Approximate Program Transformation

Alessandra Di Pierro

Università di Verona

29 May 2013

Outline

Timing Attacks

Cryptosystems often take different amounts of time to process different inputs.

Timing Attacks

Cryptosystems often take different amounts of time to process different inputs.

Diffie-Hellman and RSA private-key operations consist of computing $R = y^x \bmod n$, where n is public and y can be found by an eavesdropper. The attacker's goal is to find x, the secret.

Timing Attacks

Cryptosystems often take different amounts of time to process different inputs.

Diffie-Hellman and RSA private-key operations consist of computing $R = y^x \bmod n$, where n is public and y can be found by an eavesdropper. The attacker's goal is to find x, the secret.

By passively eavesdropping on an interactive protocol, an attacker can obtain all necessary information and timing measurements.

Timing Attacks

Cryptosystems often take different amounts of time to process different inputs.

Diffie-Hellman and RSA private-key operations consist of computing $R = y^x \bmod n$, where n is public and y can be found by an eavesdropper. The attacker's goal is to find x, the secret.

By passively eavesdropping on an interactive protocol, an attacker can obtain all necessary information and timing measurements.

Timing channels can leak data or keys revealing information from a crypto system such as the Hamming weight of the key or even the entire secret key.

Timing Attacks

Cryptosystems often take different amounts of time to process different inputs.

Diffie-Hellman and RSA private-key operations consist of computing $R = y^x \bmod n$, where n is public and y can be found by an eavesdropper. The attacker's goal is to find x, the secret.

By passively eavesdropping on an interactive protocol, an attacker can obtain all necessary information and timing measurements.

Timing channels can leak data or keys revealing information from a crypto system such as the Hamming weight of the key or even the entire secret key.
P.C. Kocher: *Cryptanalysis of Diffie-Hellman, RSA, DSS, and other cryptosystems using timing attacks*, CRYPTO '95.

A simple Algorithm for ME

The simple modular exponentiation algorithm below computes $R = y^x \mod n$, where x is w bits long:

```
Let s0 = 1.
For k = 0 upto w-1:
If (bit k of x) is 1 then
Let Rk = (sk * y) mod n.
Else
Let Rk = sk.
Let sk+1 = Rk*Rk mod n.
EndFor.
Return (Rw-1).
```

Masking Time Characteristics

The most obvious way to prevent timing attacks is to make all operations take exactly the same amount of time.

Masking Time Characteristics

The most obvious way to prevent timing attacks is to make all operations take exactly the same amount of time.
Unfortunately,

Masking Time Characteristics

The most obvious way to prevent timing attacks is to make all operations take exactly the same amount of time.
Unfortunately,

- making software run in fixed time is hard;

Masking Time Characteristics

The most obvious way to prevent timing attacks is to make all operations take exactly the same amount of time.
Unfortunately,

- making software run in fixed time is hard;
- fixed-time implementation are likely to be slow

Masking Time Characteristics

The most obvious way to prevent timing attacks is to make all operations take exactly the same amount of time.
Unfortunately,

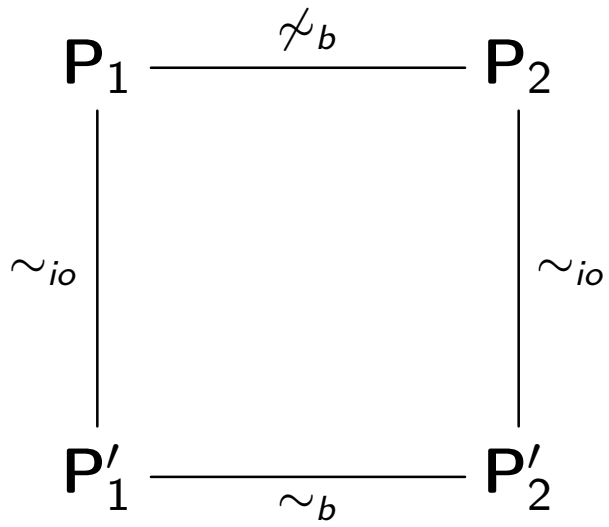
- making software run in fixed time is hard;
- fixed-time implementation are likely to be slow
  (all operations must take as long as the slowest operation).

Program Obfuscation

In order to prevent timing attacks we can 'garble' a program so as to produce another program with the same I/O behaviour as the original one but internally different.

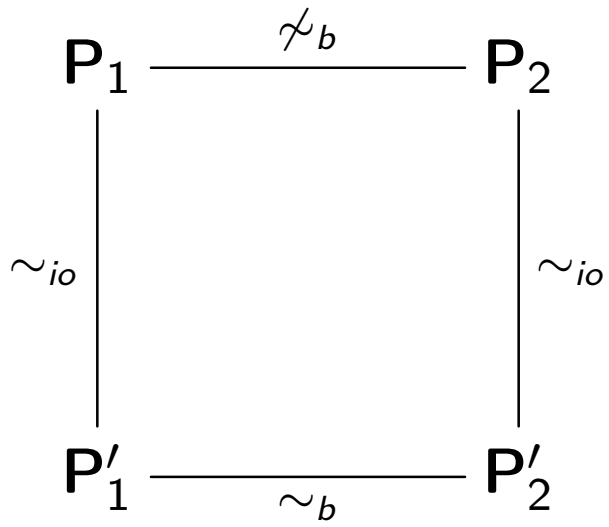
Program Obfuscation

In order to prevent timing attacks we can 'garble' a program so as to produce another program with the same I/O behaviour as the original one but internally different.



Program Obfuscation

In order to prevent timing attacks we can 'garble' a program so as to produce another program with the same I/O behaviour as the original one but internally different.



An adversarial entity that look at an obfuscated version P' of a program P is unable to understand the internal working of P.

Outline

Probabilistic Transition Systems

A model for the semantics of probabilistic languages is given via Probabilistic Transition Systems (PTS).

Probabilistic Transition Systems

A model for the semantics of probabilistic languages is given via Probabilistic Transition Systems (PTS).

A Probabilistic Transition System is a tuple $(S, A, \longrightarrow, \pi_0)$:

- S is a non-empty, set of states,
- A is a non-empty, finite set of actions,
- $\longrightarrow \subseteq S \times Dist(A \times S)$ is a transition relation,
- $\pi_0 \in Dist(S)$ is an initial distribution on S.

Matrix Representation

Definition

The matrix representation of a PTS $p = (S, A, \longrightarrow, \pi_0)$ is

$$\mathbf{M}(p) = \bigoplus_{\alpha \in A} \mathbf{M}(\xrightarrow{\alpha}).$$

Matrix Representation

Definition

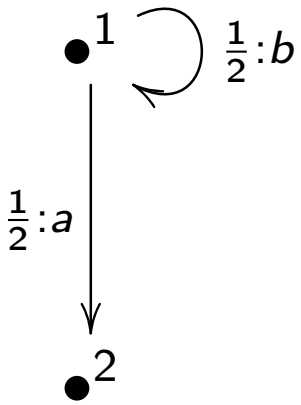
The matrix representation of a PTS $p = (S, A, \longrightarrow, \pi_0)$ is

$$\mathbf{M}(p) = \bigoplus_{\alpha \in A} \mathbf{M}(\xrightarrow{\alpha}).$$

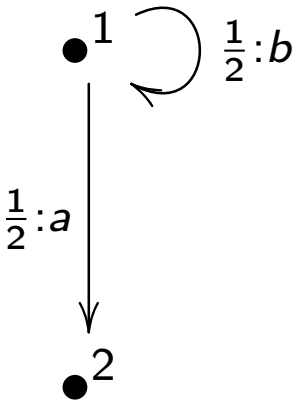
The direct sum of a set $\{\mathbf{M}_i\}_{i=1}^{k}$ of $n_i \times m_i$ matrices, is defined by the $(\sum_{i=1}^{k} n_i) \times (\sum_{i=1}^{k} m_i)$ matrix:

$$\mathbf{M} = \bigoplus_{i} \mathbf{M}_i = \begin{pmatrix} \mathbf{M}_1 & 0 & 0 & \dots & 0 \\ 0 & \mathbf{M}_2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \mathbf{M}_k \end{pmatrix}$$

Encoding a PTS into a DTMC

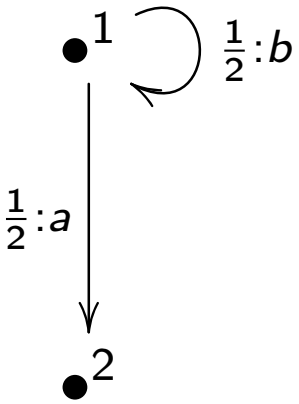


Encoding a PTS into a DTMC



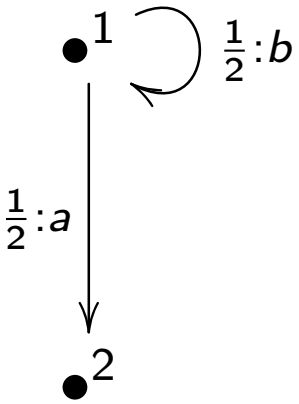
$$\mathbf{M}(A) = \mathbf{M}_a(A) \oplus \mathbf{M}_b(A)$$

Encoding a PTS into a DTMC



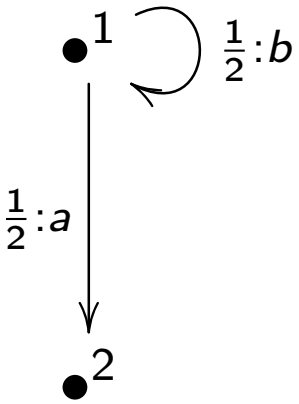
$$\mathbf{M}(A) = \begin{pmatrix} 0 & \frac{1}{2} \\ 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 0 \end{pmatrix}$$

Encoding a PTS into a DTMC



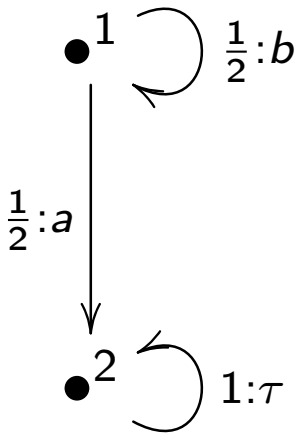
$$\mathbf{M}(A) = \begin{pmatrix} \begin{pmatrix} 0 & \frac{1}{2} \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 0 \end{pmatrix} \end{pmatrix}$$

Encoding a PTS into a DTMC

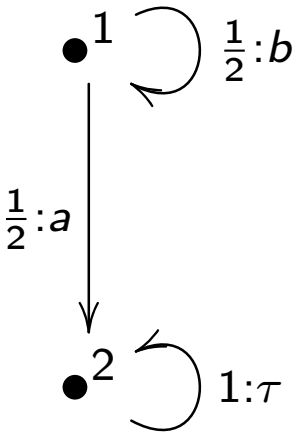


$$\mathbf{M}(A) = \begin{pmatrix} 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Encoding a PTS into a DTMC

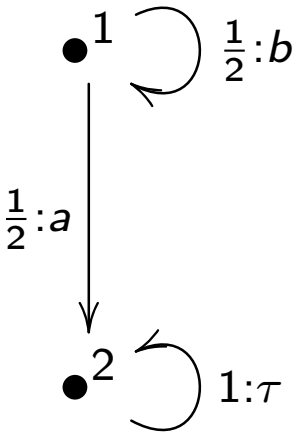


Encoding a PTS into a DTMC



$$\mathbf{M}(A) = \mathbf{M}_a(A) \oplus \mathbf{M}_b(A) \oplus \mathbf{M}_\tau(A)$$

Encoding a PTS into a DTMC



$$\mathbf{M}(A) = \begin{pmatrix} 0 & \frac{1}{2} \\ 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

Outline

Classical Bisimulation

A bisimulation is a binary relation $\sim_b$ on states of a labelled transition system satisfying for all $\alpha \in A$:

$$p \sim_b q \text{ and } p \xrightarrow{\alpha} p' \Rightarrow \exists q' : q \xrightarrow{\alpha} q' \text{ and } p' \sim_b q',$$
$$p \sim_b q \text{ and } q \xrightarrow{\alpha} q' \Rightarrow \exists p' : p \xrightarrow{\alpha} p' \text{ and } q' \sim_b p'.$$

Probabilistic Bisimulation

A probabilistic bisimulation is an equivalence relation $\sim_b$ on states of a PTS satisfying for all $\alpha \in A$:

$$p \sim_b q \quad \text{and} \quad p \longrightarrow_\alpha \pi$$
$$\Rightarrow$$
$$q \longrightarrow_\alpha \varrho \quad \text{and} \quad \pi \sim_b \varrho,$$

Probabilistic Bisimulation

A probabilistic bisimulation is an equivalence relation $\sim_b$ on states of a PTS satisfying for all $\alpha \in A$:

$$p \sim_b q \quad \text{and} \quad p \longrightarrow_\alpha \pi$$
$$\Rightarrow$$
$$q \longrightarrow_\alpha \varrho \quad \text{and} \quad \pi \sim_b \varrho,$$

where

$$\pi \sim_b \varrho \quad \text{iff} \quad \forall [s] \in S/_{\sim_b} : \pi([s]) = \varrho([s]).$$

Probabilistic Bisimulation as PAI

Given the operator representation $\mathbf{M}(X)$ and $\mathbf{M}(Y)$ of two PTS's X and Y, then X and Y are (probabilistically) bisimilar iff there exist classification matrices $\mathbf{K}_X$ and $\mathbf{K}_Y$ such that:

$$\mathbf{K}_X^\dagger \mathbf{M}(X)\mathbf{K}_X = \mathbf{K}_Y^\dagger \mathbf{M}(Y)\mathbf{K}_Y$$

Probabilistic Bisimulation as PAI

Given the operator representation $\mathbf{M}(X)$ and $\mathbf{M}(Y)$ of two PTS's X and Y, then X and Y are (probabilistically) bisimilar iff there exist classification matrices $\mathbf{K}_X$ and $\mathbf{K}_Y$ such that:

$$(\bigoplus_{\alpha} \mathbf{K}_X^\dagger)\mathbf{M}(X)(\bigoplus_{\alpha} \mathbf{K}_X) = (\bigoplus_{\alpha} \mathbf{K}_Y^\dagger)\mathbf{M}(Y)(\bigoplus_{\alpha} \mathbf{K}_Y)$$

Probabilistic Bisimulation as PAI

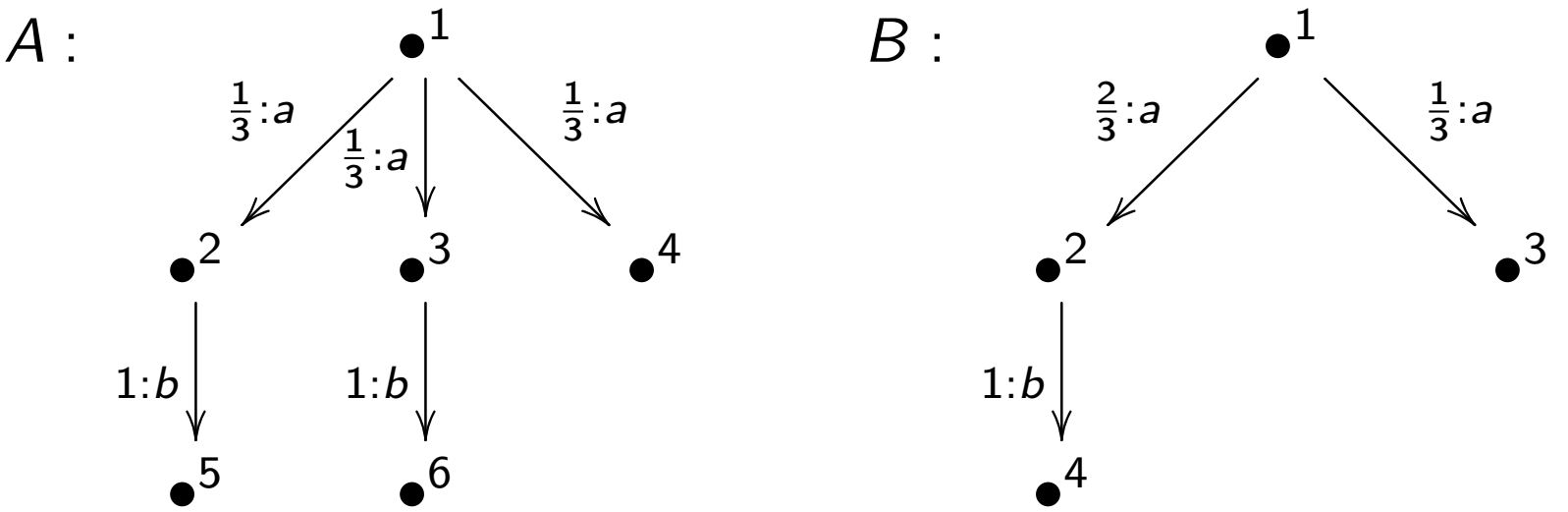
Given the operator representation $\mathbf{M}(X)$ and $\mathbf{M}(Y)$ of two PTS's X and Y, then X and Y are (probabilistically) bisimilar iff there exist classification matrices $\mathbf{K}_X$ and $\mathbf{K}_Y$ such that:

$$\mathbf{K}_X^\dagger \mathbf{M}(X)\mathbf{K}_X = \mathbf{K}_Y^\dagger \mathbf{M}(Y)\mathbf{K}_Y$$

