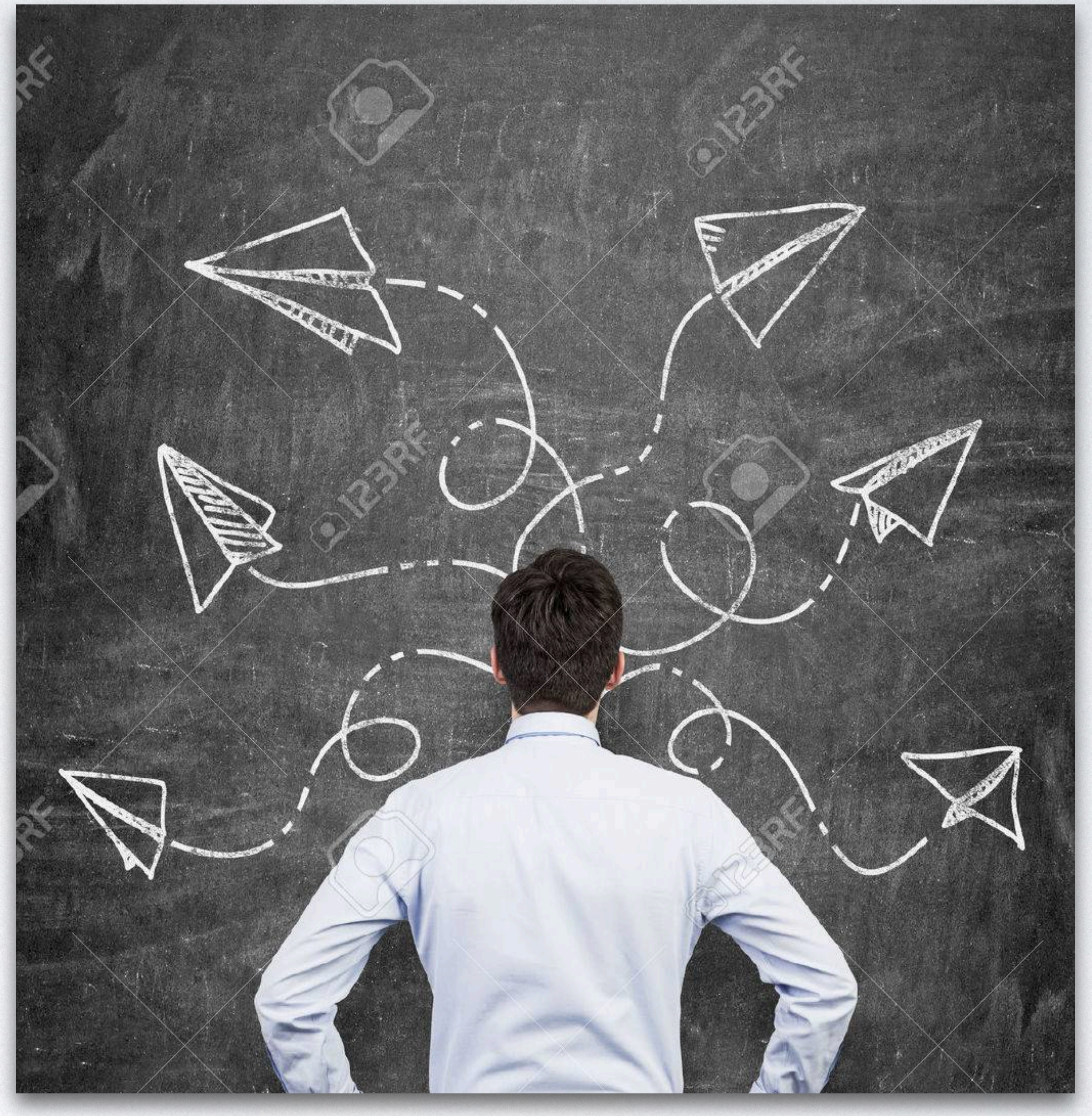
Semantics of Probabilistic Programs

An Algebraic Approach

Di Wang
Carnegie Mellon University

Probabilistic Programs



Draw random data from distributions

Condition control-flow at random

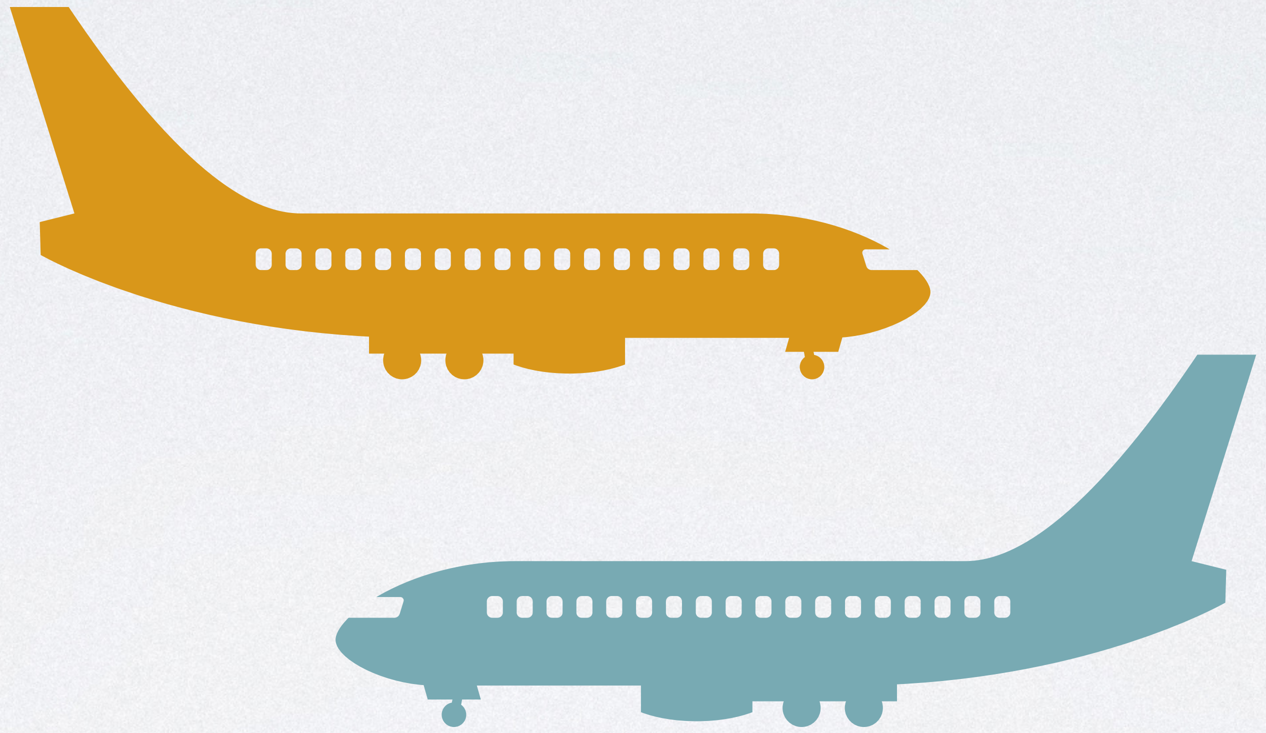
Probabilistic Programs

- True randomness
- Distributions on executions

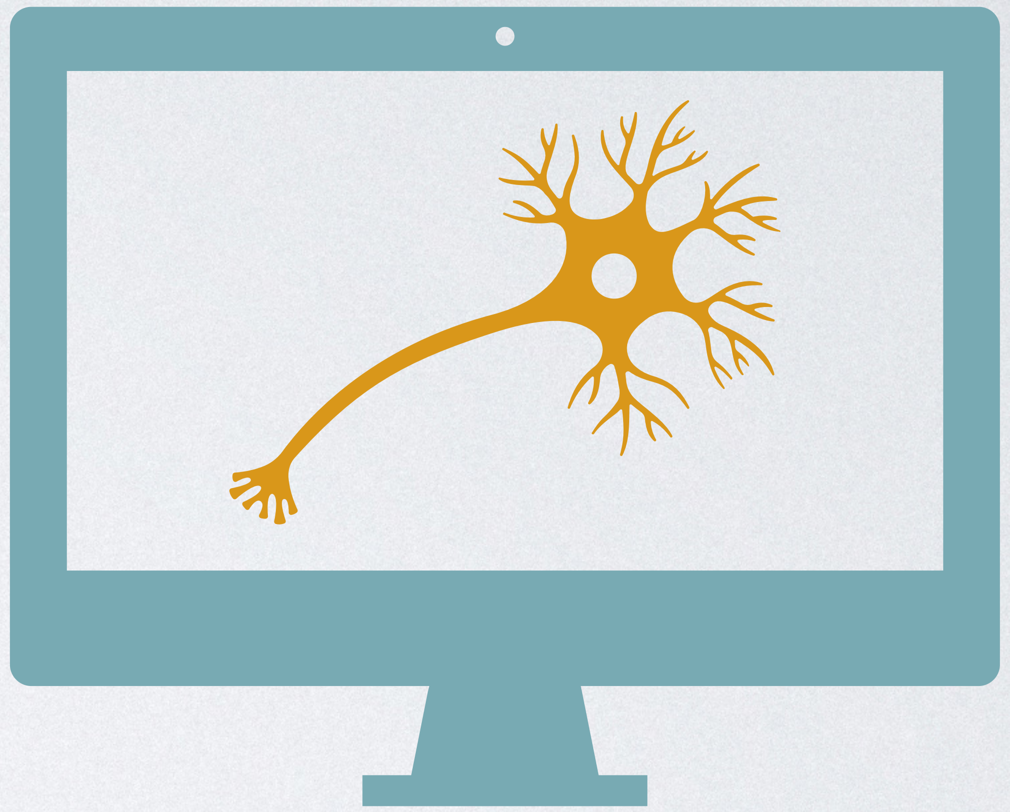
```
b1 ~ Bernoulli(0.5);
b2 ~ Bernoulli(0.7);
while (b1 && b2) do
  if prob(0.6) then
    b1 ~ Bernoulli(0.5)
  else
    b2 ~ Bernoulli(0.7)
  fi;
  tick(1.0)
od;
return (b1, b2)
```

Applications of Prob. Prog.

