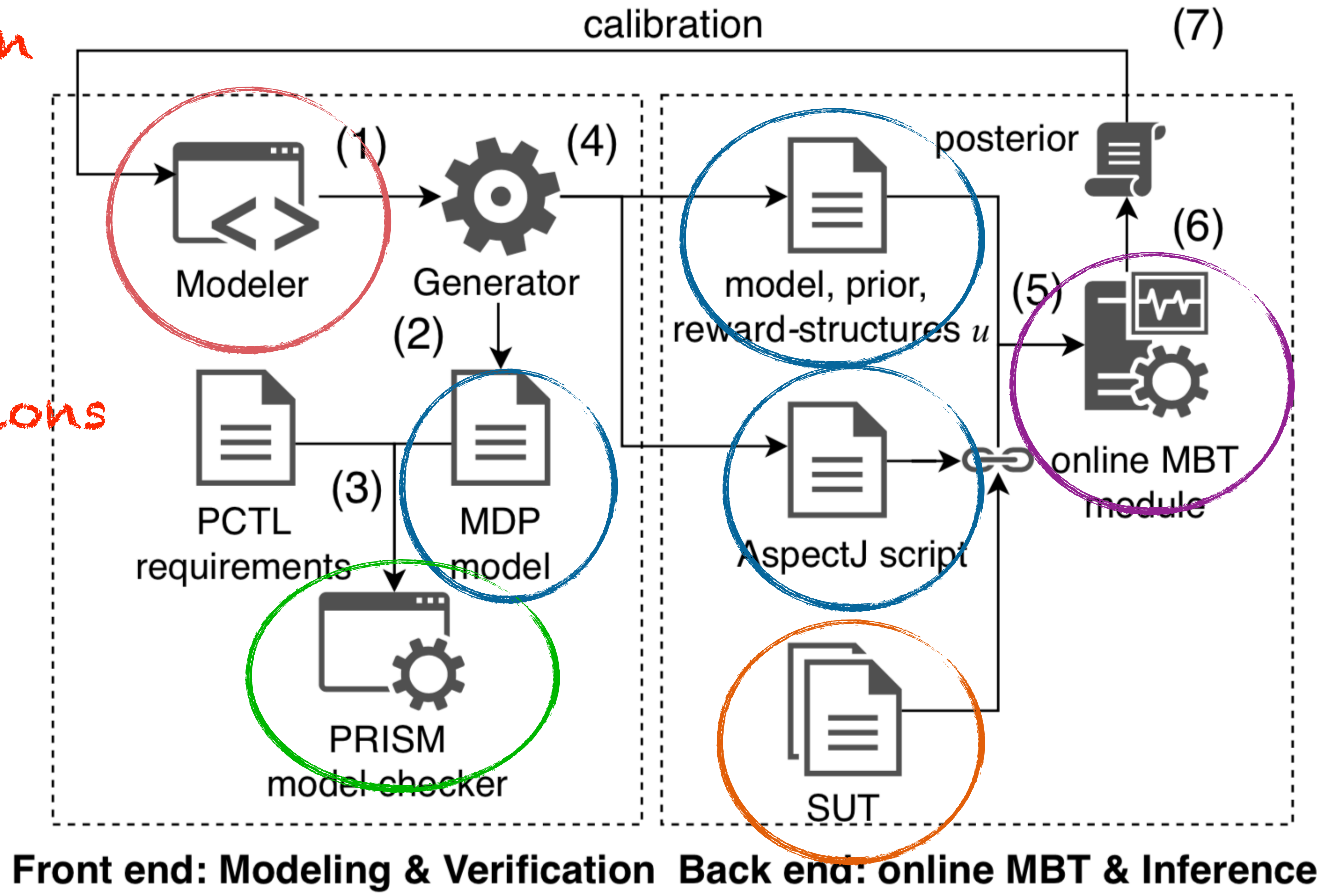
Difference between assumptions θ and θ′ is not substantial

Current toolchain implementation

MDP definition + uncertain transition probabilities θ

binding to the SUT:
actions -> inputs
acs -> routine postconditions

automatic generation



online MBT and inference/ calibration

model checking of PCTL requirements

Java program

Current toolchain implementation (2)

MBT module

https://github.com/SELab-unimi/mbt-module

Java 99.3%
Other 0.7%

- Current stage
  - Uncertainty-aware strategies have been implemented
  - Systematic evaluation of their cost-effectiveness —> currently inedited

Current toolchain implementation (3)

- Evaluation summary
  - We assessed **statistical difference** (Mann-Whitney U test [1])
  - We evaluated **practical value** (Vargha & Delaney's $\hat{A}_{12}$ measure [1])
    - In **our context** —> assuming same effort (i.e., #tests), the probability that **target strategy** yields **smaller HPD width** values than **flat** one (i.e., baseline)

Table 3: Vargha and Delaney's $\hat{A}_{12}$ measure

	%uncertainty			#actions		
balanced	20	50	80	5	10	20
hist	1.000	0.716	0.531	0.617	0.704	0.926
dist	1.000	0.790	0.679	0.741	0.802	0.951
unbalanced	20	50	80	5	10	20
hist	0.963	0.716	0.556	0.531	0.642	0.901
dist	0.988	0.951	0.691	0.741	0.741	0.975

1. Andrea Arcuri and Lionel Briand, A practical guide for using statistical tests to assess randomized algorithms in software engineering, ICSE'11, New York, NY, USA

Summary

- We discussed MBT for probabilistic systems and the problem of testing with uncertain model components
- Depending on the Prior knowledge (hypothesis) and information that can be gathered during testing, we derived different uncertainty-aware exploration strategies and evaluated their cost-effectiveness
  - Flat —> uniform selection
  - History —> balanced exploration
  - Distance —> balanced delivered confidence
- Next
  - Hands on session with the MBT module
  - Design/develop an additional exploration strategy

One more thing: collaboration opportunities

- Relationship among uncertain regions
  - Identify the uncertain regions depending on other uncertain regions along execution paths
  - Compute Posterior probabilities following the discovered relationships
- Refactoring of MDP models
  - Identify critical portion of the models w.r.t. requirements
  - Model-based refactoring and evaluation
- Automatic construction of MDP model from the implementation
  - Static analysis or symbolic execution
- Testing using Reinforcement Learning (based on MDP theory)
  - Discover the location of the uncertain regions

References

- Camilli M., Gargantini A., Scandurra P., Bellettini C. (2017) Towards Inverse Uncertainty Quantification in Software Development (Short Paper). In: Cimatti A., Sirjani M. (eds) Software Engineering and Formal Methods. SEFM 2017. LNCS, vol 10469. Springer, Cham.
- M. Camilli, C. Bellettini, A. Gargantini and P. Scandurra, (2018) Online Model-Based Testing under Uncertainty, IEEE 29th International Symposium on Software Reliability Engineering (ISSRE), Memphis, TN, 2018, pp. 36-46.
- M. Camilli, A. Gargantini, R. Madaudo and P. Scandurra, (2019) HYPpOTesT: Hypothesis Testing Toolkit for Uncertain Service-based Web Applications. In proceeding of the 15th International conference on integrated Formal Methods. iFM2019. LNCS, vol 11918. Springer, Cham.
- M. Camilli, A. Gargantini, and P. Scandurra, (2020) Model-based Hypothesis Testing of Uncertain Software Systems, Software Testing Verification and Reliability, John Wiley & Sons, Ltd. https://doi.org/10.1002/stvr.1730, To appear.

License of these slides