We call an $n \times m$-matrix $\mathbf{K}$ a classification matrix iff $\mathbf{K}$ represents a surjective function $\kappa : \{1, \ldots, n\} \to \{1, \ldots, m\}$

$$\mathbf{K}_{ij} = \begin{cases} 1 & \text{if } j = \kappa(i) \\ 0 & \text{otherwise.} \end{cases}$$

Example



Example

$$\mathbf{M}(A) = \mathbf{M}_a(A) \oplus \mathbf{M}_b(A) =$$

$$= \begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Example

$$\mathbf{M}(B) = \mathbf{M}_a(B) \oplus \mathbf{M}_b(B) =$$

$$= \begin{pmatrix} 0 & \frac{2}{3} & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Example

$$\mathbf{K}_A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{K}_A^{\dagger} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Example

$$\mathbf{K}_A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{K}_A^\dagger = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

$$\mathbf{K}_A^\dagger \mathbf{M}(A) \mathbf{K}_A = \mathbf{M}(B)$$

Example

$$\mathsf{K}_A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathsf{K}_A^\dagger = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

$$\begin{aligned} \mathsf{K}_A^\dagger \mathsf{M}_a(A) \mathsf{K}_A &= \mathsf{M}_a(B) \\ \mathsf{K}_A^\dagger \mathsf{M}_b(A) \mathsf{K}_A &= \mathsf{M}_b(B) \end{aligned}$$

Outline

Security Types

Security levels	$s ::= \mathsf{L} \mid \mathsf{H}$
Base types	$\overline{\tau} ::= \mathbf{Int} \mid \mathbf{Bool}$
Security types	$\tau ::= \overline{\tau}_s$

Low Equivalence

We associate to every variable x a security level

$$\Gamma(x) \in \bar{\tau}_{\{L,H\}}$$

Low Equivalence

We associate to every variable x a security level

$$\Gamma(x) \in \bar{\tau}_{\{L,H\}}$$

Identify environments which agree on the low variables:

$$E_1 =_L E_2 \quad \text{iff} \quad \forall l \text{ with } \Gamma(l) = \bar{\tau}_L : E_1(l) = E_2(l)$$

Low Equivalence

We associate to every variable x a security level

$$\Gamma(x) \in \bar{\tau}_{\{L,H\}}$$

Identify environments which agree on the low variables:

$$E_1 =_L E_2 \text{ iff } \forall l \text{ with } \Gamma(l) = \bar{\tau}_L : E_1(l) = E_2(l)$$

Identify configurations which agree on the low variables:

$$\langle E_1 \mid C \rangle =_L \langle E_2 \mid C \rangle \text{ iff } \forall l \text{ with } \Gamma(l) = \bar{\tau}_L : E_1(l) = E_2(l)$$

Lifted Bisimilarity

We have to deal for probabilistic programs we need to consider distributions **Dist(Conf)**.

Lifted Bisimilarity

We have to deal for probabilistic programs we need to consider distributions **Dist**(**Conf**).

An equivalence relation $\sim \subseteq S \times S$ on S can be lifted to an (equivalence) relation $\sim^* \subseteq \mathbf{Dist}(S) \times \mathbf{Dist}(S)$ on distributions on S via

$$\mu \sim^* \nu \text{ iff } \forall [s] \in S/_\sim : \mu([s]_\sim) = \nu([s]_\sim).$$

Probabilistic Time Bisimilarity

Define the transition relation $\Longrightarrow \in (\mathbf{Conf}, \mathbf{JDist})$ between configurations to joint distributions $\chi \in \mathbf{JDist} = \mathbf{Dist}(\mathbb{T} \times \mathbf{Conf})$, that is distributions on time $t \in \mathbb{T}$ and states $s \in \mathbf{Conf}$ such that $\chi(c', t) = p$ iff $c \stackrel{p:t}{\Longrightarrow} c'$.

Probabilistic Time Bisimilarity

Define the transition relation $\Longrightarrow \in (\mathbf{Conf}, \mathbf{JDist})$ between configurations to joint distributions $\chi \in \mathbf{JDist} = \mathbf{Dist}(\mathbb{T} \times \mathbf{Conf})$, that is distributions on time $t \in \mathbb{T}$ and states $s \in \mathbf{Conf}$ such that $\chi(c', t) = p$ iff $c \stackrel{p:t}{\Longrightarrow} c'$.

A probabilistic time bisimilarity $\sim_{PT} = \sim$ is the largest symmetric relation on configurations such that whenever $c_1 \sim c_2$, then

$$c_1 \Longrightarrow \chi_1 \text{ implies } \exists \chi_2 \text{ such that } c_2 \Longrightarrow \chi_2 \text{ and } \chi_1 \sim_L^* \chi_2$$

Probabilistic Time Bisimilarity

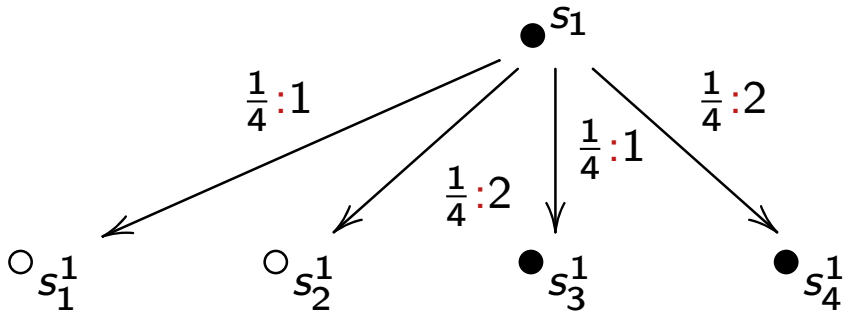
Define the transition relation $\Longrightarrow \in (\mathbf{Conf}, \mathbf{JDist})$ between configurations to joint distributions $\chi \in \mathbf{JDist} = \mathbf{Dist}(\mathbb{T} \times \mathbf{Conf})$, that is distributions on time $t \in \mathbb{T}$ and states $s \in \mathbf{Conf}$ such that $\chi(c', t) = p$ iff $c \stackrel{p:t}{\Longrightarrow} c'$.

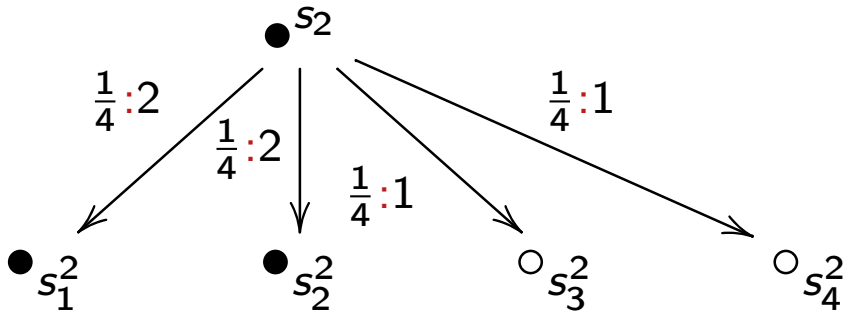
A probabilistic time bisimilarity $\sim_{PT} = \sim$ is the largest symmetric relation on configurations such that whenever $c_1 \sim c_2$, then

$$c_1 \Longrightarrow \chi_1 \text{ implies } \exists \chi_2 \text{ such that } c_2 \Longrightarrow \chi_2 \text{ and } \chi_1 \sim_L^* \chi_2$$

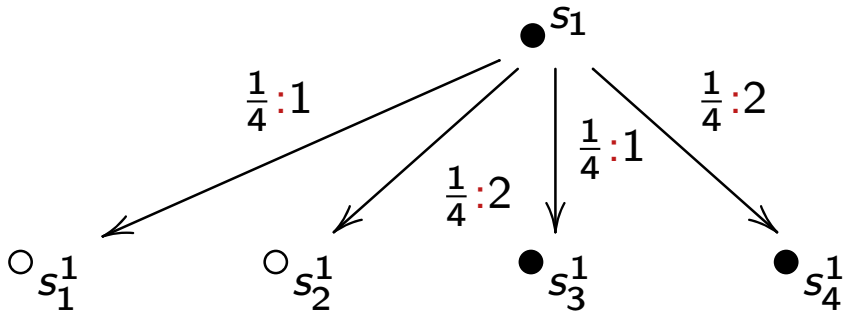
$$\text{with } \langle E_1 \mid C \rangle \sim_L \langle E_2 \mid C \rangle \text{ iff } \forall l \text{ with } \Gamma(l) = \bar{\tau}_L : E_1(l) = E_2(l)$$

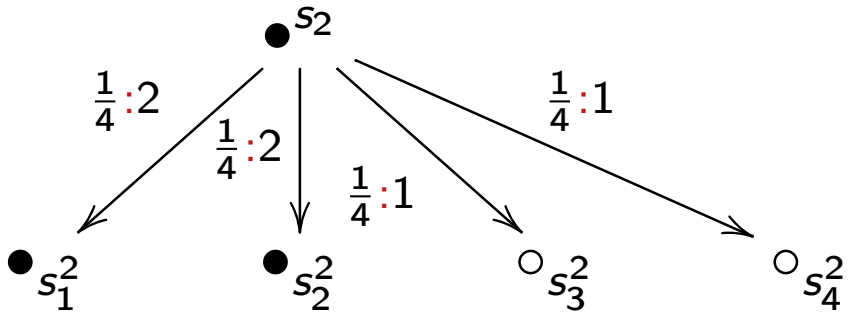
$$\sim_L^* = (\sim_L)^* \neq (\sim^*)_L$$





$$\sim_L^* = (\sim_L)^* \neq (\sim^*)_L$$

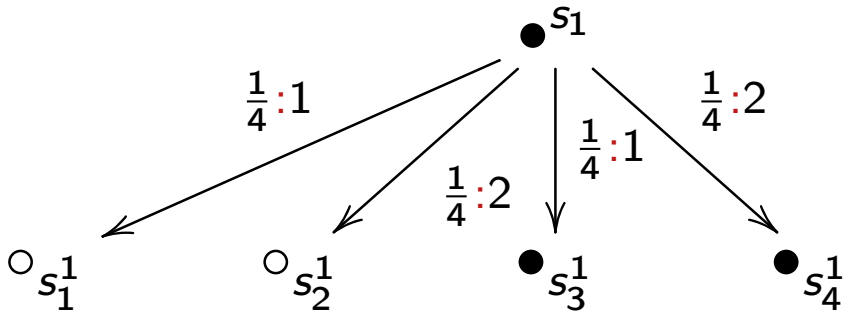


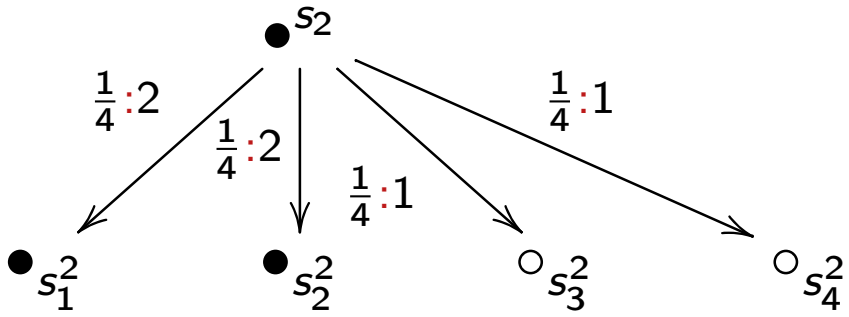


$\chi_1(t,l)$	1	2
●	$\frac{1}{4}$	$\frac{1}{4}$
○	$\frac{1}{4}$	$\frac{1}{4}$

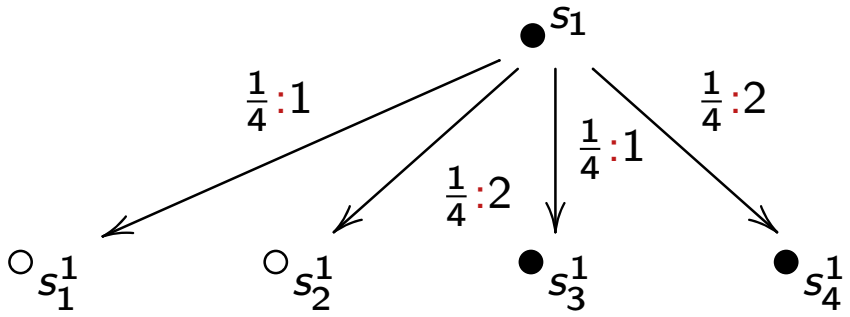
$\chi_2(t,l)$	1	2
●	0	$\frac{1}{2}$
○	$\frac{1}{2}$	0

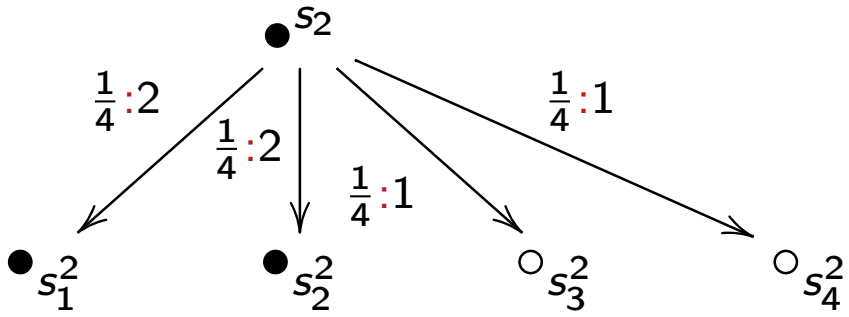
Probabilistic Time Bisimilarity





Probabilistic Time Bisimilarity





$\chi_1(t,l)$	1	2
●	$\frac{1}{4}$	$\frac{1}{4}$
○	$\frac{1}{4}$	$\frac{1}{4}$

$\chi_2(t,l)$	1	2
●	0	$\frac{1}{2}$
○	$\frac{1}{2}$	0

Bisimilarity and Security

Security of code C corresponds to bisimilar behaviour for all low-equivalent initial configurations.

$$\forall \langle E_1, C \rangle \sim_L \langle E_2, C \rangle \text{ we have } \langle E_1, C \rangle \sim_{PT} \langle E_2, C \rangle$$

Bisimilarity and Security

Security of code C corresponds to bisimilar behaviour for all low-equivalent initial configurations.

$$\forall \langle E_1, C \rangle \sim_L \langle E_2, C \rangle \text{ we have } \langle E_1, C \rangle \sim_{PT} \langle E_2, C \rangle$$

- Are T_1 and T_2 bisimilar?

Bisimilarity and Security

Security of code C corresponds to bisimilar behaviour for all low-equivalent initial configurations.

$$\forall \langle E_1, C \rangle \sim_L \langle E_2, C \rangle \text{ we have } \langle E_1, C \rangle \sim_{PT} \langle E_2, C \rangle$$

- Are T_1 and T_2 bisimilar?
- Can we make T_1 and T_2 bisimilar?

Checking Bisimulation

Checking whether two systems are bisimilar or not is important in many other fields, e.g.