Randomized Algorithms
(improve efficiency)



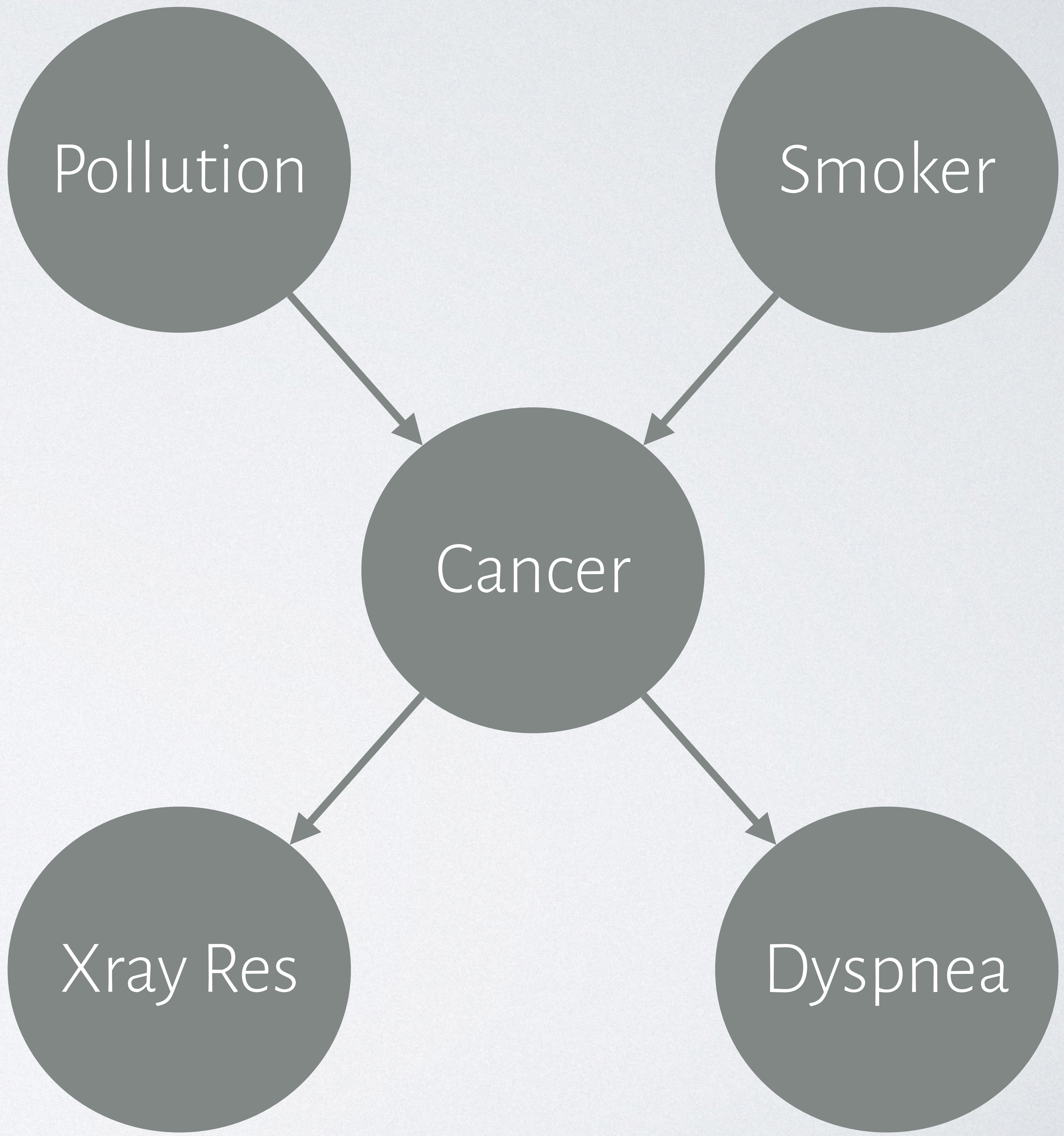
Cyber-Physical Systems
(model uncertainty)



Machine Learning Algorithms
(describe statistical models)

Bayesian Networks

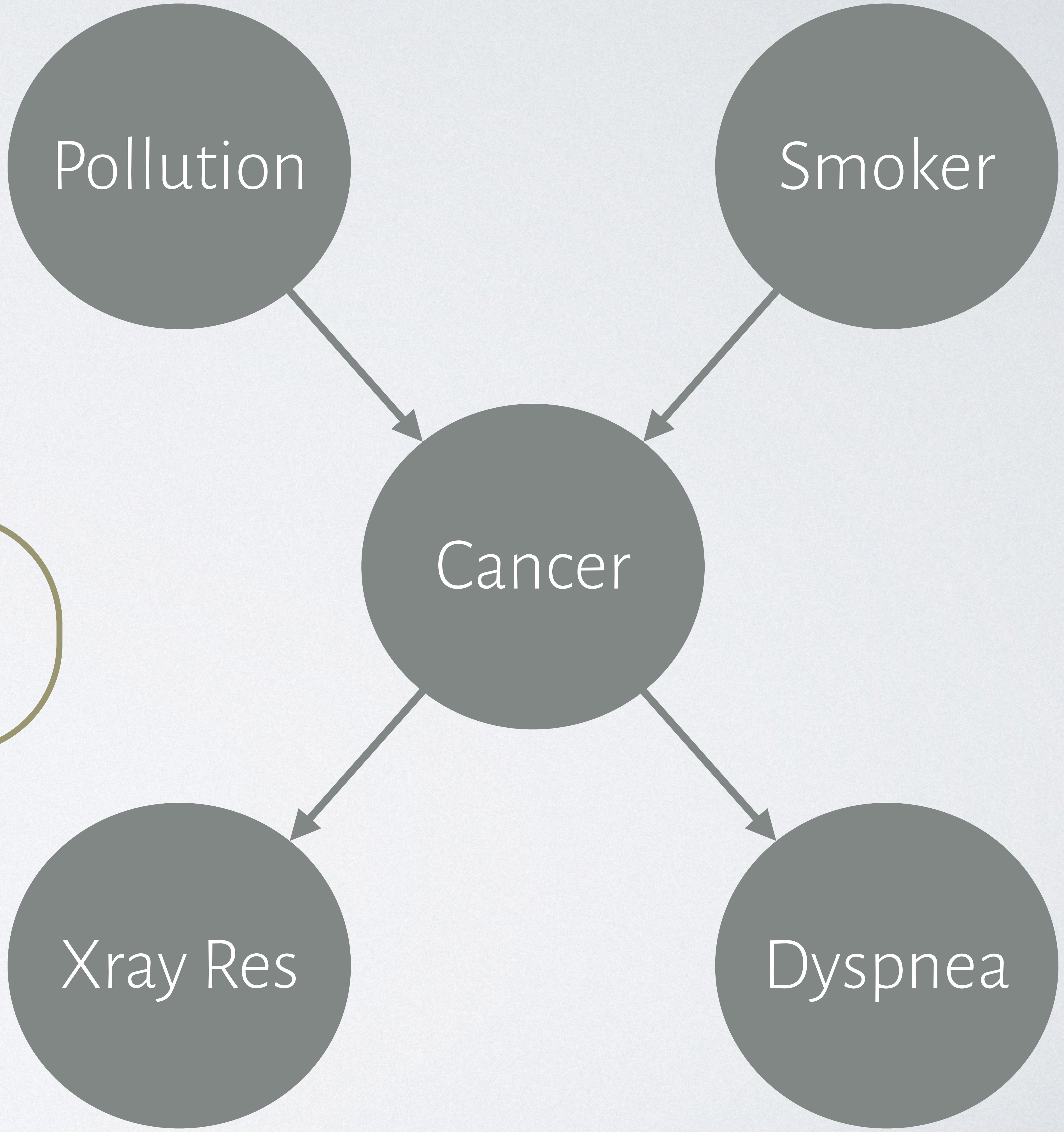
- Conditional distributions
- Query about the posterior



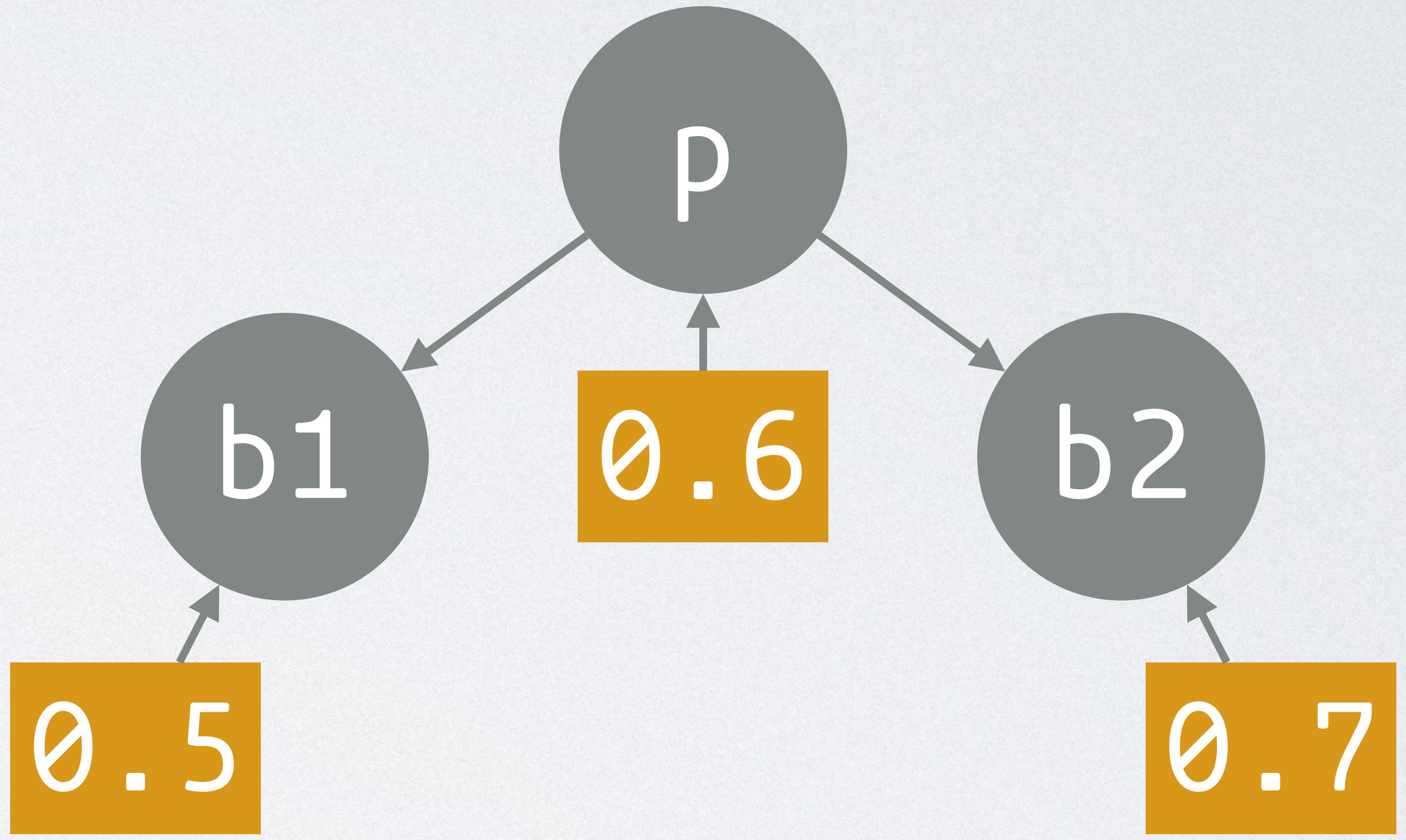
Bayesian Networks

- Conditional distributions
- Query about the posterior

Prob[Cancer | Smoker ∧ Xray Res] = ?