Checking Bisimulation

Checking whether two systems are bisimilar or not is important in many other fields, e.g.
Model Checking: use bisimulation to minimise the states space.

Checking Bisimulation

Checking whether two systems are bisimilar or not is important in many other fields, e.g.
Model Checking: use bisimulation to minimise the states space.
Paige-Tarjan algorithm solves the problem in linear time.

Checking Bisimulation

Checking whether two systems are bisimilar or not is important in many other fields, e.g.
Model Checking: use bisimulation to minimise the states space.
Paige-Tarjan algorithm solves the problem in linear time.
R.Paige, R.Tarjan. *Three partition refinement algorithms*, SIAM Journal of Computation, 1987.

Checking Bisimulation

Checking whether two systems are bisimilar or not is important in many other fields, e.g.
Model Checking: use bisimulation to minimise the states space.
Paige-Tarjan algorithm solves the problem in linear time.
R.Paige, R.Tarjan. *Three partition refinement algorithms*, SIAM Journal of Computation, 1987.
Derisavi-Hermanns-Sanders algorithm constructs the optimal lumping quotient of a finite Markov chain in linear time.

Checking Bisimulation

Checking whether two systems are bisimilar or not is important in many other fields, e.g.
Model Checking: use bisimulation to minimise the states space.
Paige-Tarjan algorithm solves the problem in linear time.
R.Paige, R.Tarjan. *Three partition refinement algorithms*, SIAM Journal of Computation, 1987.
Derisavi-Hermanns-Sanders algorithm constructs the optimal lumping quotient of a finite Markov chain in linear time.
S.Derisavi, H.Hermanns, W.H. Sanders, *Optimal state-space lumping in Markov chains*, Information Processing Letters, 2003.

Partition Refinement

Construct a partition of a state space Σ which is stable for a given transition relation $\rightarrow$,

Partition Refinement

Construct a partition of a state space Σ which is stable for a given transition relation $\rightarrow$, that is it does not need further refinement.

Partition Refinement

Construct a partition of a state space Σ which is stable for a given transition relation $\rightarrow$, that is it does not need further refinement.

Theorem

The partition obtained by the PT algorithm corresponds to a bisimulation equivalence on the transition system $(\Sigma, \rightarrow)$.

Partition Refinement

Construct a partition of a state space Σ which is stable for a given transition relation $\rightarrow$, that is it does not need further refinement.

Theorem

The partition obtained by the PT algorithm corresponds to a bisimulation equivalence on the transition system $(\Sigma, \rightarrow)$.

Splitting: replace each block B in a partition P with

$$B \cap pre(S) \text{ and } B \setminus pre(S),$$

Partition Refinement

Construct a partition of a state space Σ which is stable for a given transition relation $\rightarrow$, that is it does not need further refinement.

Theorem

The partition obtained by the PT algorithm corresponds to a bisimulation equivalence on the transition system $(\Sigma, \rightarrow)$.

Splitting: replace each block B in a partition P with

$$B \cap pre(S) \text{ and } B \setminus pre(S),$$

where for $X \subseteq \Sigma$, $pre(X) = \{s \in \Sigma \mid s \rightarrow x \text{ for some } x \in X\}$

Are T_1 and T_2 bisimilar?

In order to check whether two execution trees T_1 and T_2 in our PTS model are bisimilar, apply a similar refinement technique

```
1: procedure Lumping(T)
2:     P ← {S}
3:     while ¬Stable(P, T) do
4:         choose B ∈ P
5:         P ← Splitting(B, P)
6:     end while
7: end procedure
```

Are T_1 and T_2 bisimilar?

In order to check whether two execution trees T_1 and T_2 in our PTS model are bisimilar, apply a similar refinement technique

```
1: procedure Lumping(T)
2:     P ← {S}
3:     while ¬Stable(P, T) do
4:         choose B ∈ P
5:         P ← Splitting(B, P)
6:     end while
7: end procedure
```

(i) Apply PT to combined system $T_1 \oplus T_2$

Are T_1 and T_2 bisimilar?

In order to check whether two execution trees T_1 and T_2 in our PTS model are bisimilar, apply a similar refinement technique

1: **procedure** Lumping($T_1 \oplus T_2$)
2: $P \leftarrow \{S\}$
3: **while** $\neg$Stable(P, $T_1 \oplus T_2$) **do**
4: choose $B \in P$
5: $P \leftarrow$ Splitting(B, P)
6: **end while**
7: **end procedure**

(i) Apply PT to combined system $T_1 \oplus T_2$

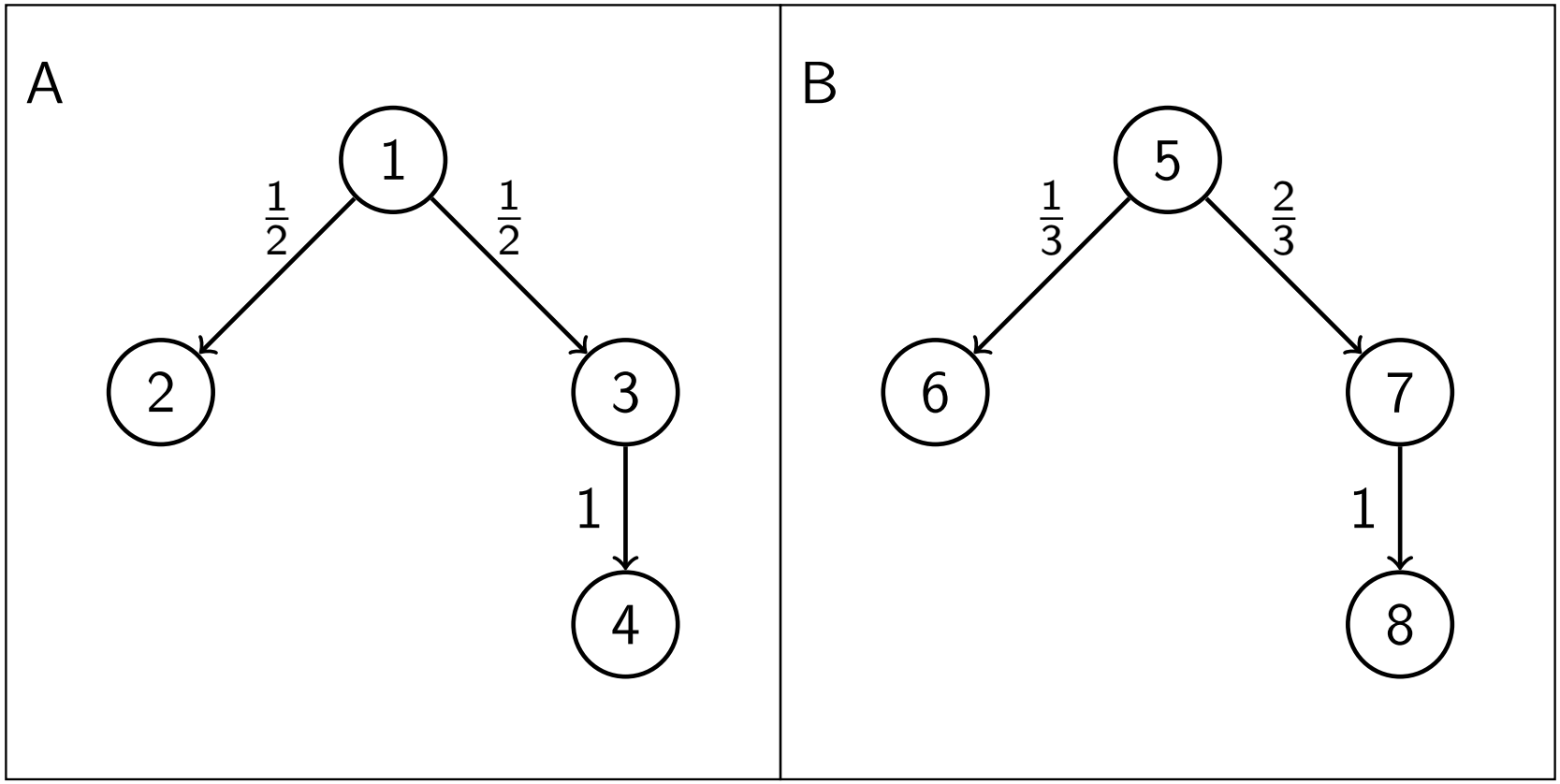
Are T_1 and T_2 bisimilar?

In order to check whether two execution trees T_1 and T_2 in our PTS model are bisimilar, apply a similar refinement technique

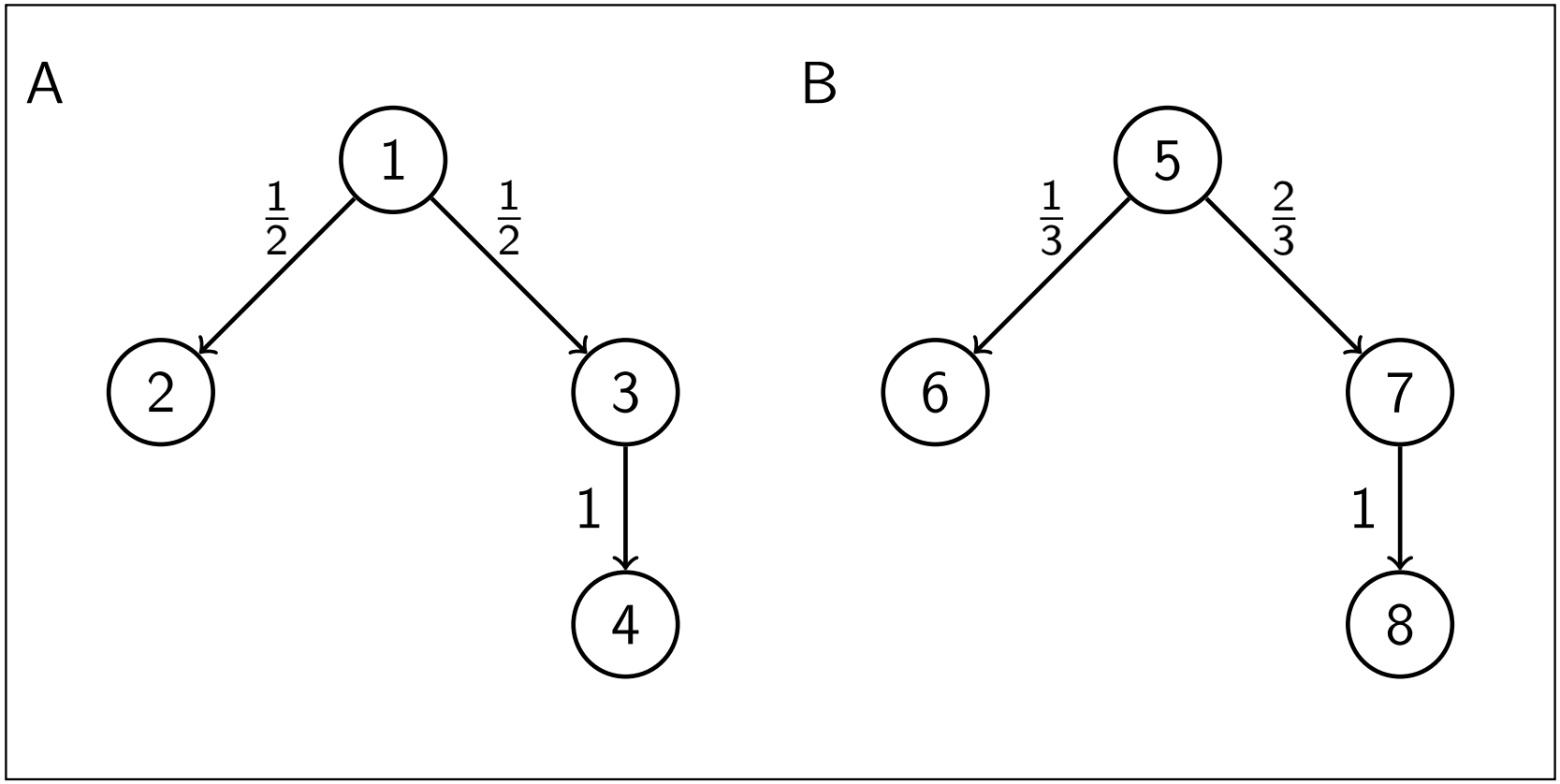
1: **procedure** QLumping(T_1, T_2)
2: $P \leftarrow \{S\}$
3: **while** $\neg$Stable(P, $T_1 \oplus T_2$) **do**
4: choose $B \in P$
5: $P \leftarrow$ Splitting(B, P)
6: CompDelta(L_1, L_2)
7: **end while**
8: **end procedure**

(i) Apply PT to combined system $T_1 \oplus T_2$
(ii) Check for corresponding classes matches

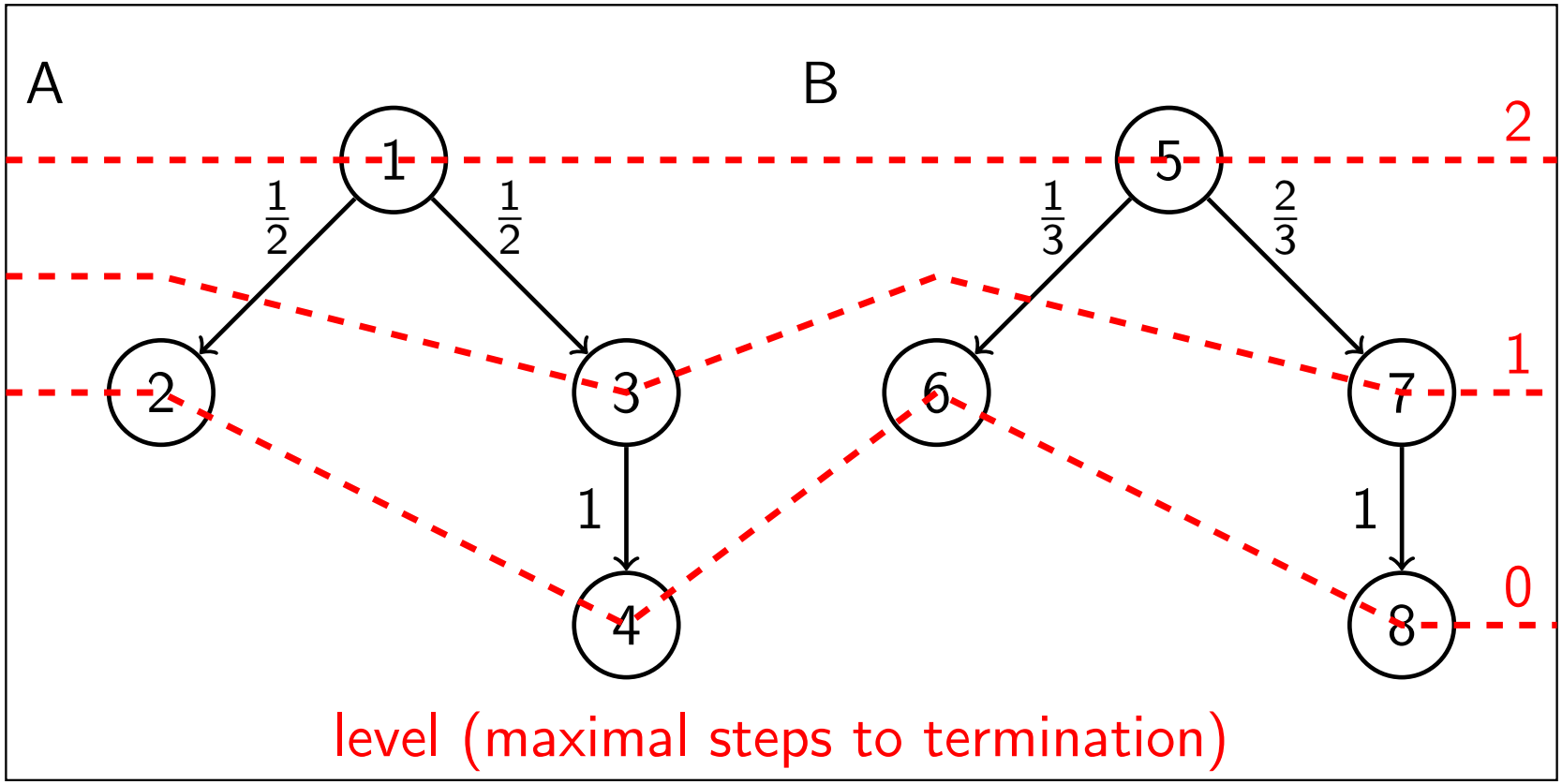
Lumping



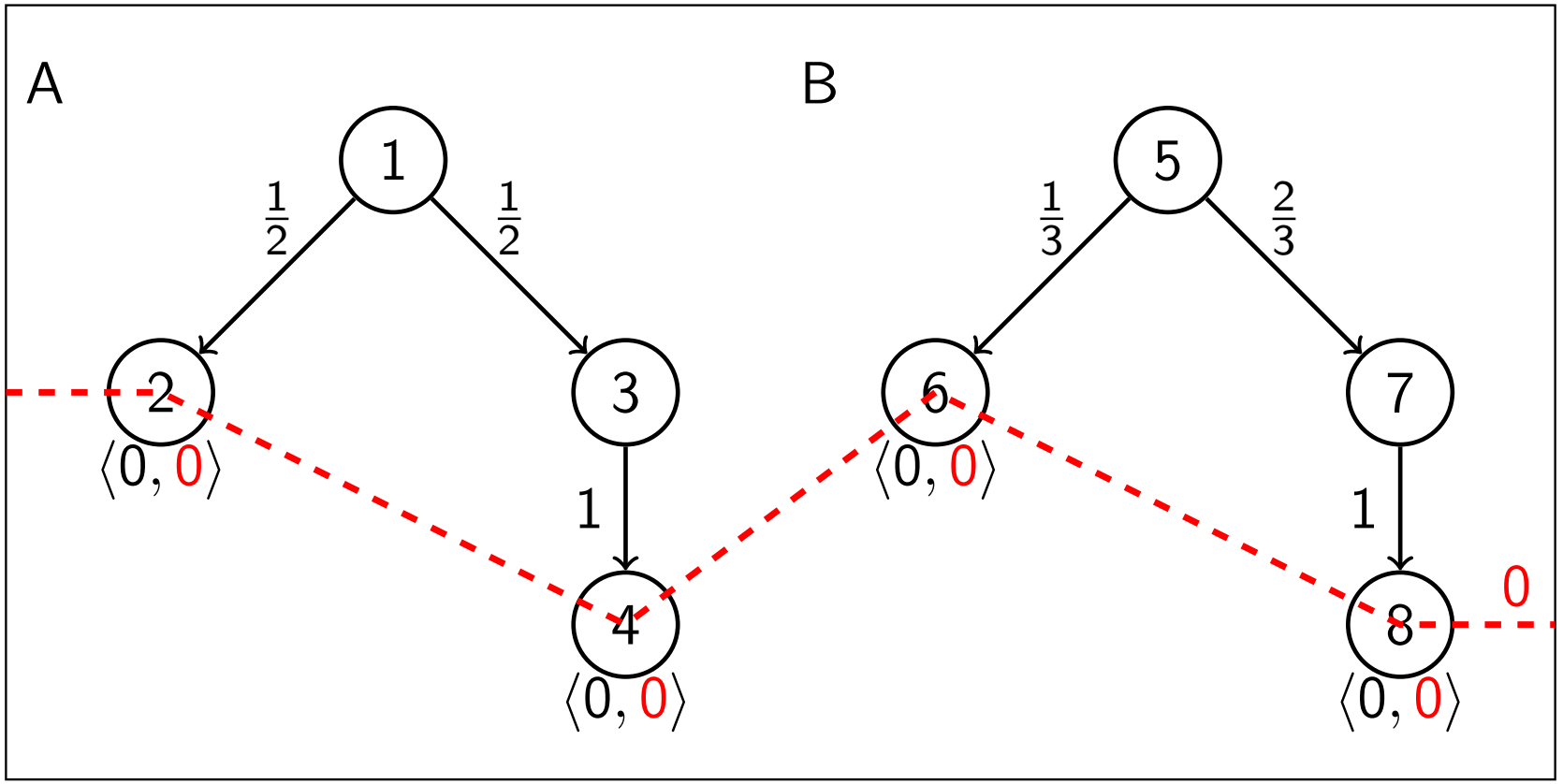
Lumping



Lumping

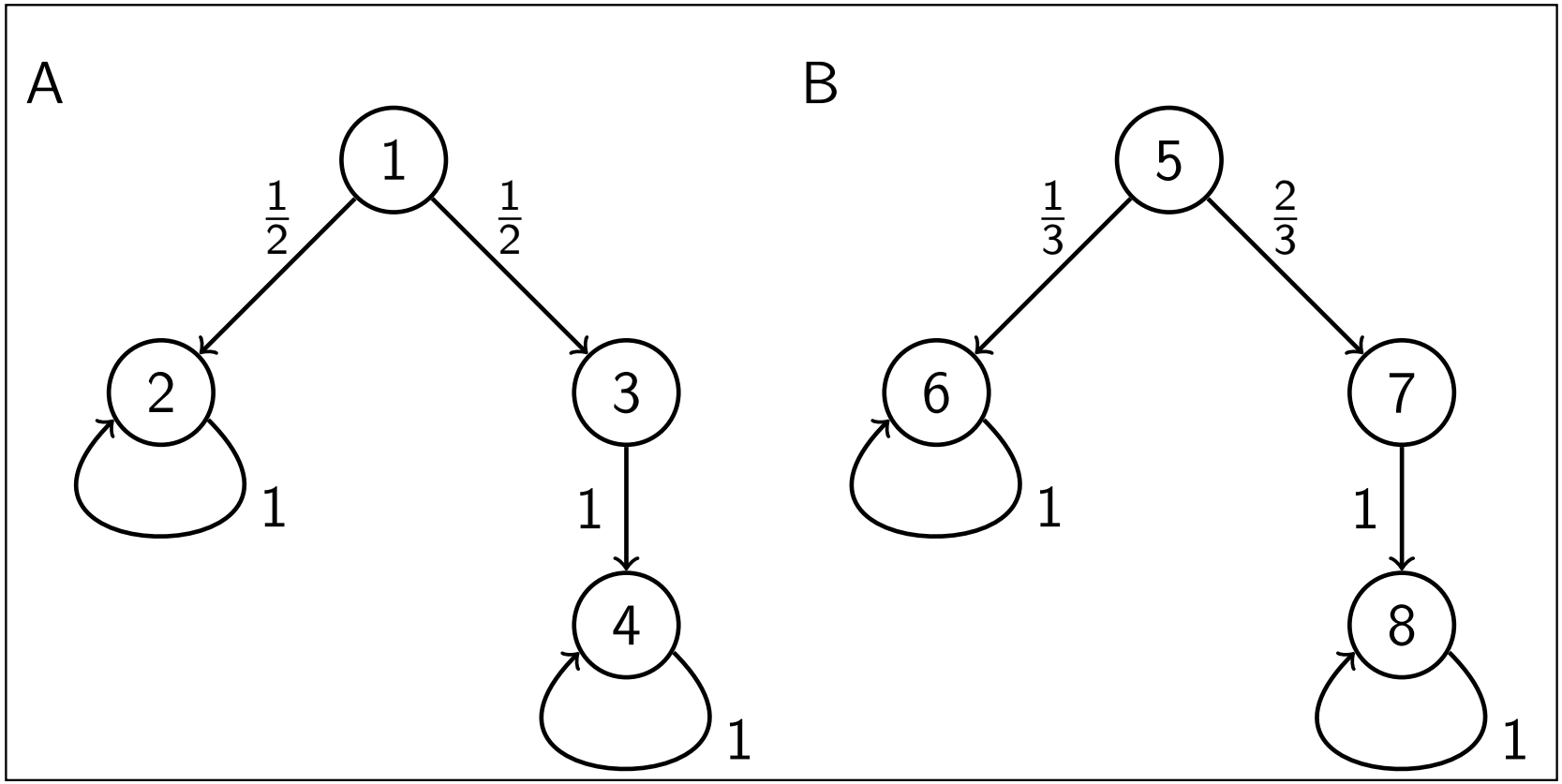

A
B
1
2
3
4
5
6
7
8
$\frac{1}{2}$
$\frac{1}{2}$
$\frac{1}{3}$
$\frac{2}{3}$
1
1
2
1
0
level (maximal steps to termination)