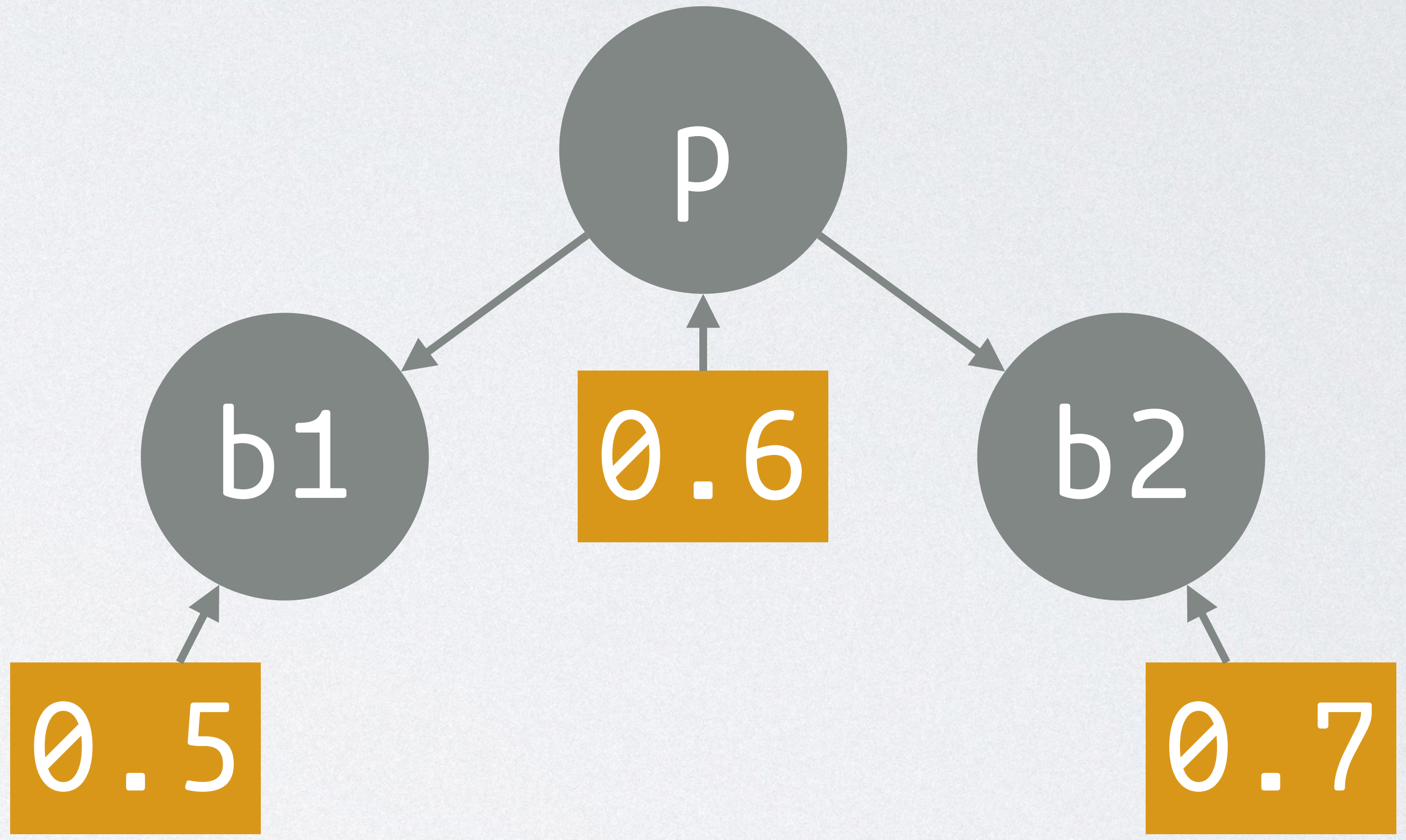


Bayesian Networks as Prob. Prog.



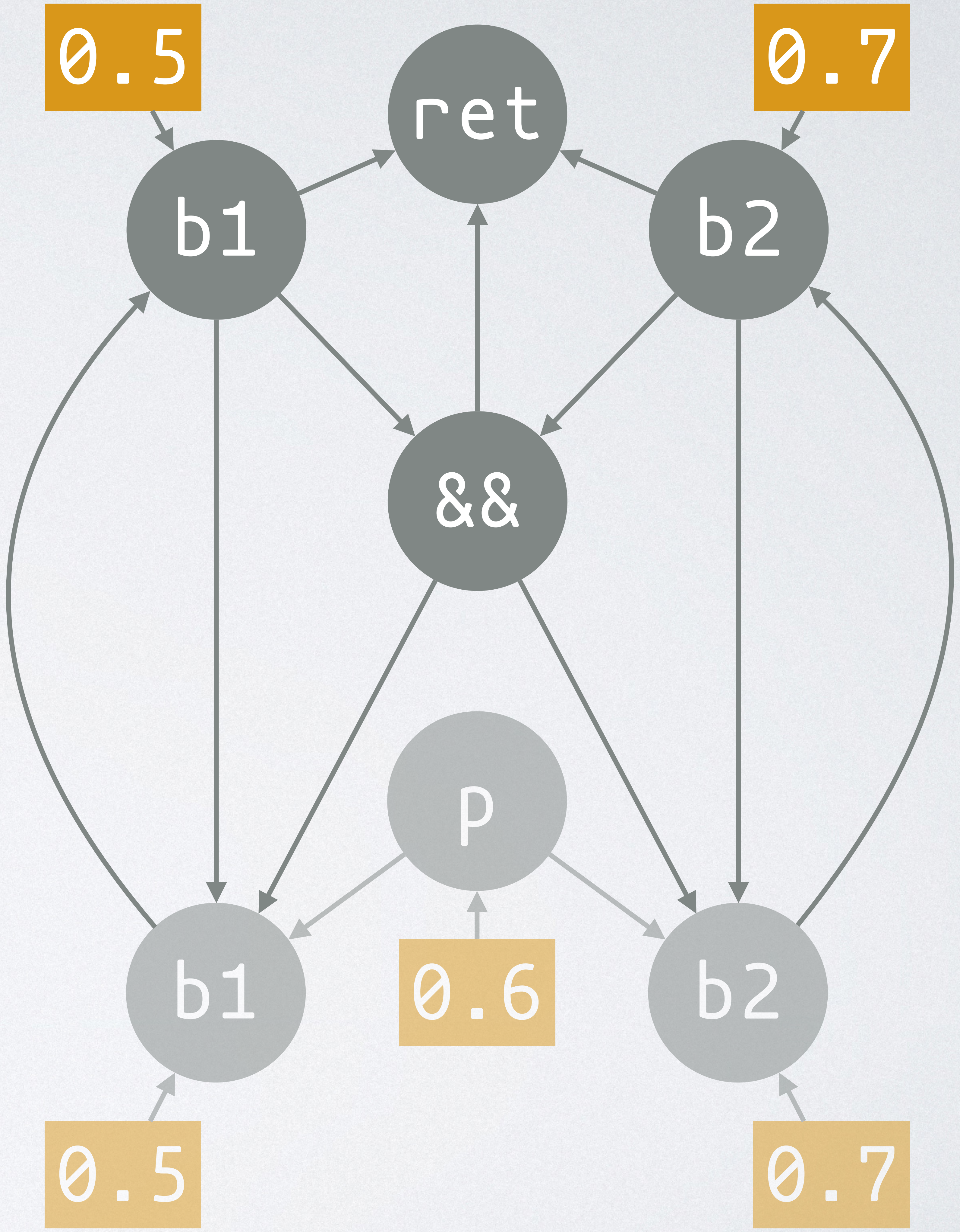
Bayesian Networks as Prob. Prog.

```
if prob(0.6) then
  b1 ~ Bernoulli(0.5)
else
  b2 ~ Bernoulli(0.7)
fi
```