Lumping

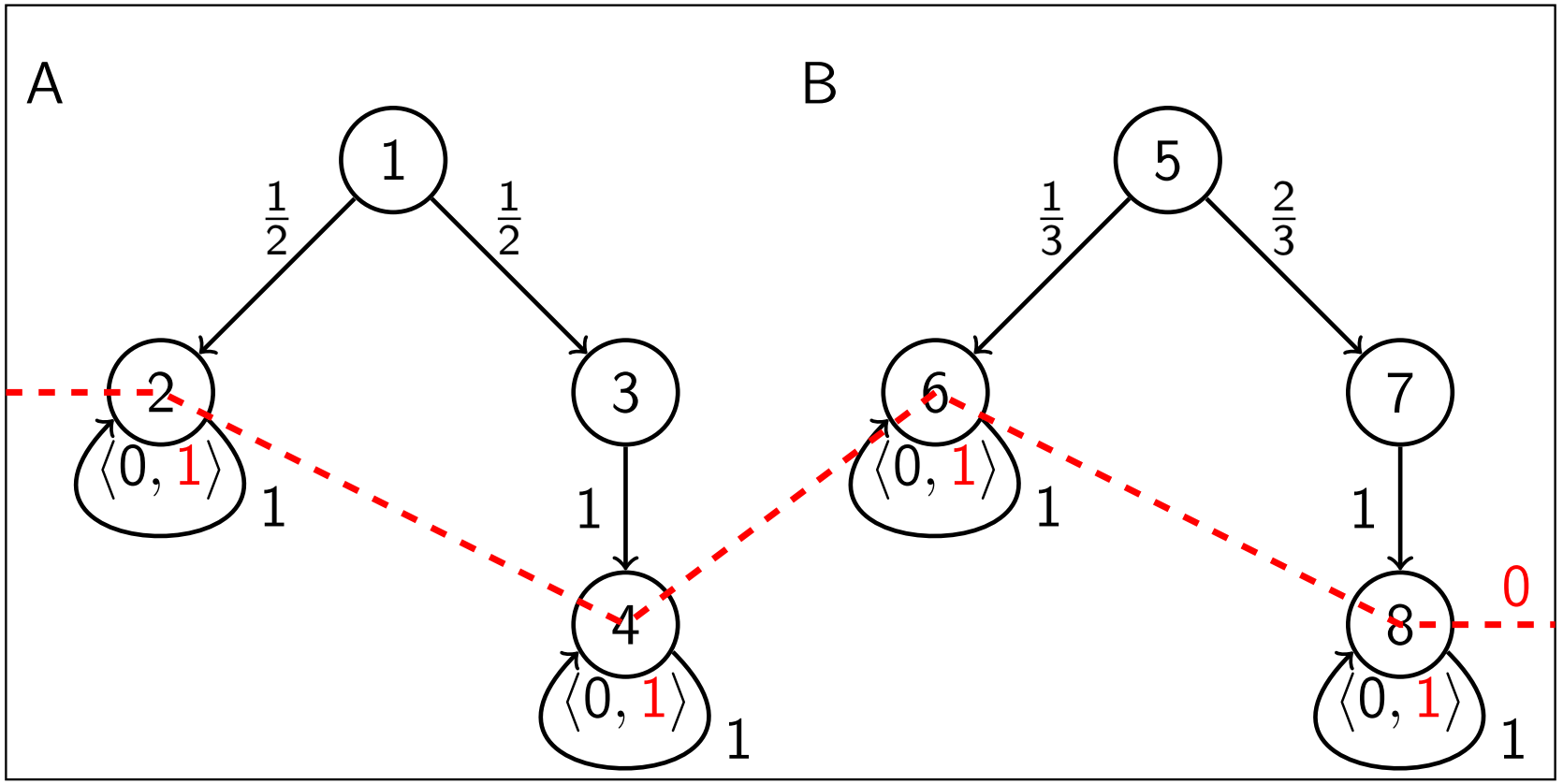


$P = \{\{2, 4, 6, 8\}\}$ $S = \{\{1, 3, 5, 7\}, \{2, 4, 6, 8\}\}$

Lumping

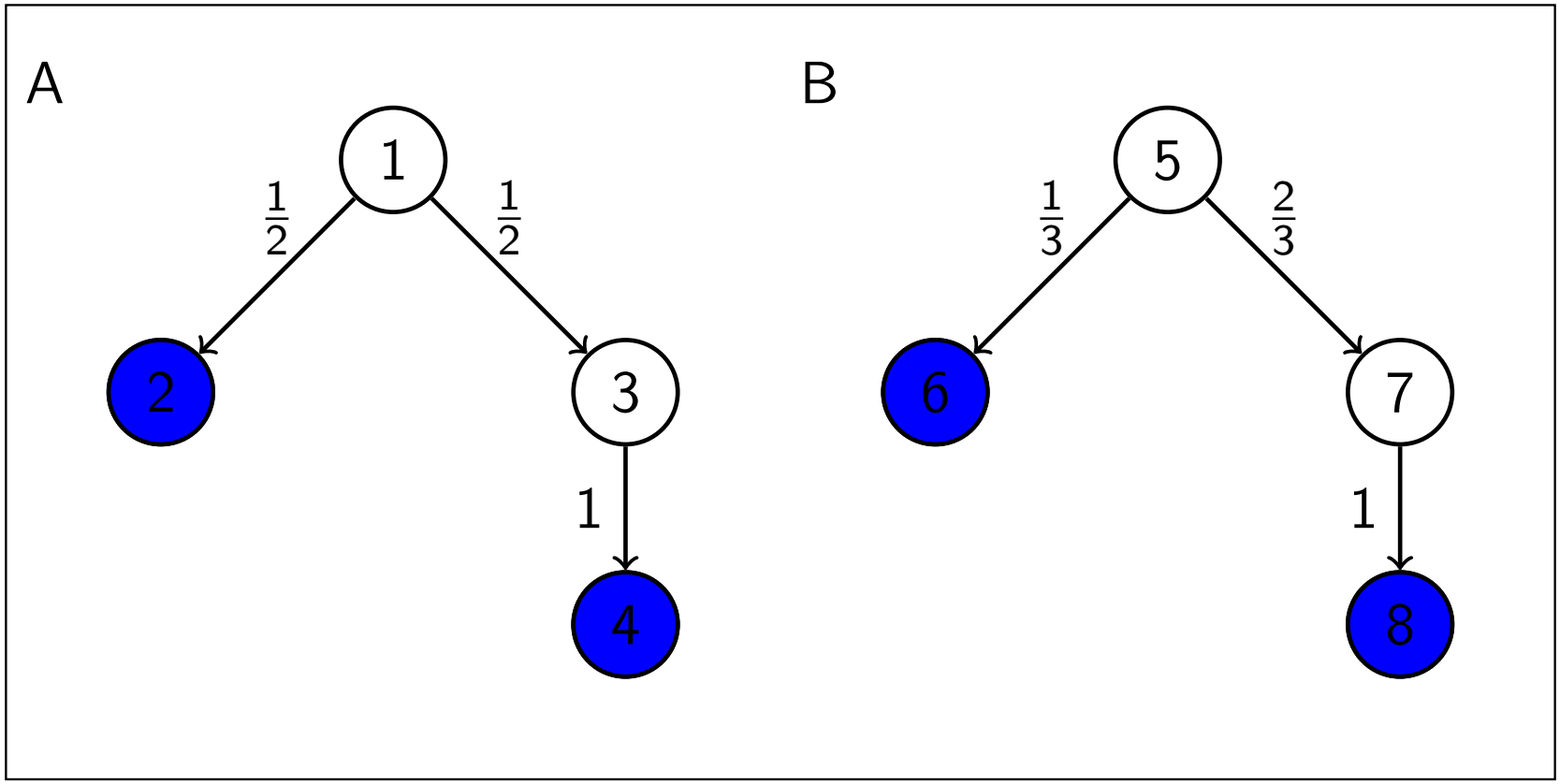


Lumping

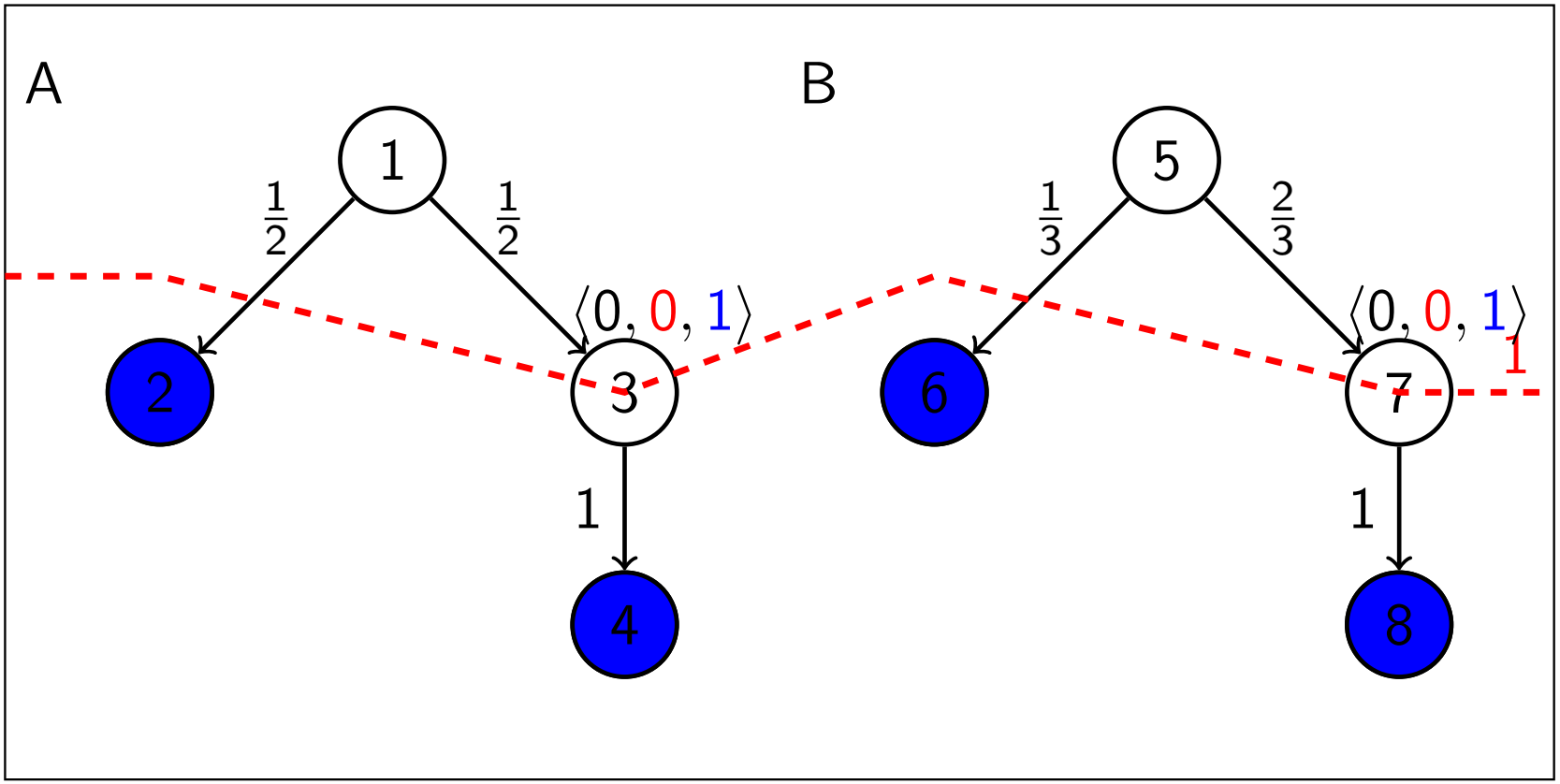


$P = \{\{2, 4, 6, 8\}\}$ $S = \{\{1, 3, 5, 7\}, \{2, 4, 6, 8\}\}$

Lumping

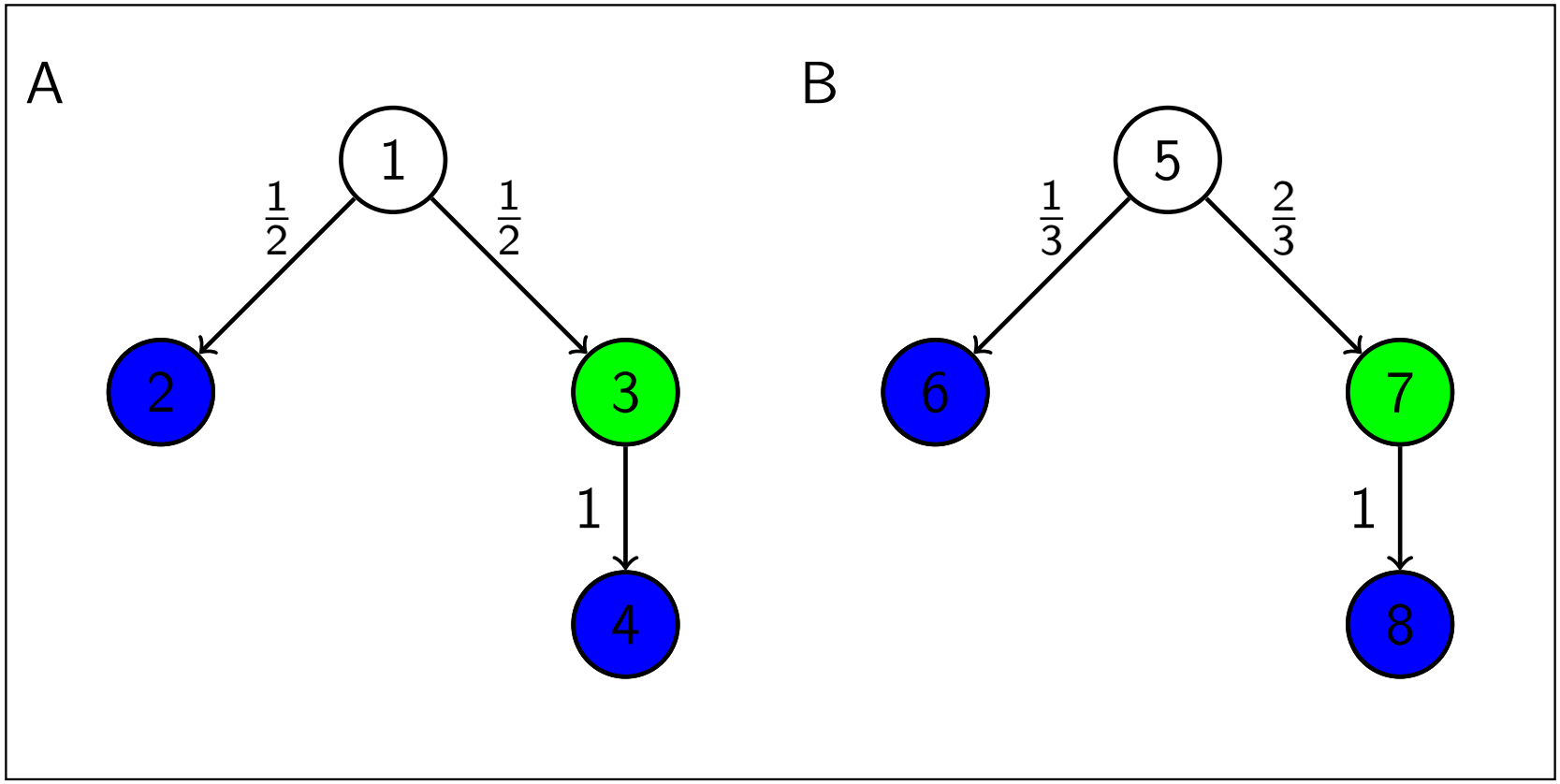


Lumping

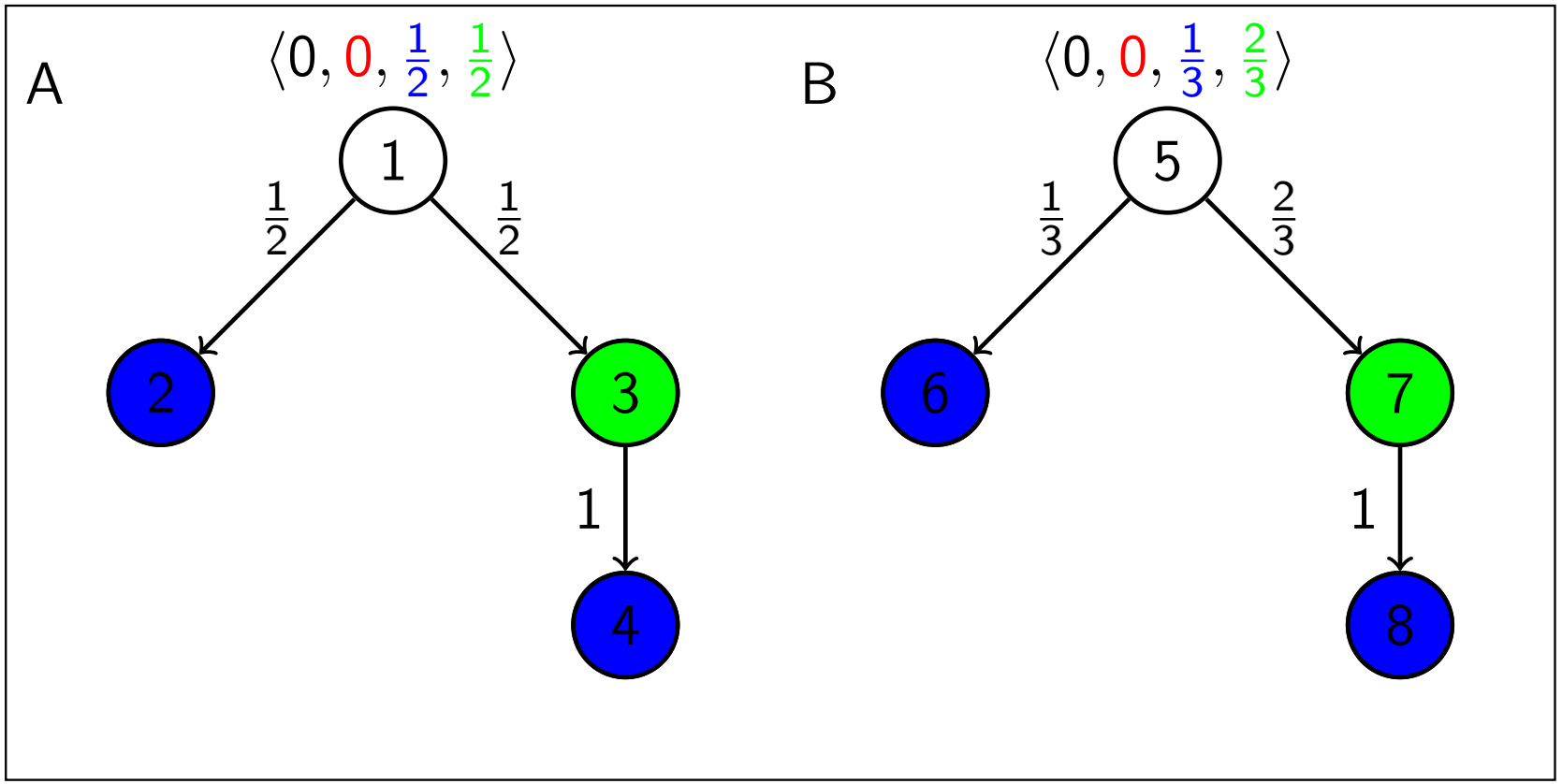


$$P = \{\{3,7\},\{2,4,6,8\}\} \qquad S = \{\{1,5\},\{3,7\},\{2,4,6,8\}\}$$

Lumping

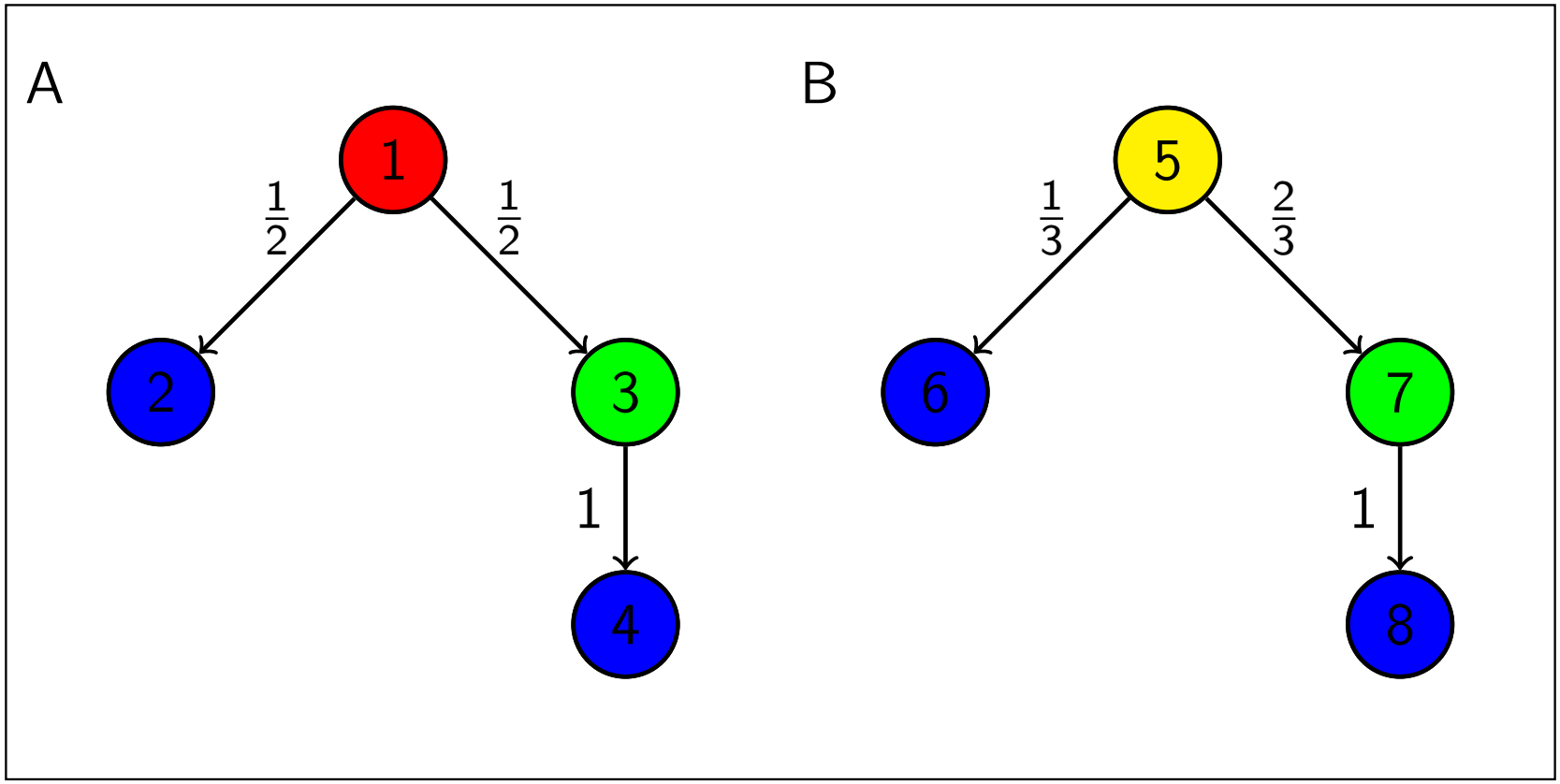


Lumping



$P = \{\{3, 7\}, \{2, 4, 6, 8\}\}$ $S = \{\{1, 5\}, \{3, 7\}, \{2, 4, 6, 8\}\}$

Lumping



Outline

Program Transformation

If two programs are not indistinguishable we make them so by transforming their internal behaviour.

Program Transformation

If two programs are not indistinguishable we make them so by transforming their internal behaviour.
In order to "fix" the bisimulation we introduce depleted actions/statement/transitions, e.g. statements which have no effect (e.g. tick) labels but just wait for time passing.