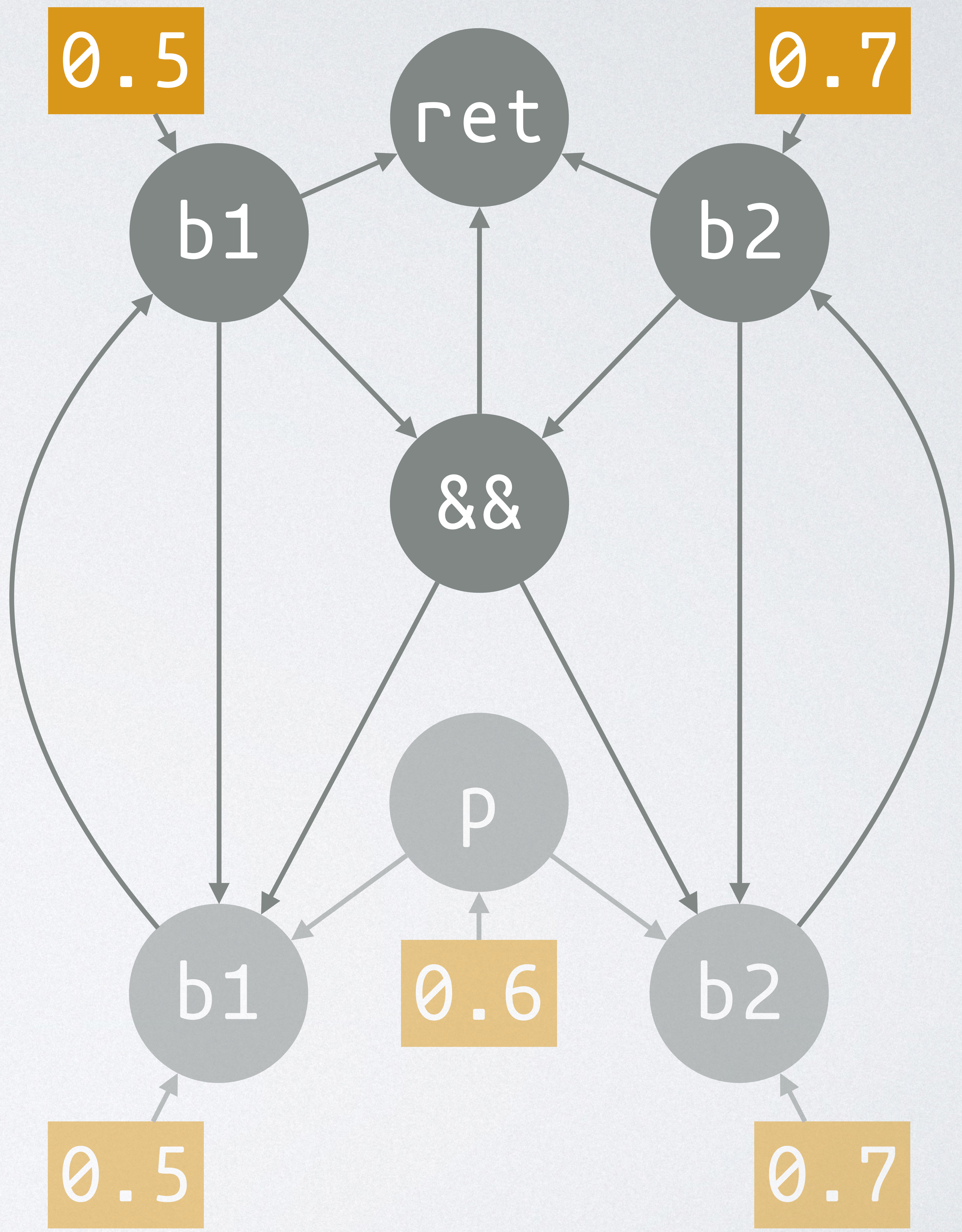
Bayesian Networks as Prob. Prog.

```
b1 ~ Bernoulli(0.5);
b2 ~ Bernoulli(0.7);
while (b1 && b2) do
   if prob(0.6) then
      b1 ~ Bernoulli(0.5)
   else
      b2 ~ Bernoulli(0.7)
   fi;
   tick(1.0)
od;
return (b1, b2)
```



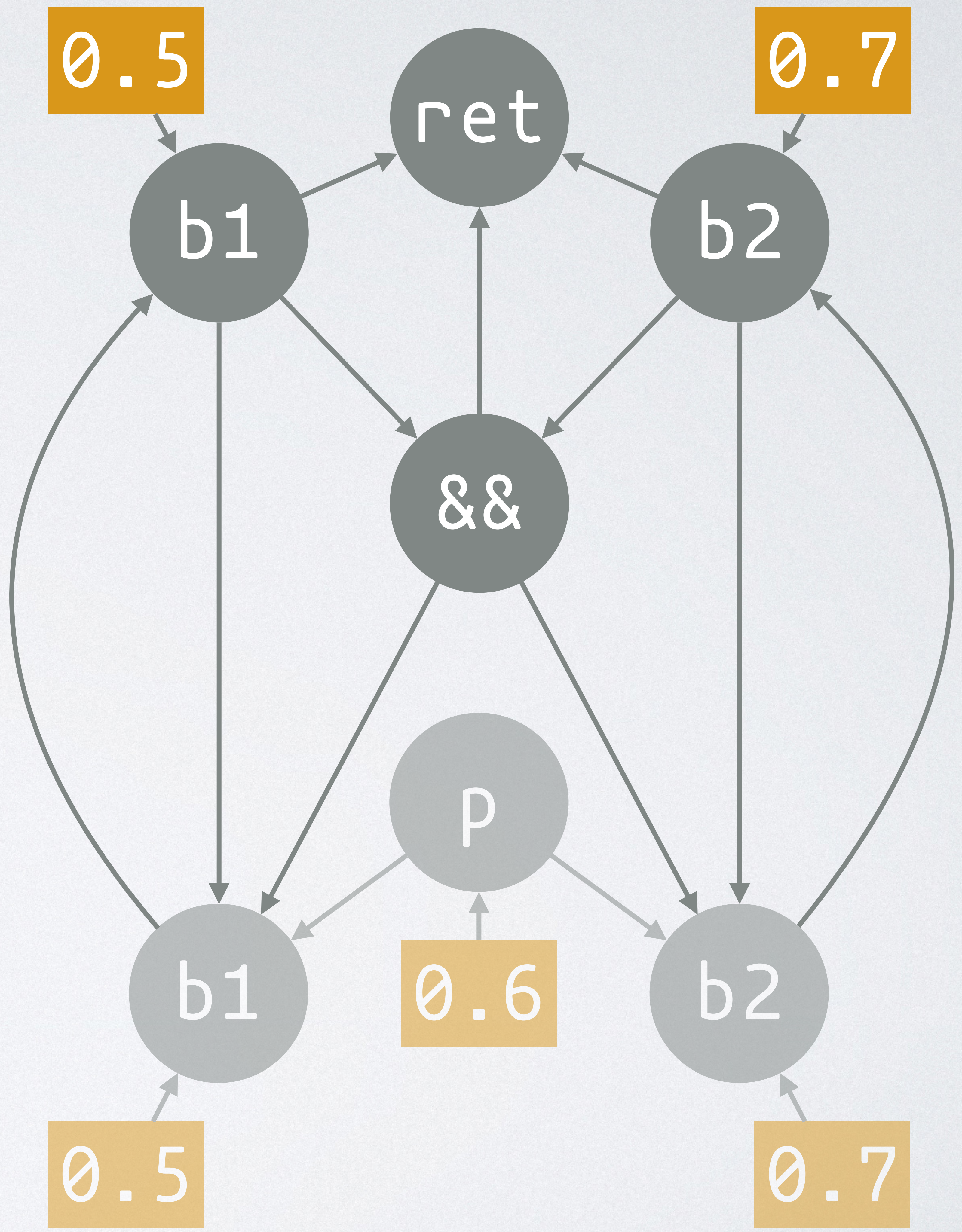
Quantitative Reasoning about Prob. Prog.

```
b1 ~ Bernoulli(0.5);
b2 ~ Bernoulli(0.7);
while (b1 && b2) do
   if prob(0.6) then
      b1 ~ Bernoulli(0.5)
   else
      b2 ~ Bernoulli(0.7)
   fi;
   tick(1.0)
od;
return (b1, b2)
```



Quantitative Reasoning about Prob. Prog.

```
b1 ~ Bernoulli(0.5);
b2 ~ Bernoulli(0.7);
while (b1 && b2) do
  if prob(0.6) then
    b1 ~ Bernoulli(0.5)
  else
    b2 ~ Bernoulli(0.7)
  fi;
  tick(1.0)
od;
return (b1, b2)
```

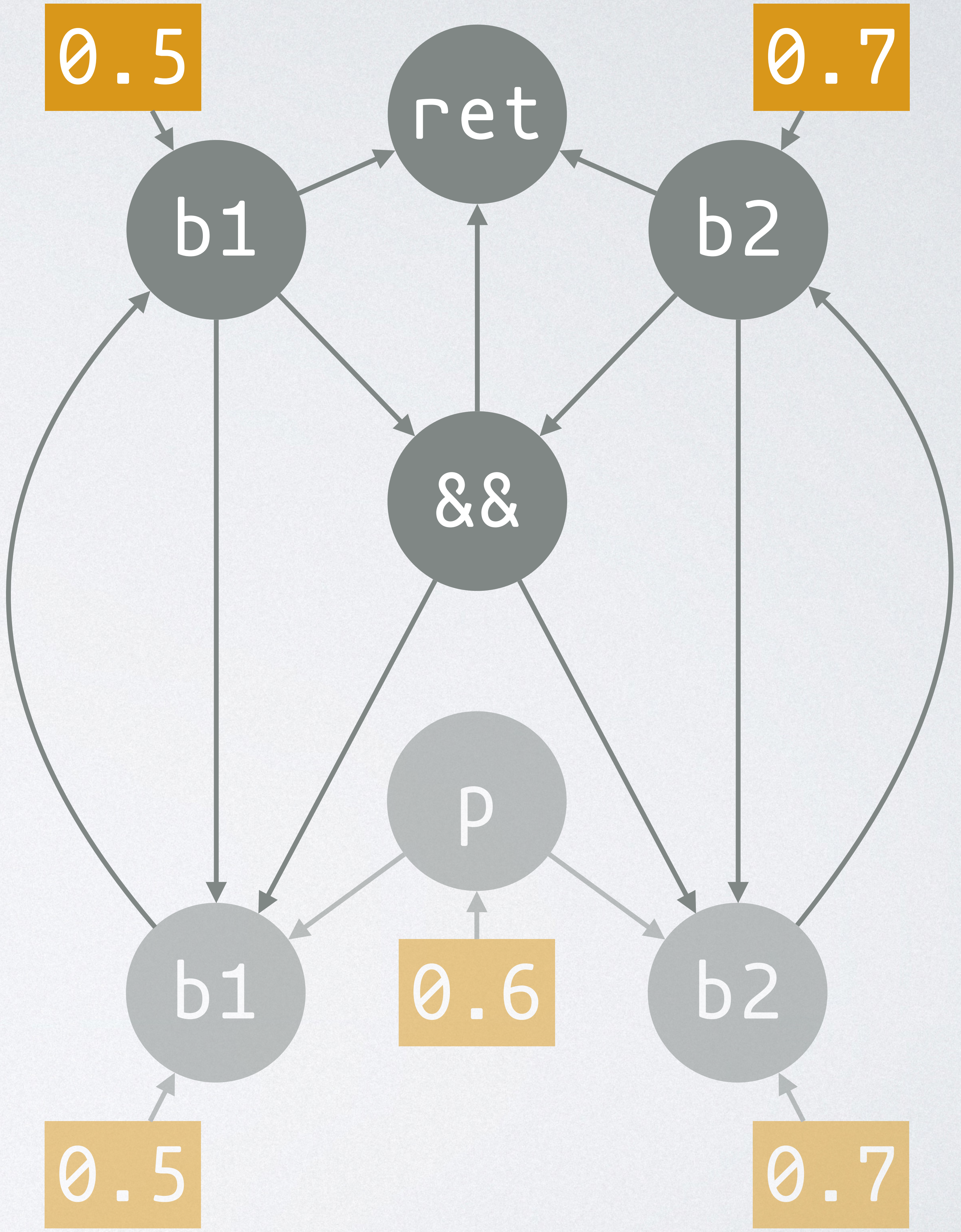


- **Query:** probability that `b1` and `b2` are both `false`?

Quantitative Reasoning about Prob. Prog.

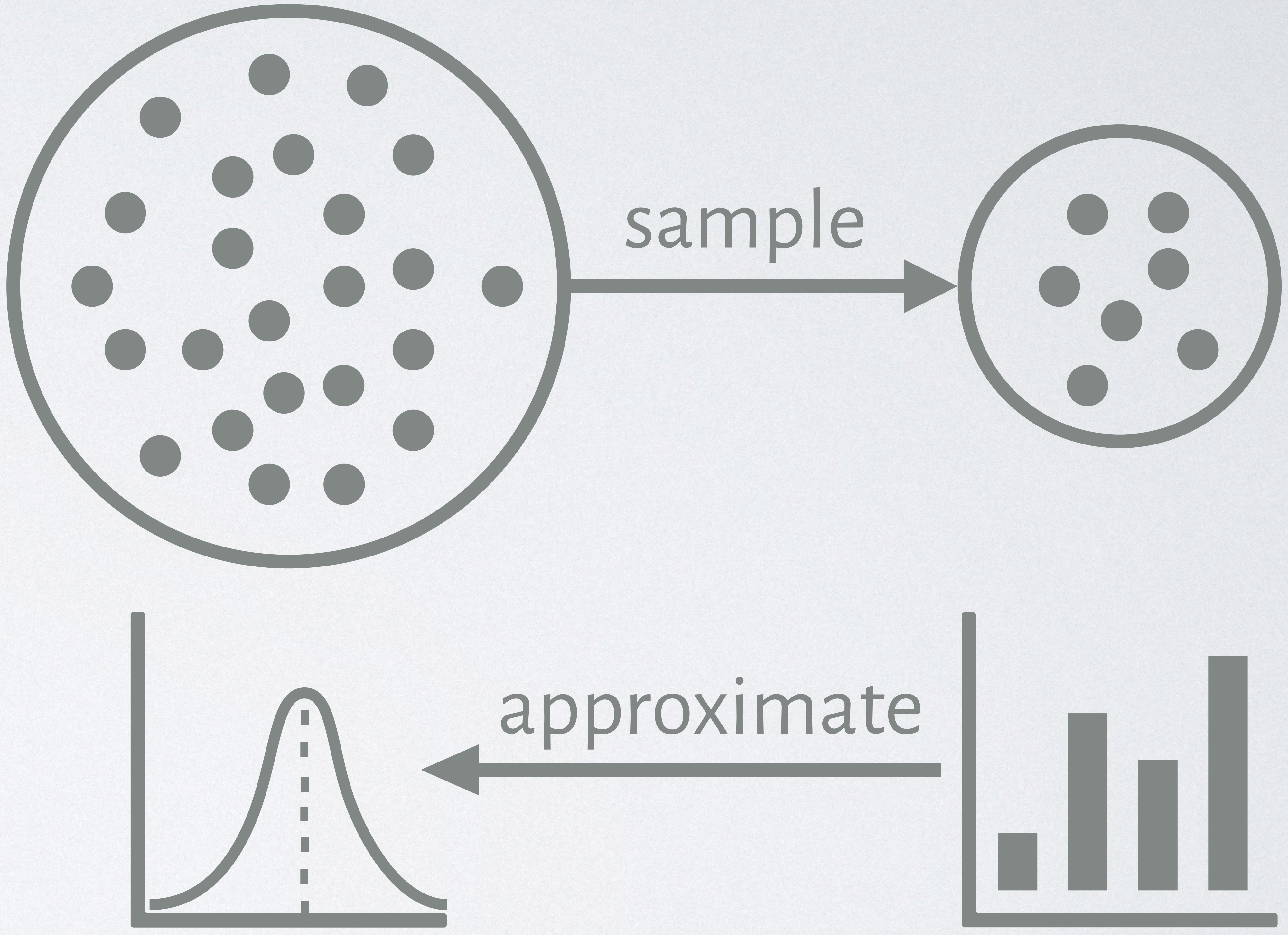
```
b1 ~ Bernoulli(0.5);
b2 ~ Bernoulli(0.7);
while (b1 && b2) do
  if prob(0.6) then
    b1 ~ Bernoulli(0.5)
  else
    b2 ~ Bernoulli(0.7)
  fi;
  tick(1.0)
od;
return (b1, b2)
```

- **Query:** expected termination time?



Sampling-Based Techniques

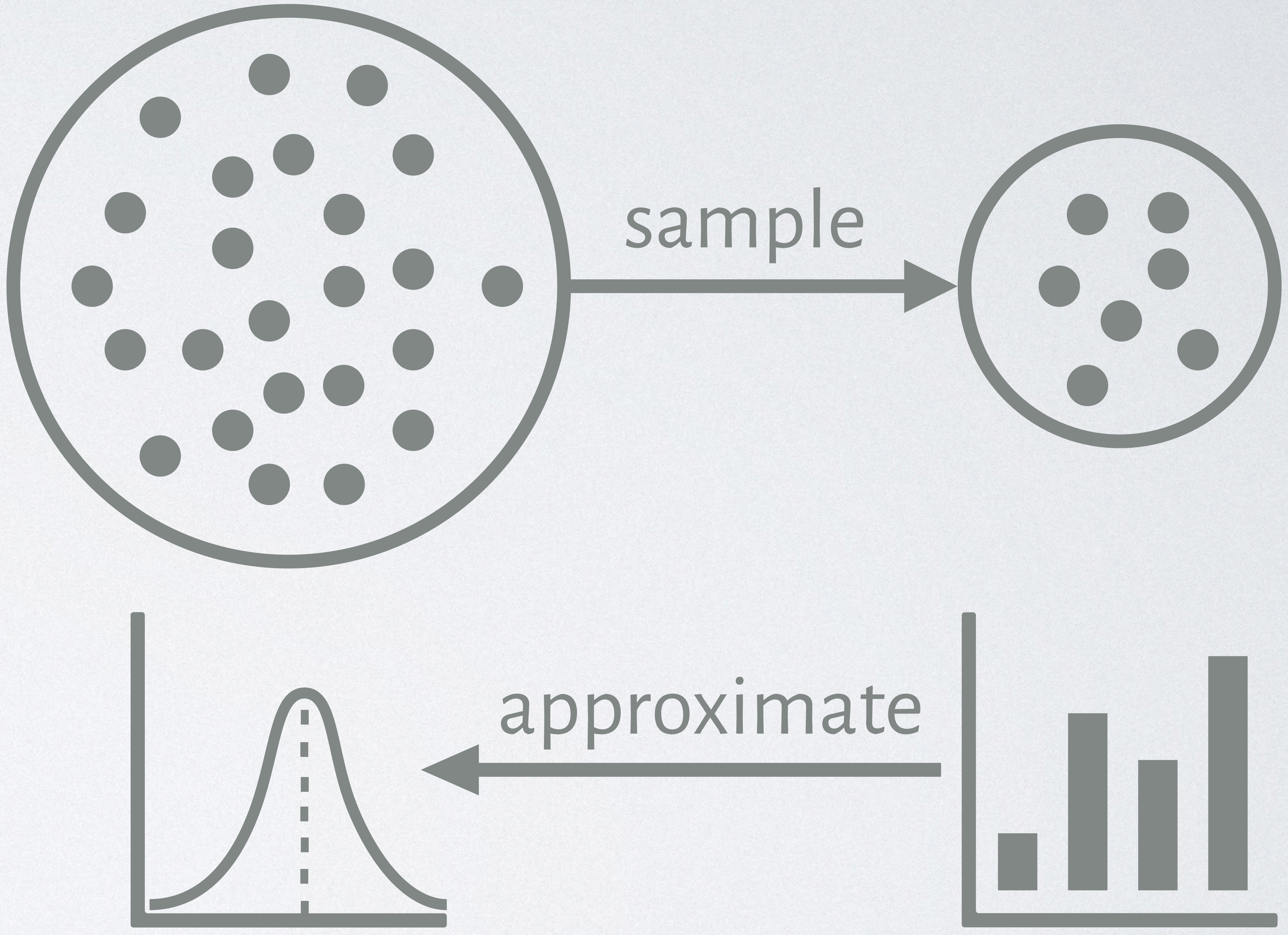
- Simulation & frequency count
- Flexible & universal
- Potentially unsound & inefficient



Sampling-Based Techniques

- Simulation & frequency count
- Flexible & universal
- Potentially unsound & inefficient