Program Transformation

If two programs are not indistinguishable we make them so by transforming their internal behaviour.
In order to "fix" the bisimulation we introduce depleted actions/statement/transitions, e.g. statements which have no effect (e.g. tick) labels but just wait for time passing.

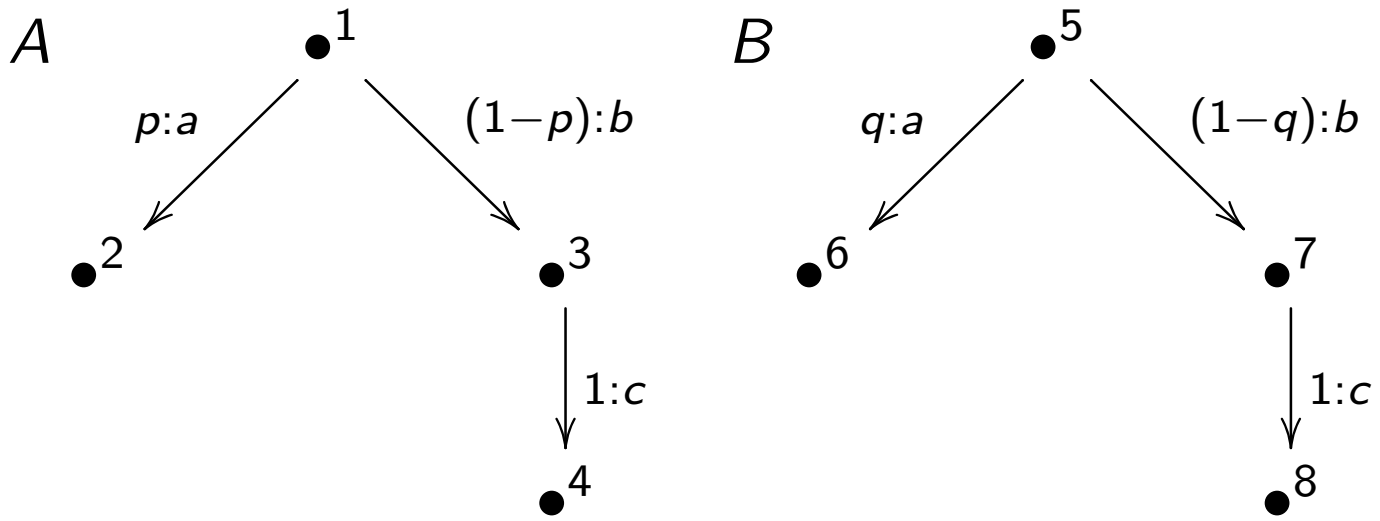
Concretly we

- introduce missing states, and
- redistribute distributions

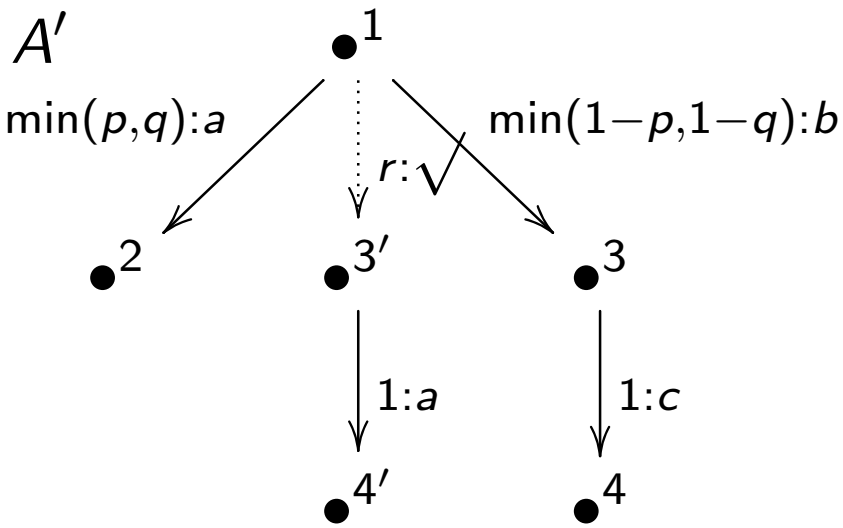
Transformation Algorithm

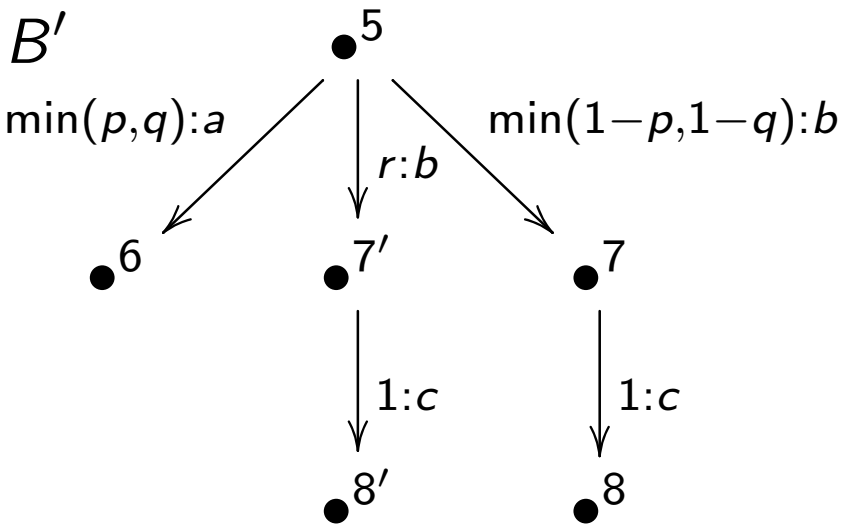
```
1:  procedure Padding(T1, T2)
2:      P ← {S}
3:      while ¬Stable(P, T1 ⊕ T2) do
4:          choose B ∈ P
5:          P ← Splitting(B, P)
6:          if ClassMismatch then
7:              Fixing(T1, T2)
8:          end if
9:      end while
10: end procedure
```

A simple Example



A simple Example



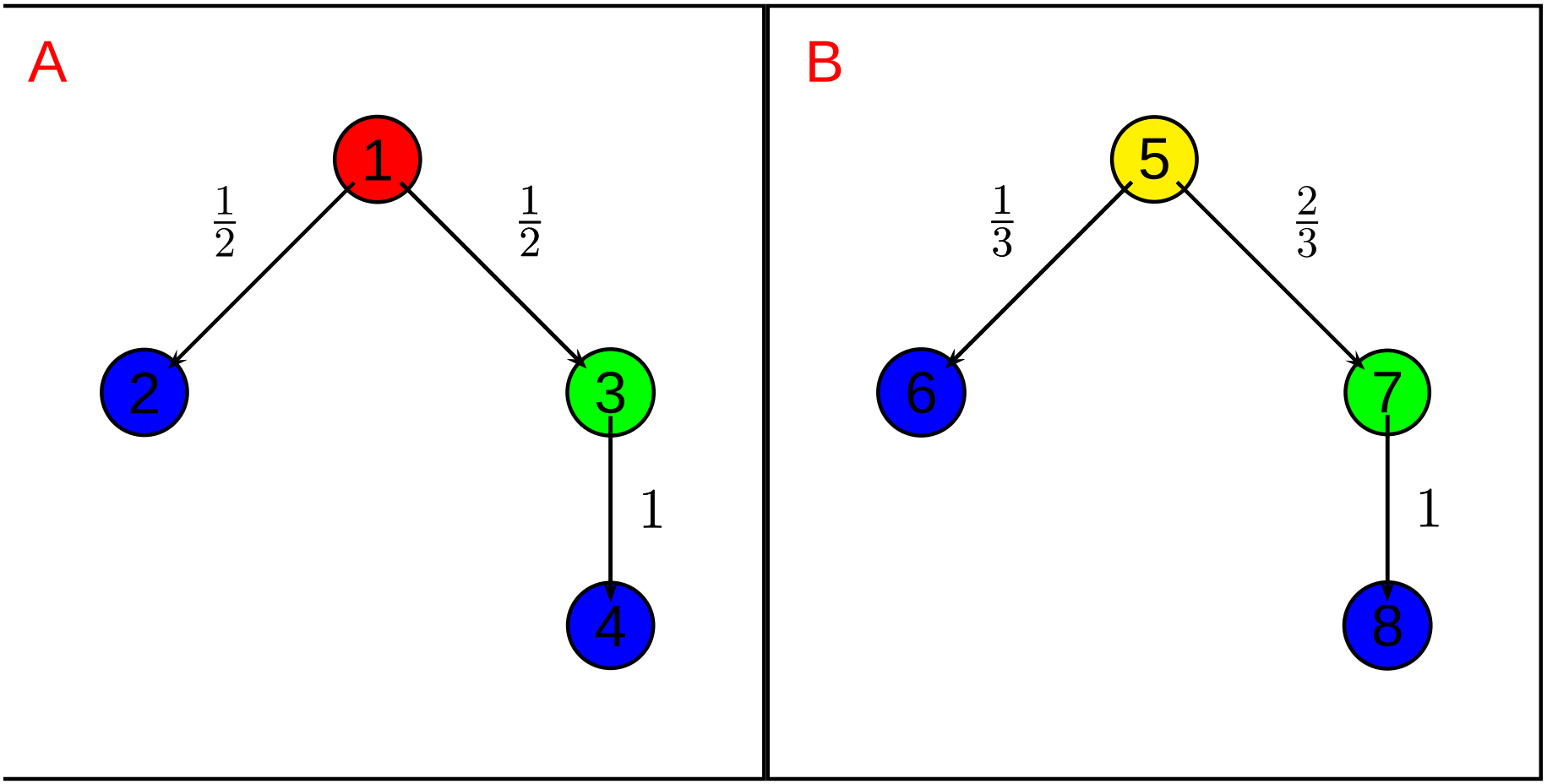


Outline

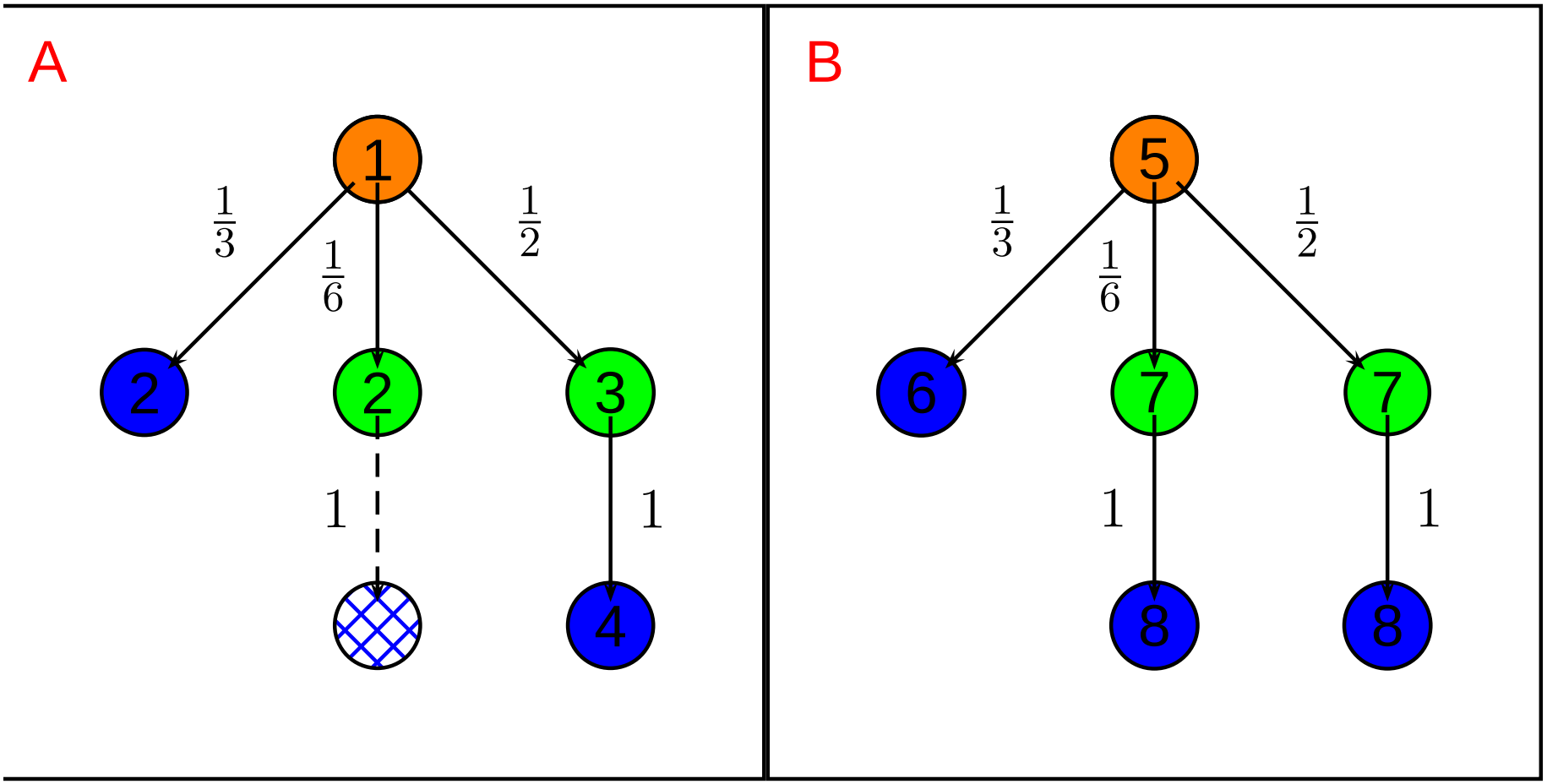
Probabilistic Fixing

```
i := 1;
while i<=3 do
  if k[i]==1 then
     s := s;
  else
     skip;
  fi;
  i := i+1;
od;
```