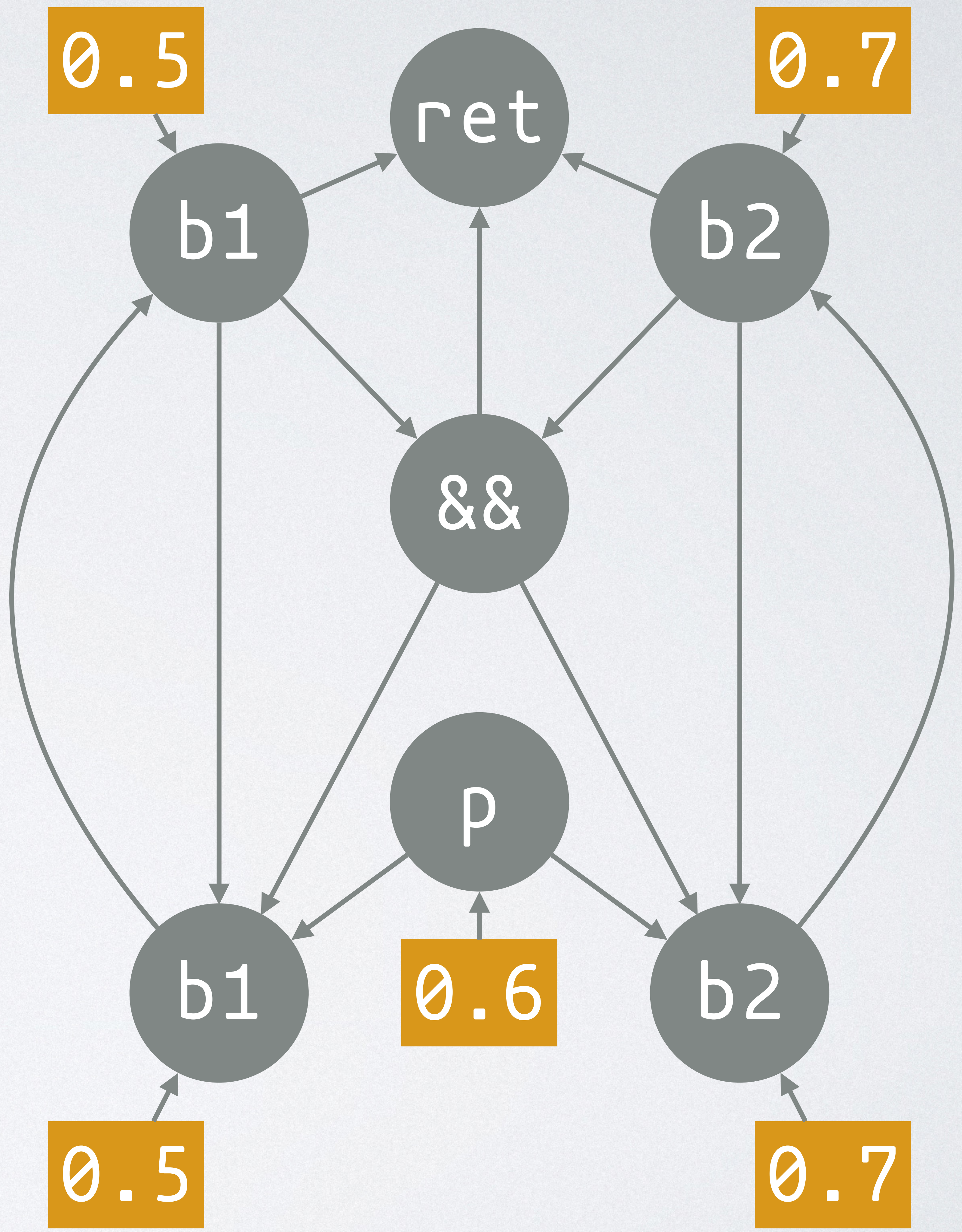
What about static analysis?



SEMANTICS OF PROB. PROG.

```
b1 ~ Bernoulli(0.5);
b2 ~ Bernoulli(0.7);
while (b1 && b2) do
  if prob(0.6) then
    b1 ~ Bernoulli(0.5)
  else
    b2 ~ Bernoulli(0.7)
  fi;
  tick(1.0)
od;
return (b1, b2)
```

- To develop static analysis, we need to first define a proper semantics



Semantics of Prob. Prog.

[1] J. Borgström, U. D. Lago, A. D. Gordon, and M. Szymczak. A Lambda-Calculus Foundation for Universal Probabilistic Programming. In *ICFP'16*.
[2] F. Ferrer, M. Luis, and H. Hermanns. Probabilistic Termination: Soundness, Completeness, and Compositionality. In *POPL'15*.
[3] S. Staton, H. Yang, C. Heunen, and O. Kammar. Semantics for Probabilistic Programming: Higher-Order Functions, Continuous Distributions, and Soft Constraints. In *LICS'16*.
[4] C. Heunen, O. Kammar, S. Staton, and H. Yang. A Convenient Category for Higher-Order Probability Theory. In *LICS'17*.

Semantics of Prob. Prog.

Existing Operational Semantics

- Untyped lambda calculus[1]
- Markov chains[2]

[1] J. Borgström, U. D. Lago, A. D. Gordon, and M. Szymczak. A Lambda-Calculus Foundation for Universal Probabilistic Programming. In *ICFP'16*.
[2] F. Ferrer, M. Luis, and H. Hermanns. Probabilistic Termination: Soundness, Completeness, and Compositionality. In *POPL'15*.
[3] S. Staton, H. Yang, C. Heunen, and O. Kammar. Semantics for Probabilistic Programming: Higher-Order Functions, Continuous Distributions, and Soft Constraints. In *LICS'16*.
[4] C. Heunen, O. Kammar, S. Staton, and H. Yang. A Convenient Category for Higher-Order Probability Theory. In *LICS'17*.

SEMANTICS OF PROB. PROG.

Existing Operational Semantics

- Untyped lambda calculus[1]
- Markov chains[2]

What was Missing?

- Compositionality

[1] J. Borgström, U. D. Lago, A. D. Gordon, and M. Szymczak. A Lambda-Calculus Foundation for Universal Probabilistic Programming. In *ICFP'16*.
[2] F. Ferrer, M. Luis, and H. Hermanns. Probabilistic Termination: Soundness, Completeness, and Compositionality. In *POPL'15*.
[3] S. Staton, H. Yang, C. Heunen, and O. Kammar. Semantics for Probabilistic Programming: Higher-Order Functions, Continuous Distributions, and Soft Constraints. In *LICS'16*.
[4] C. Heunen, O. Kammar, S. Staton, and H. Yang. A Convenient Category for Higher-Order Probability Theory. In *LICS'17*.

Semantics of Prob. Prog.

Existing Operational Semantics

- Untyped lambda calculus[1]
- Markov chains[2]

Existing Denotational Semantics

- First-order programs[3]
- Higher-order programs[4]

What was Missing?

- Compositionality

[1] J. Borgström, U. D. Lago, A. D. Gordon, and M. Szymczak. A Lambda-Calculus Foundation for Universal Probabilistic Programming. In *ICFP'16*.
[2] F. Ferrer, M. Luis, and H. Hermanns. Probabilistic Termination: Soundness, Completeness, and Compositionality. In *POPL'15*.
[3] S. Staton, H. Yang, C. Heunen, and O. Kammar. Semantics for Probabilistic Programming: Higher-Order Functions, Continuous Distributions, and Soft Constraints. In *LICS'16*.
[4] C. Heunen, O. Kammar, S. Staton, and H. Yang. A Convenient Category for Higher-Order Probability Theory. In *LICS'17*.

Semantics of Prob. Prog.

Existing Operational Semantics

- Untyped lambda calculus[1]
- Markov chains[2]

Existing Denotational Semantics

- First-order programs[3]
- Higher-order programs[4]

What was Missing?

- Compositionality

What was Missing?

- A general treatment of nondeterminism

[1] J. Borgström, U. D. Lago, A. D. Gordon, and M. Szymczak. A Lambda-Calculus Foundation for Universal Probabilistic Programming. In *ICFP'16*.
[2] F. Ferrer, M. Luis, and H. Hermanns. Probabilistic Termination: Soundness, Completeness, and Compositionality. In *POPL'15*.
[3] S. Staton, H. Yang, C. Heunen, and O. Kammar. Semantics for Probabilistic Programming: Higher-Order Functions, Continuous Distributions, and Soft Constraints. In *LICS'16*.
[4] C. Heunen, O. Kammar, S. Staton, and H. Yang. A Convenient Category for Higher-Order Probability Theory. In *LICS'17*.

Static Analysis of Prob. Prog.

[5] P. Cousot and M. Monerau. Probabilistic Abstract Interpretation. In *ESOP'12*.

Static Analysis of Prob. Prog.

Existing Approaches

- Static analysis of different kinds of program properties for prob. prog.
- Probabilistic Abstract Interpretation (**PAI**)[5]

[5] P. Cousot and M. Monerau. Probabilistic Abstract Interpretation. In *ESOP'12*.

STATIC ANALYSIS OF PROB. PROG.

Existing Approaches

- Static analysis of different kinds of program properties for prob. prog.
- Probabilistic Abstract Interpretation (**PAI**)[5]

What was Missing?

- A unifying framework that covers multiple analyses
- Compositionality and flexibility
- **PAI**'s treatment of nondeterminism sometimes turns out to be *not desirable*

[5] P. Cousot and M. Monerau. Probabilistic Abstract Interpretation. In *ESOP'12*.

Algebraic Program Analysis[6]

[6] Z. Kincaid, T. Reps, and J. Cyphert. Algebraic Program Analysis. In *CAV'21*.

Algebraic Program Analysis[6]

- A flexible framework for understanding compositional static analyses

[6] Z. Kincaid, T. Reps, and J. Cyphert. Algebraic Program Analysis. In *CAV'21*.

Algebraic Program Analysis[6]

- A flexible framework for understanding compositional static analyses
- Usually set up with an algebraic semantics, e.g., a ***Kleene algebra***

[6] Z. Kincaid, T. Reps, and J. Cyphert. Algebraic Program Analysis. In *CAV'21*.

ALGEBRAIC PROGRAM ANALYSIS[6]

- A flexible framework for understanding compositional static analyses
- Usually set up with an algebraic semantics, e.g., a ***Kleene algebra***

$$A \oplus B \text{ for } \textbf{\textit{branching}} \text{ between } A \text{ and } B$$

[6] Z. Kincaid, T. Reps, and J. Cyphert. Algebraic Program Analysis. In *CAV'21*.

ALGEBRAIC PROGRAM ANALYSIS[6]

- A flexible framework for understanding compositional static analyses
- Usually set up with an algebraic semantics, e.g., a ***Kleene algebra***

$A \oplus B$ for ***branching*** between A and B
$A \otimes B$ for ***sequencing*** of A and B

[6] Z. Kincaid, T. Reps, and J. Cyphert. Algebraic Program Analysis. In *CAV'21*.

Algebraic Program Analysis[6]

- A flexible framework for understanding compositional static analyses
- Usually set up with an algebraic semantics, e.g., a ***Kleene algebra***

$A \oplus B$ for ***branching*** between A and B
$A \otimes B$ for ***sequencing*** of A and B
A^* for the Kleene-star (a ***loop***) of A

[6] Z. Kincaid, T. Reps, and J. Cyphert. Algebraic Program Analysis. In *CAV'21*.

Algebraic Program Analysis[6]

- A flexible framework for understanding compositional static analyses
- Usually set up with an algebraic semantics, e.g., a ***Kleene algebra***

$A \oplus B$ for ***branching*** between A and B
$A \otimes B$ for ***sequencing*** of A and B
A^* for the Kleene-star (a ***loop***) of A

- Different algebras yield different semantics, e.g., a concrete semantics, or an abstract semantics for static analysis

[6] Z. Kincaid, T. Reps, and J. Cyphert. Algebraic Program Analysis. In *CAV'21*.

Thesis

- **Algebraic** static analysis helps people reason about prob. prog. at compile time in a **compositional** and **flexible** way
- **Markov algebras** provide a natural way to define a **denotational semantics** of prob. prog. with **nondeterminism**
- **An algebraic framework for static analysis** can cover multiple existing analyses and lay the foundation for new analyses

Overview

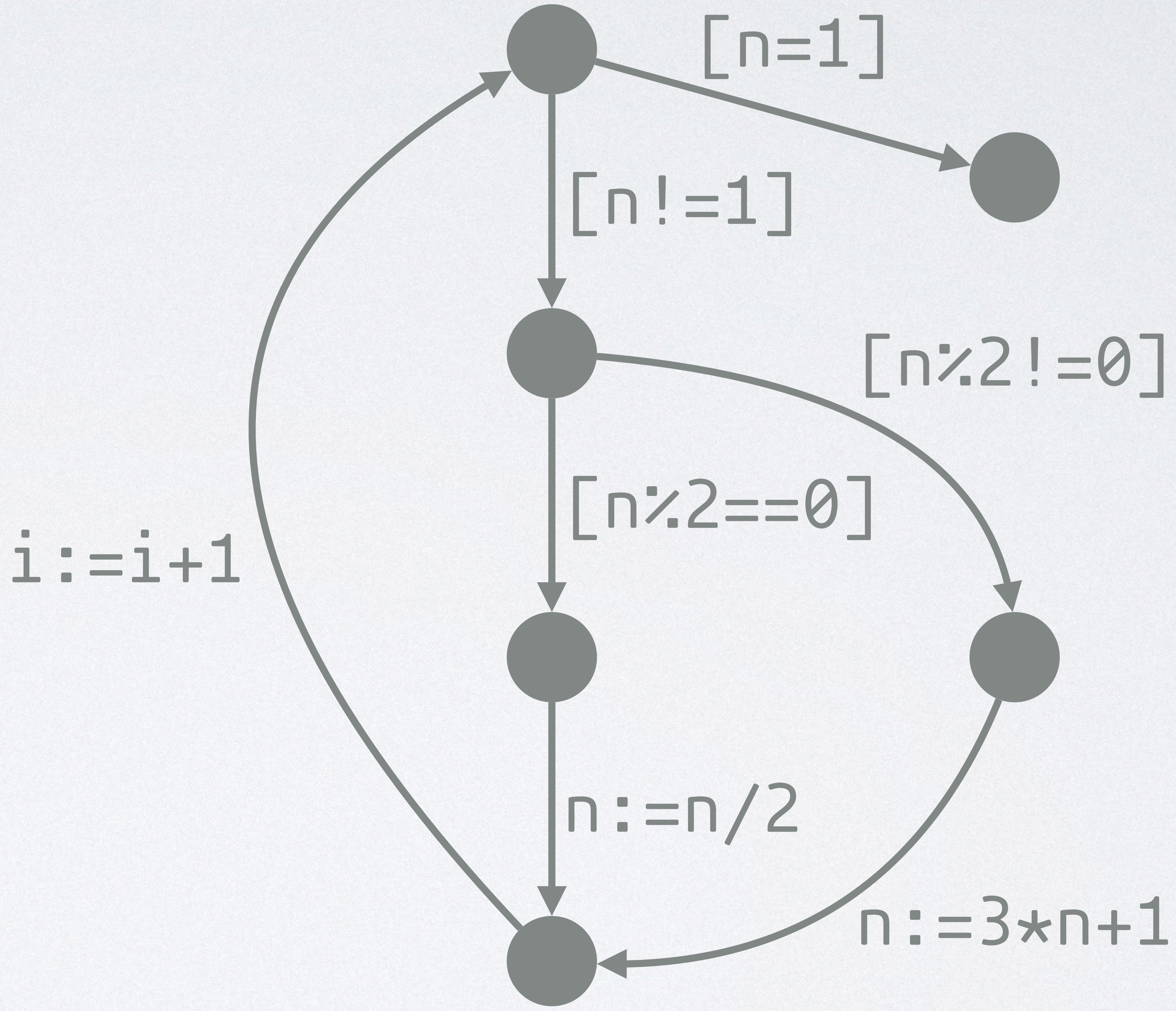
- ☑ Motivation
- ☐ An Algebraic Denotational Semantics
- ☐ Pre-Markov Algebra Framework (PMAF)

An Algebraic Denotational Semantics

Contributions

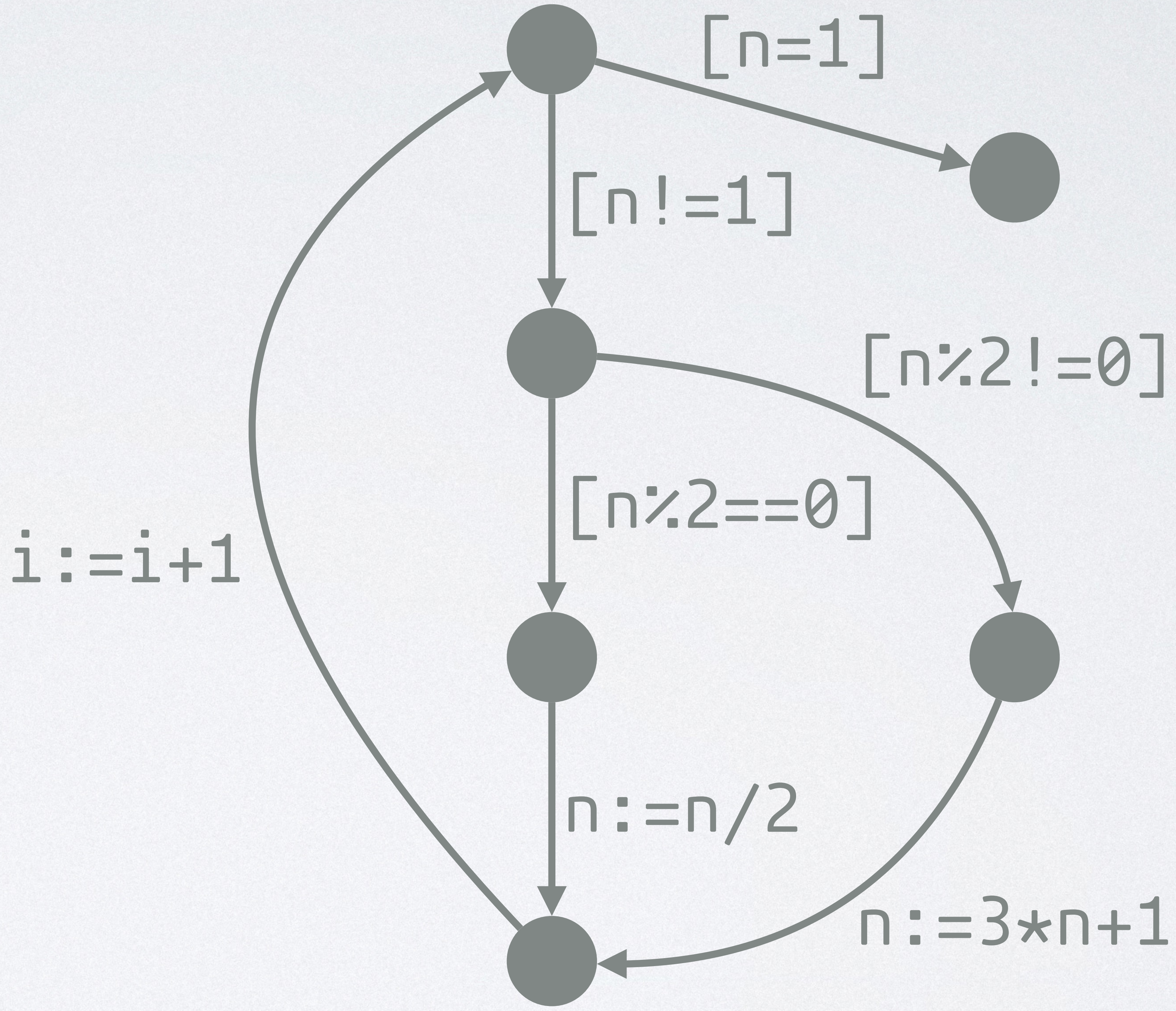
- A hyper-path semantics for low-level prob. prog.
- Markov algebras for understanding prob. prog. with nondeterminism
- A new model for resolving nondeterminism
- Published as **A Denotational Semantics for Low-Level Probabilistic Programs with Nondeterminism** in *MFPS'19*

Path Semantics of Non-Prob. Prog.