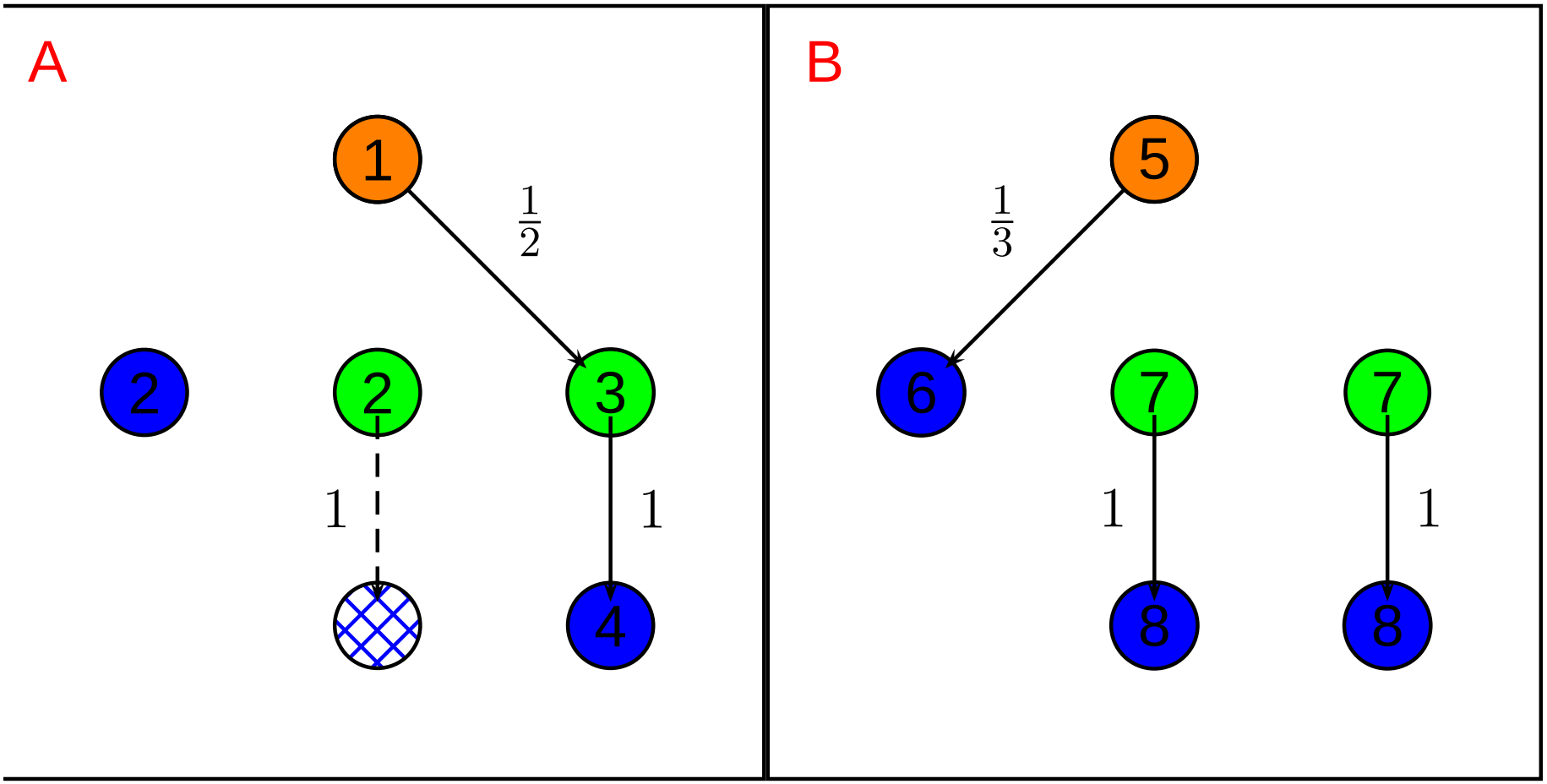
Probabilistic Padding



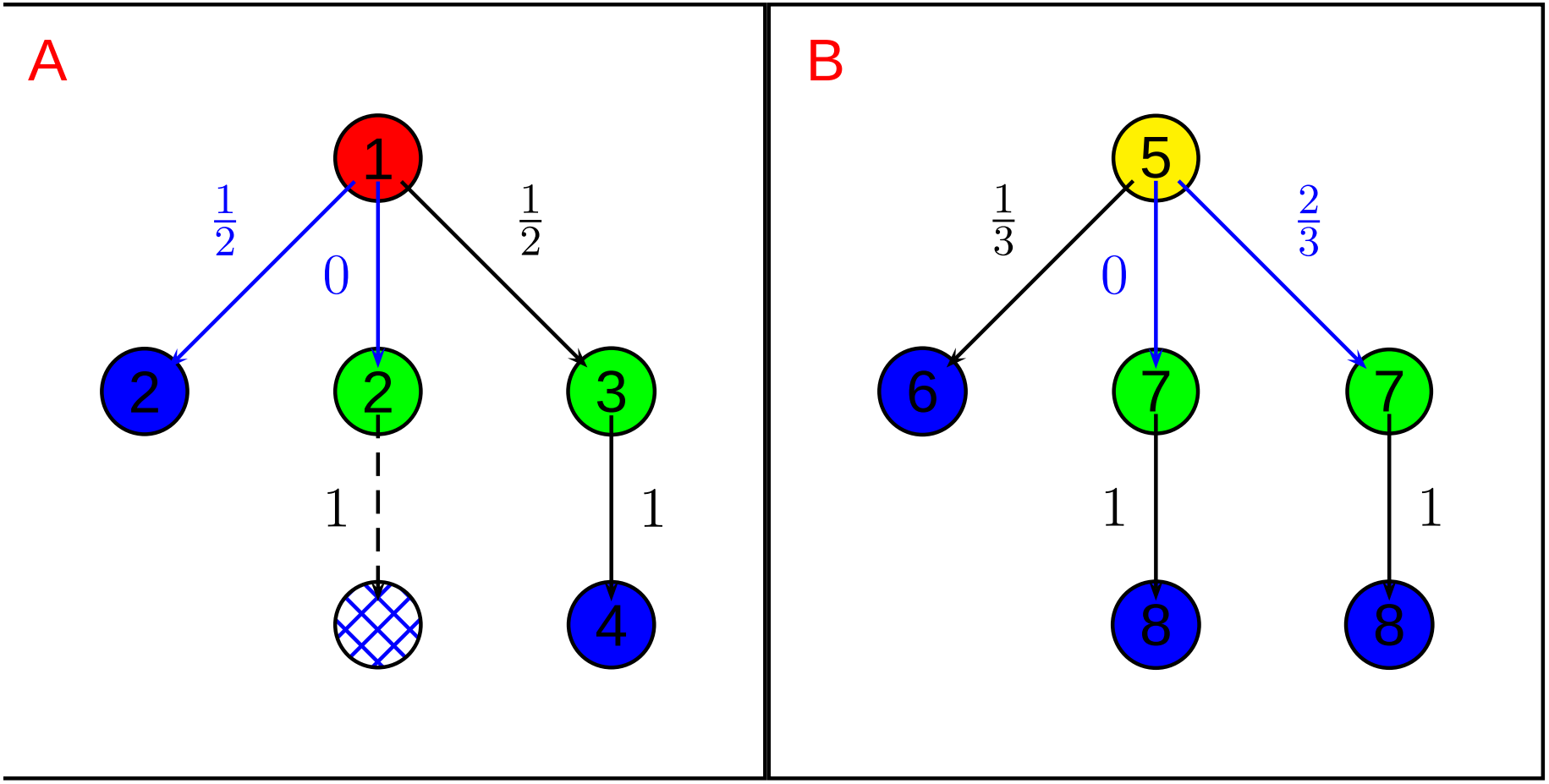
Probabilistic Padding



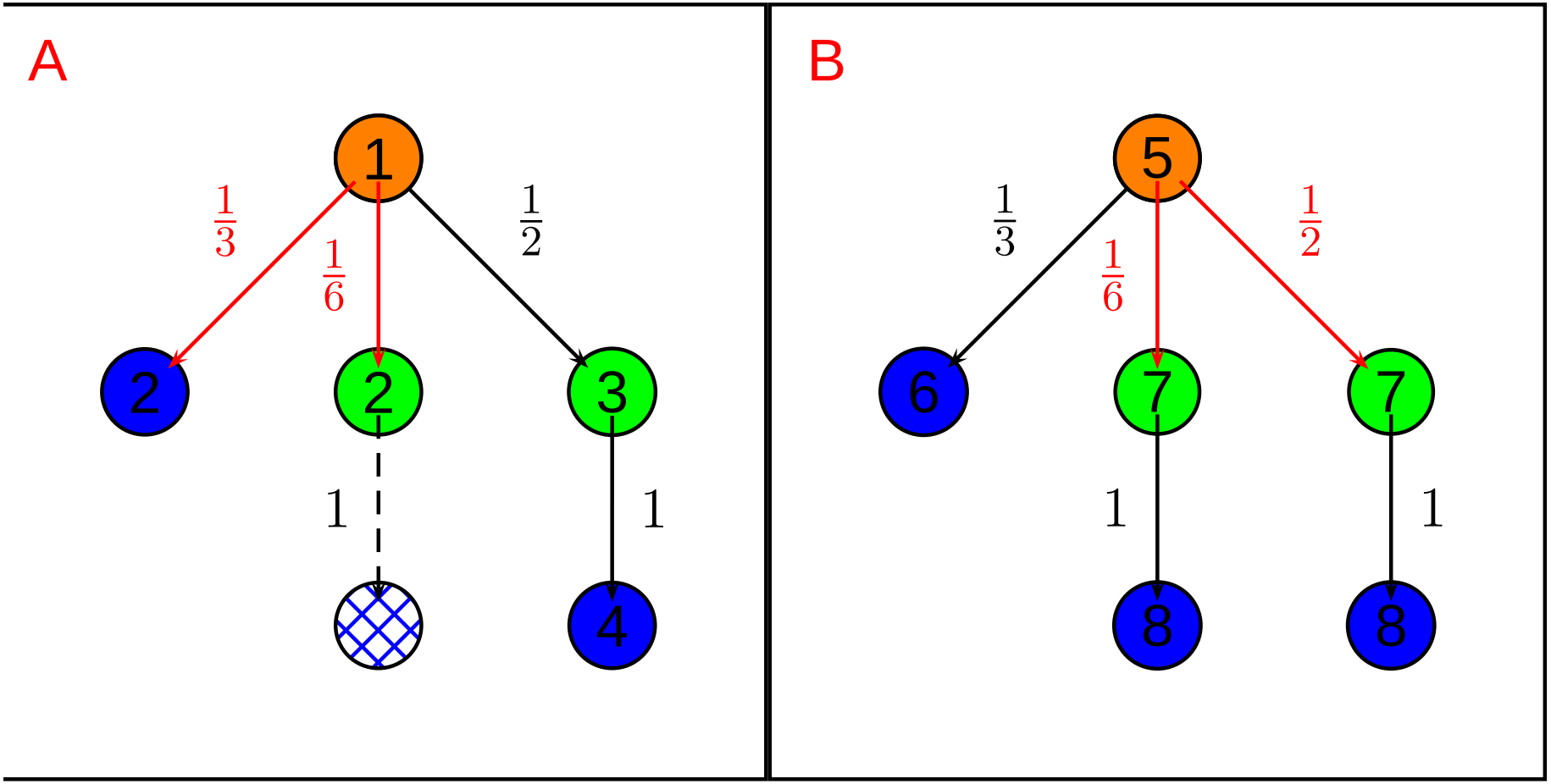
Probabilistic Padding



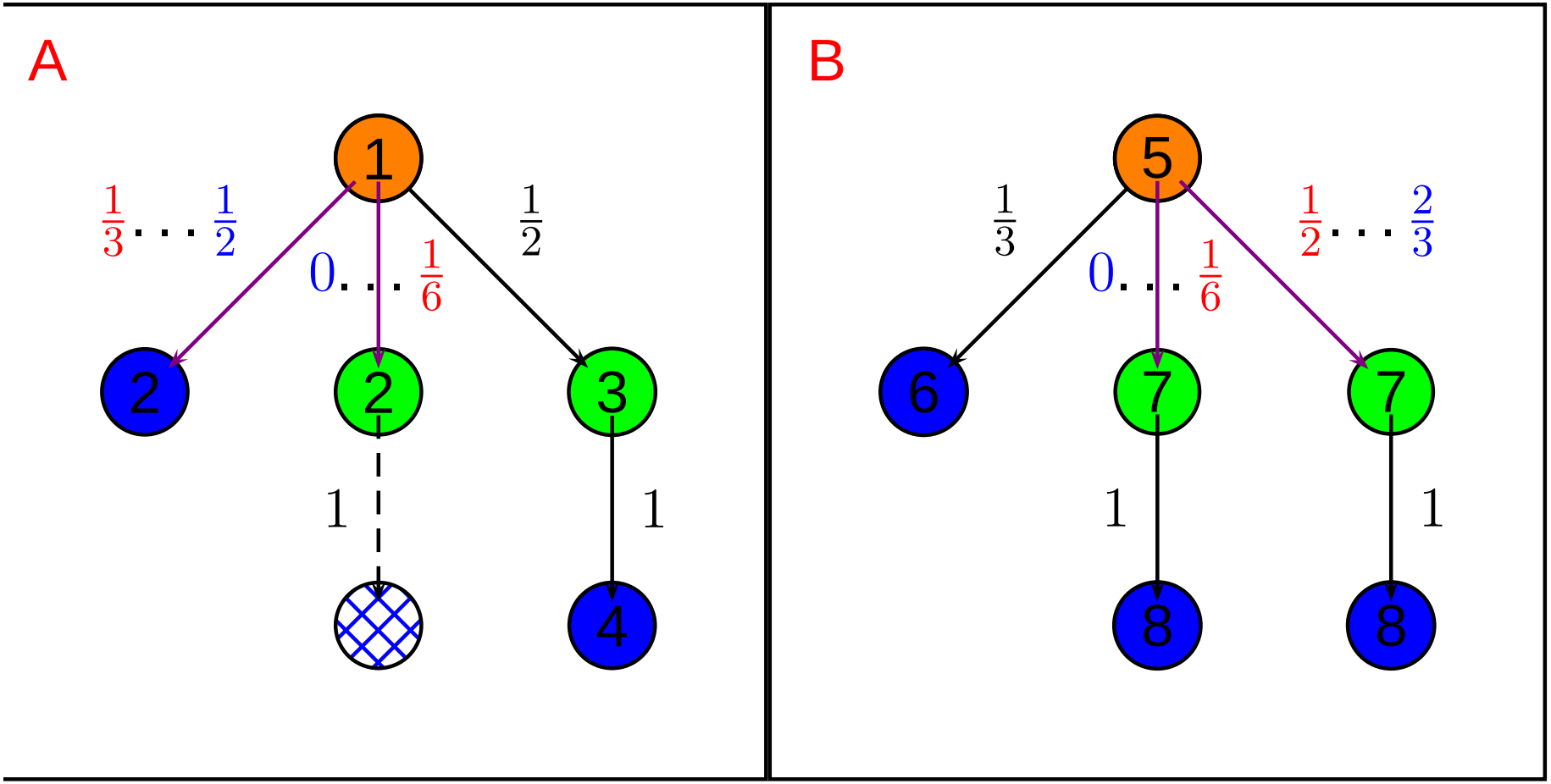
Probabilistic Padding



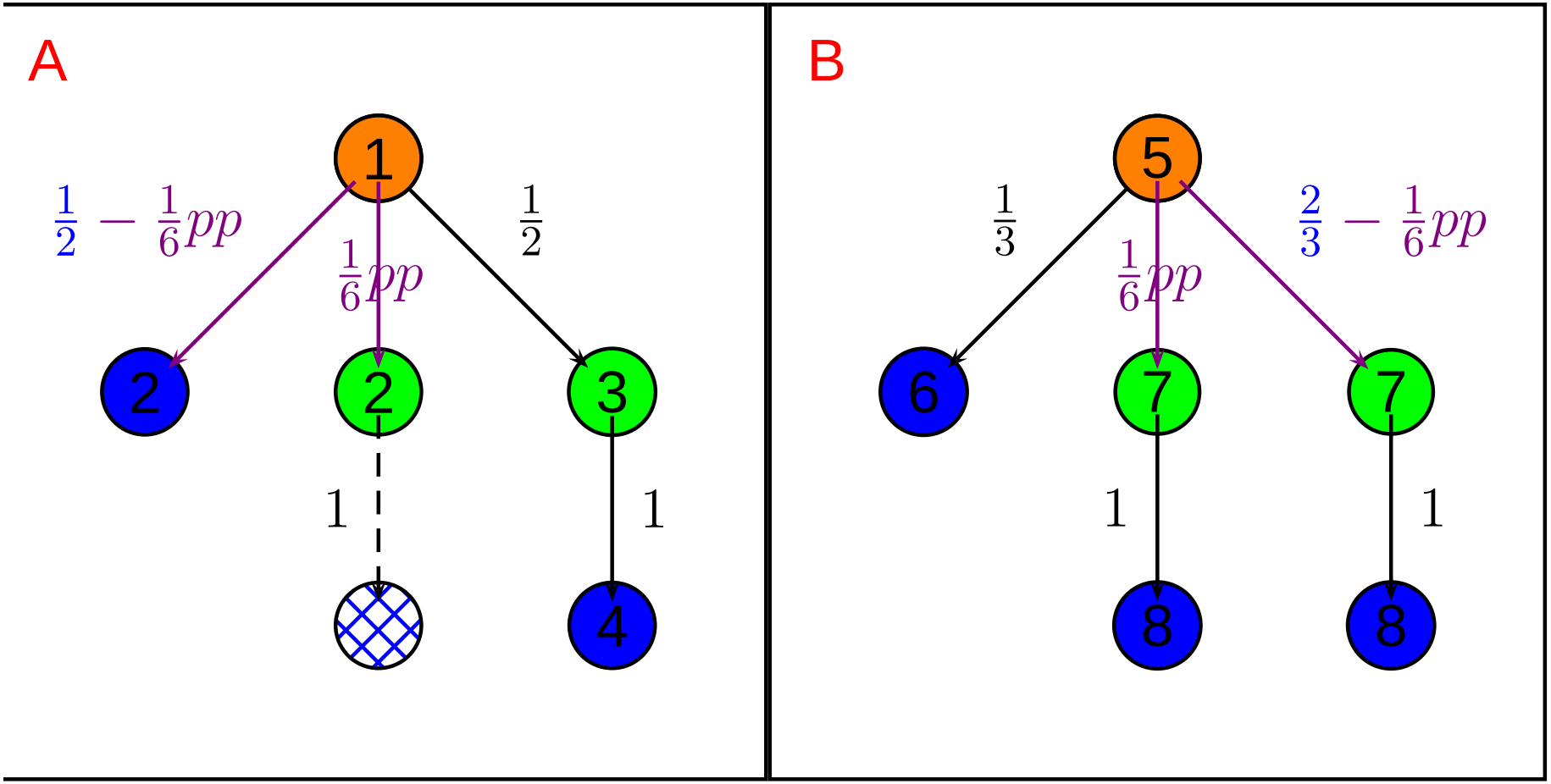
Probabilistic Padding



Probabilistic Padding



Probabilistic Padding



Probabilistic Fixing

```
i := 1;
while i<=3 do
  if k[i]==1 then
     s := s;
  else
     skip;
  fi;
  i := i+1;
od;
```

```
i := 1;
while i<=3 do
  if k[i]==1 then
     s := s; [skip]
  else
     [s := s]; skip
  fi;
  i := i+1;
od;
```

Probabilistic Fixing

```
i := 1;
while i<=3 do
  if k[i]==1 then
     choose p:  s := s; [skip]
     or 1-p:  s := s
     ro
  else
     choose p:  skip
     or 1-p:  [s := s]; skip
     ro
  fi;
  i := i+1;
od;
```

Computing Non-Bisimilarity

1: **procedure** QLumping(T_1, T_2)
2: $P \leftarrow \{S\}$
3: **while** $\neg$Stable(P, $T_1 \oplus T_2$) **do**
4: choose $B \in P$
5: $P \leftarrow$ Splitting(B, P)
6: CompDelta(L_1, L_2)
7: **end while**
8: **end procedure**

Computing Non-Bisimilarity

1: **procedure** QLumping(T_1, T_2)
2: $P \leftarrow \{S\}$
3: **while** $\neg$Stable(P, $T_1 \oplus T_2$) **do**
4: choose $B \in P$
5: $P \leftarrow$ Splitting(B, P)
6: **for** $s_1 \in L_1$ **do**
7: **for** $s_2 \in L_2$ **do**
8: $\beta \leftarrow \min(\beta, \|\chi_{s_1} - \chi_{s_2}\|_\infty)$
9: **end for**
10: $\delta \leftarrow \max(\delta, \beta)$
11: **end for**
12: **end while**
13: **end procedure**

Computing Non-Bisimilarity

1: **procedure** QLumping(T_1, T_2)
2: $P \leftarrow \{S\}$
3: **while** $\neg$Stable(P, $T_1 \oplus T_2$) **do**
4: choose $B \in P$
5: $P \leftarrow$ Splitting(B, P)
6: **for** $s_1 \in L_1$ **do**
7: **for** $s_2 \in L_2$ **do**
8: $\beta' \leftarrow \min(\beta', \|\omega\chi_{s_1} - \omega\chi_{s_2}\|_\infty)$
9: **end for**
10: $\delta' \leftarrow \max(\delta', \beta')$
11: **end for**
12: **end while**
13: **end procedure**

Outline

Security Tradeoffs

Clearly, padding leads to a performance penalty.

Security Tradeoffs

Clearly, padding leads to a performance penalty.
The probabilistic parameter of our transformation can be utilised for trading security for performance.

Security Tradeoffs

Clearly, padding leads to a performance penalty.
The probabilistic parameter of our transformation can be utilised for trading security for performance.
Our central theses are:

Security Tradeoffs

Clearly, padding leads to a performance penalty.
The probabilistic parameter of our transformation can be utilised for trading security for performance.
Our central theses are:

- it can make sense to not fix a time leak completely but instead – given the additional security costs – to settle for a partial solution;

Security Tradeoffs

Clearly, padding leads to a performance penalty.
The probabilistic parameter of our transformation can be utilised for trading security for performance.
Our central theses are:

- it can make sense to not fix a time leak completely but instead – given the additional security costs – to settle for a partial solution;
- the analysis of such a trade-off situation (security vs. costs) can be done in a formal way.

An Example

```
i := 1;
while i<=3
do
 if k[i]==1
 then
   s := s;
 else
   skip;
 fi;
 i := i+1;
od;
```

```
i := 1;
while i<=3
do
 if k[i]==1
 then
   choose p: s := s; skip
   or     q: s := s
   ro
 else
   choose p: s := s; skip
   or     q: skip
   ro
 fi;
 i := i+1;
od;
```

```
i := 1;
while i<=3
do
 if k[i]==1
 then
   s := s; skip
 else
   s := s; skip
 fi;
 i := i+1;
od;
```

Assumptions

In our concrete experiments we used the following assumptions:

- $\mathtt{i} \in \{1, .., 3\}$

Assumptions

In our concrete experiments we used the following assumptions:

- $\texttt{i} \in \{1, .., 3\}$
- $\texttt{k}$ is a three element array with values in $\{0, 1\}$

Assumptions

In our concrete experiments we used the following assumptions:

- $\mathtt{i} \in \{1, .., 3\}$
- $\mathtt{k}$ is a three element array with values in $\{0, 1\}$
- $\mathtt{k}$, representing a *secret key*, and $\mathtt{s}$ have security typing H,

Assumptions

In our concrete experiments we used the following assumptions:

- $\mathtt{i} \in \{1, .., 3\}$
- $\mathtt{k}$ is a three element array with values in $\{0, 1\}$
- $\mathtt{k}$, representing a *secret key*, and $\mathtt{s}$ have security typing H,
- $\mathtt{i}$ is the only low variable which can be observed by an attacker.

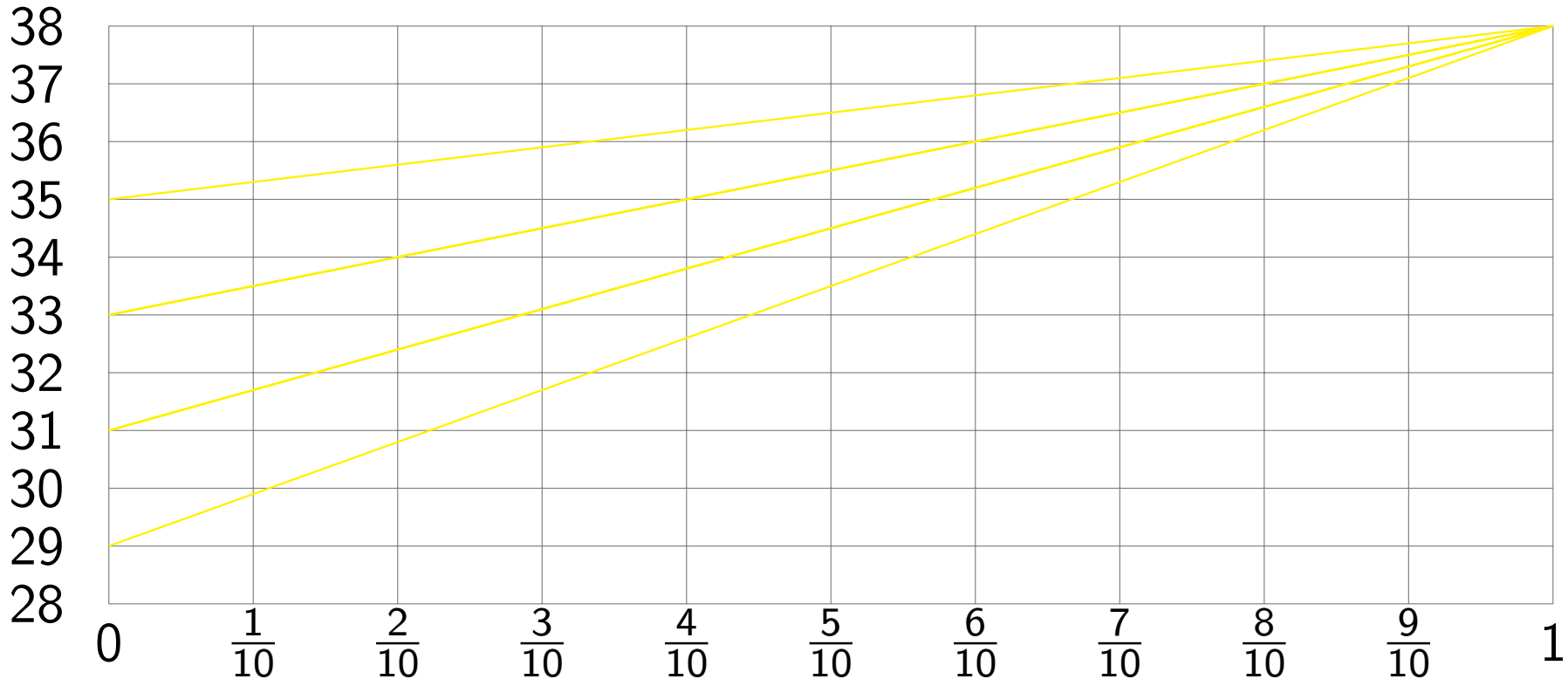
Assumptions

In our concrete experiments we used the following assumptions:

- $\mathtt{i} \in \{1, .., 3\}$
- $\mathtt{k}$ is a three element array with values in $\{0, 1\}$
- $\mathtt{k}$, representing a *secret key*, and $\mathtt{s}$ have security typing H,
- $\mathtt{i}$ is the only low variable which can be observed by an attacker.

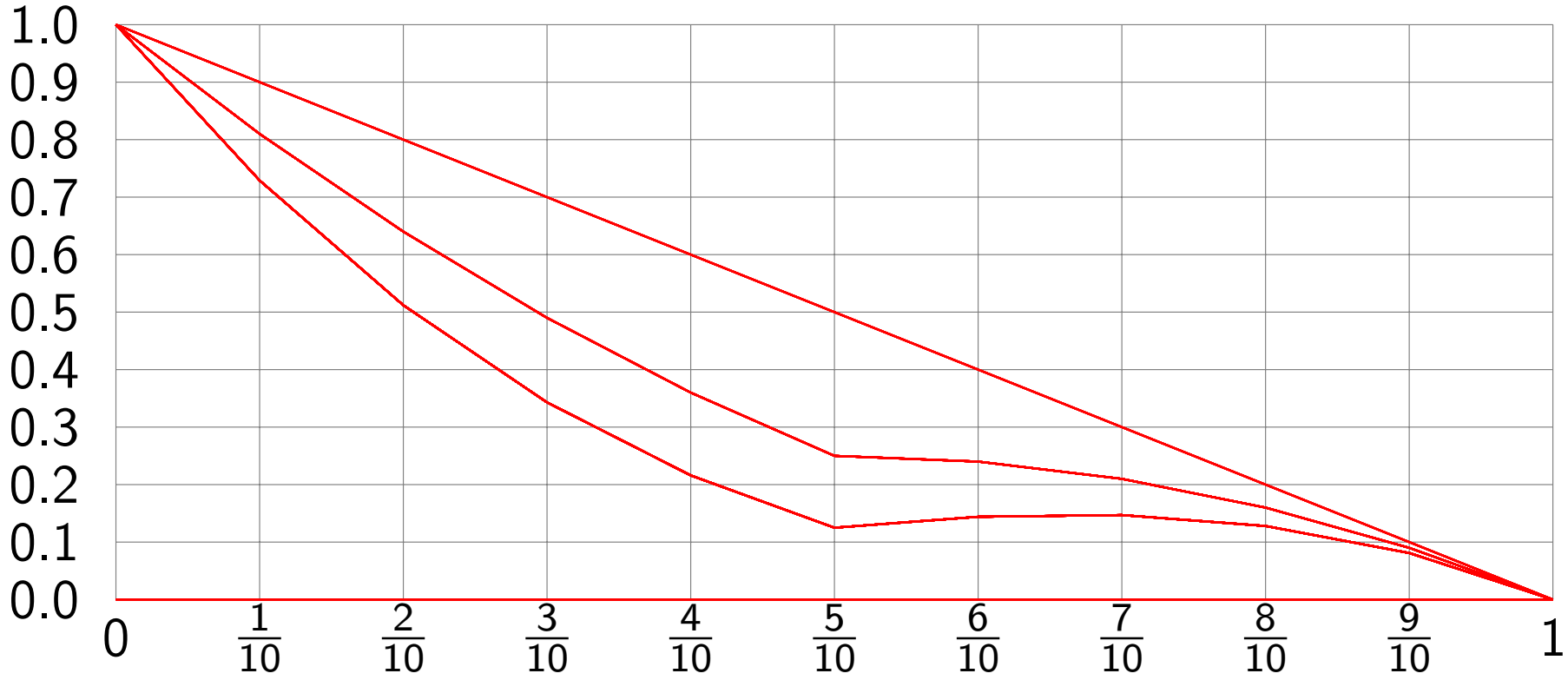
We implemented this example using execution times: $t_{asn} = 3$, $t_{br} = 2$, $t_{skip} = 1$ and $t_{ch} = 0$.

Concrete Experiments



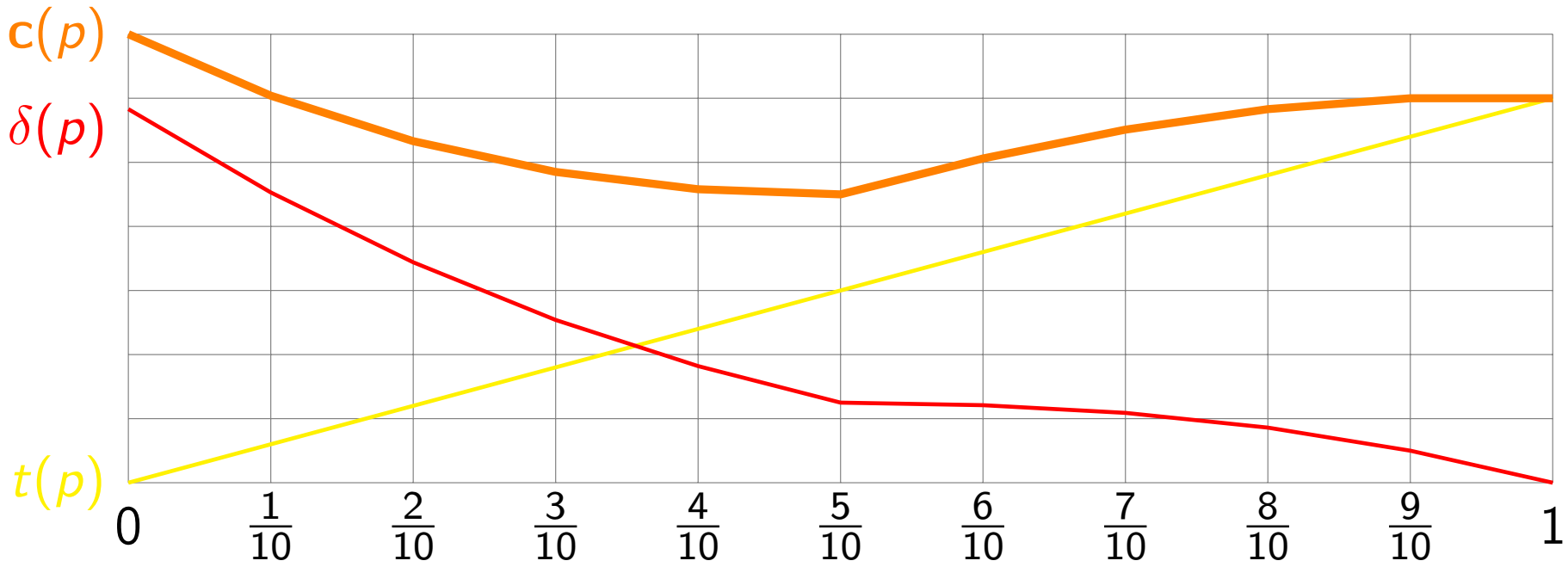
Average Running Time $t(p)$ for $k = 000, k = 100, k = 110, k = 111$.

Concrete Experiments



Security measured via $\varepsilon(p)$

Concrete Experiments



Cost Function: $c(p) = 6\varepsilon(p) + t(p)$