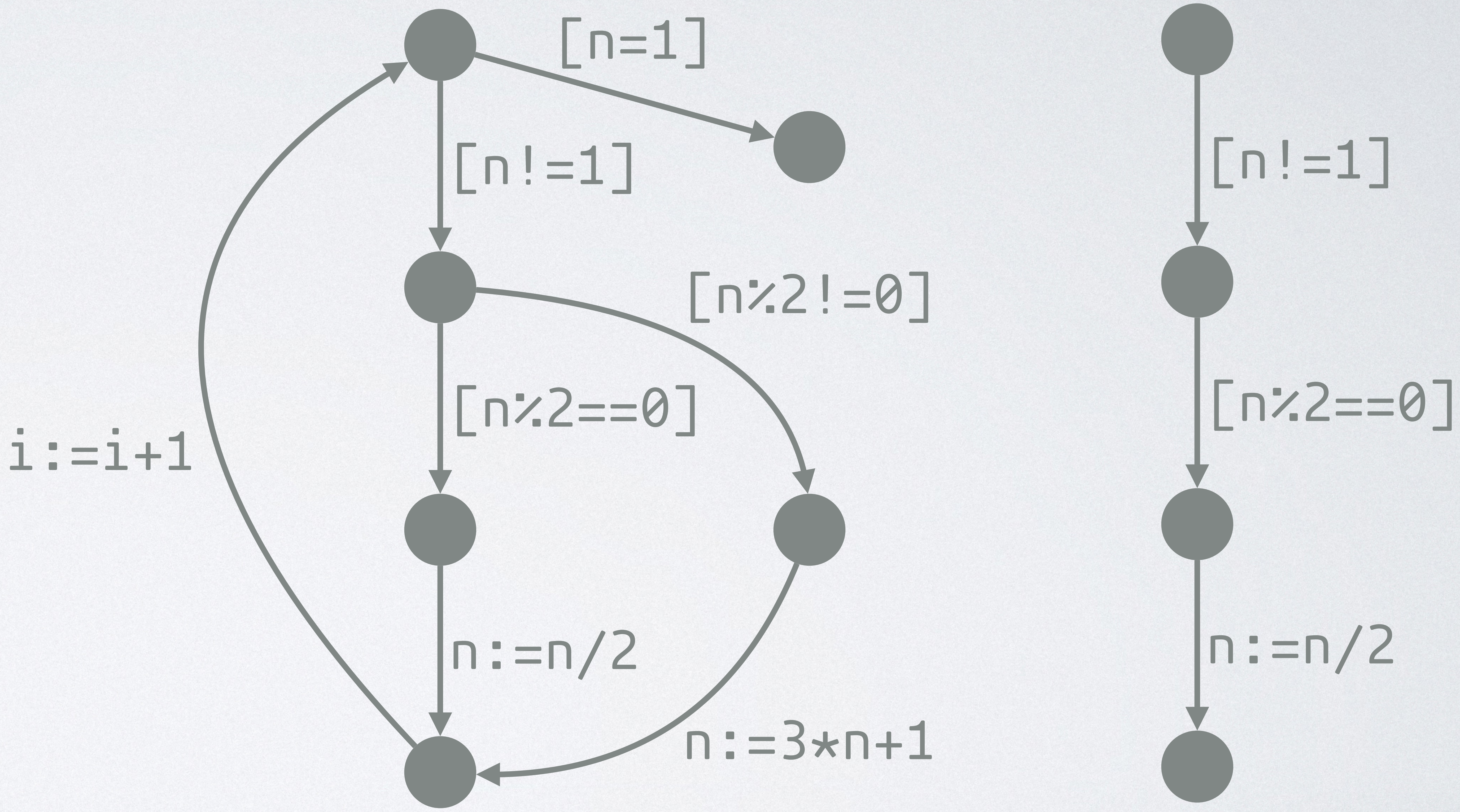
Path Semantics of Non-Prob. Prog.

- Control-flow graphs



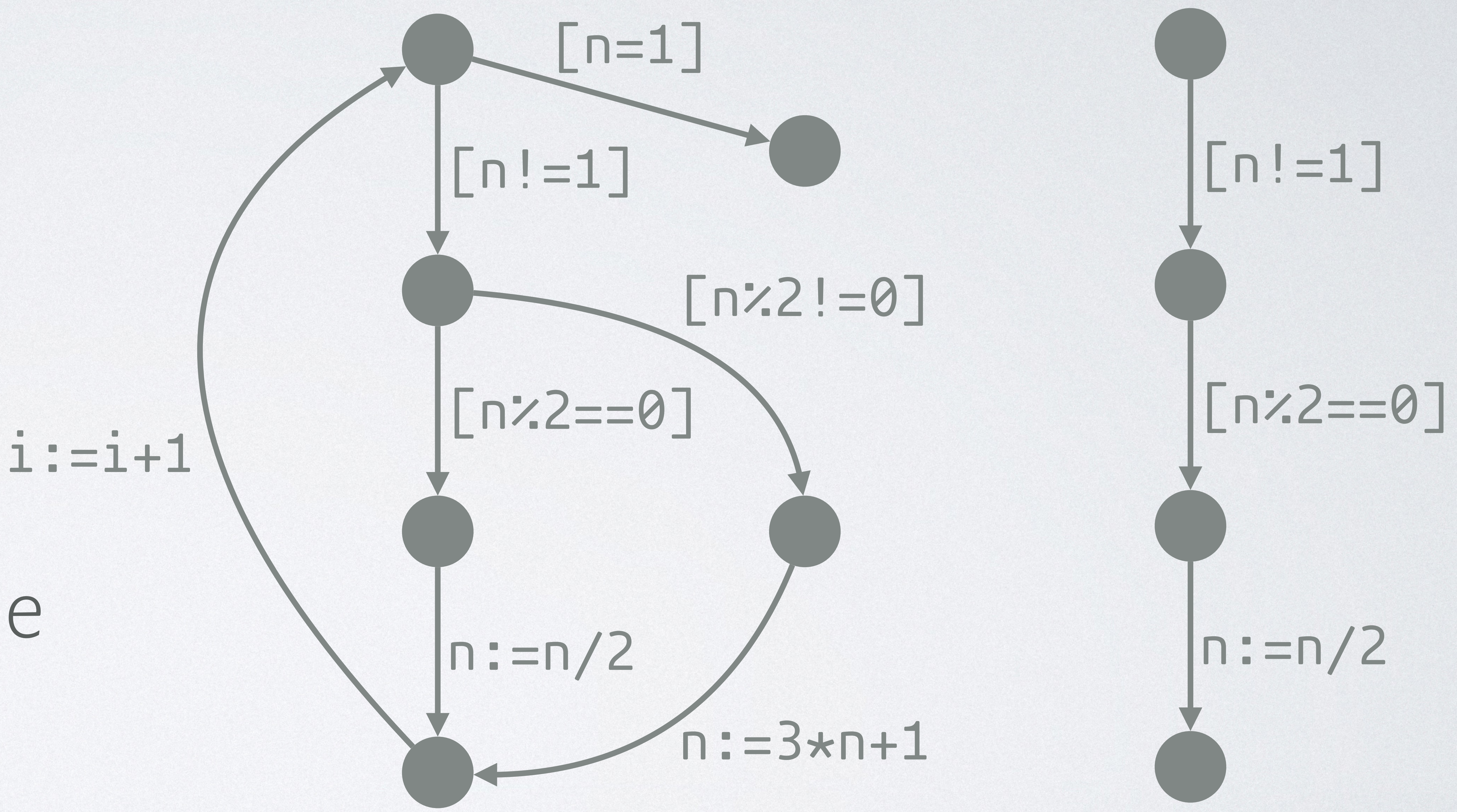
Path Semantics of Non-Prob. Prog.

- Control-flow graphs
- Reason about paths



Path Semantics of Non-Prob. Prog.

- Control-flow graphs
- Reason about paths
- Kleene algebras are suitable to describe path semantics



Control-Flow Graphs and Path Semantics

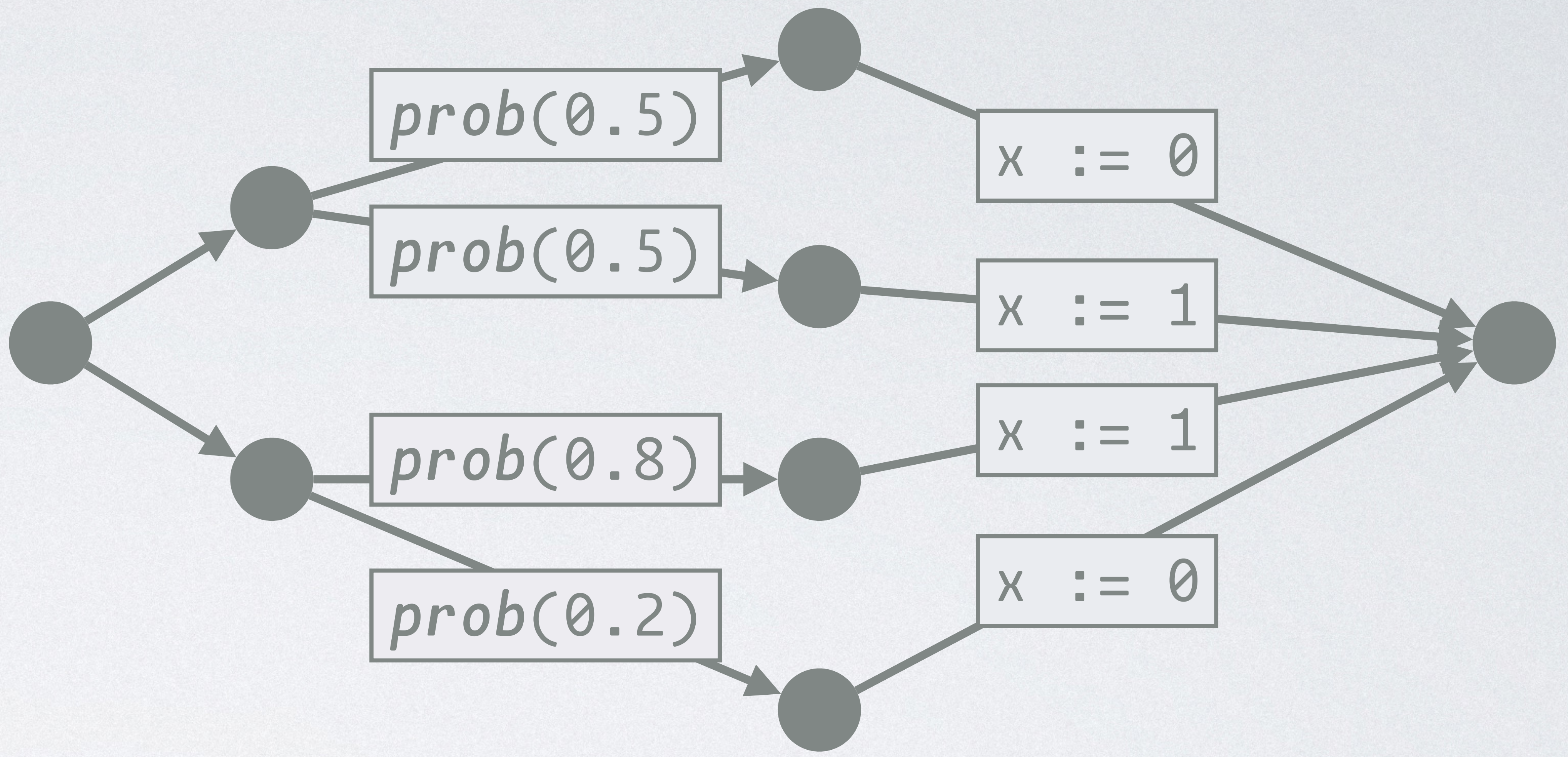
⋆ denotes nondeterministic-choice

```
if ⋆ then
  if prob(0.5) then
    x := 0
  else
    x := 1
  fi
else
  if prob(0.8) then
    x := 1
  else
    x := 0
  fi
fi
```

Control-Flow Graphs and Path Semantics

⋆ denotes nondeterministic-choice

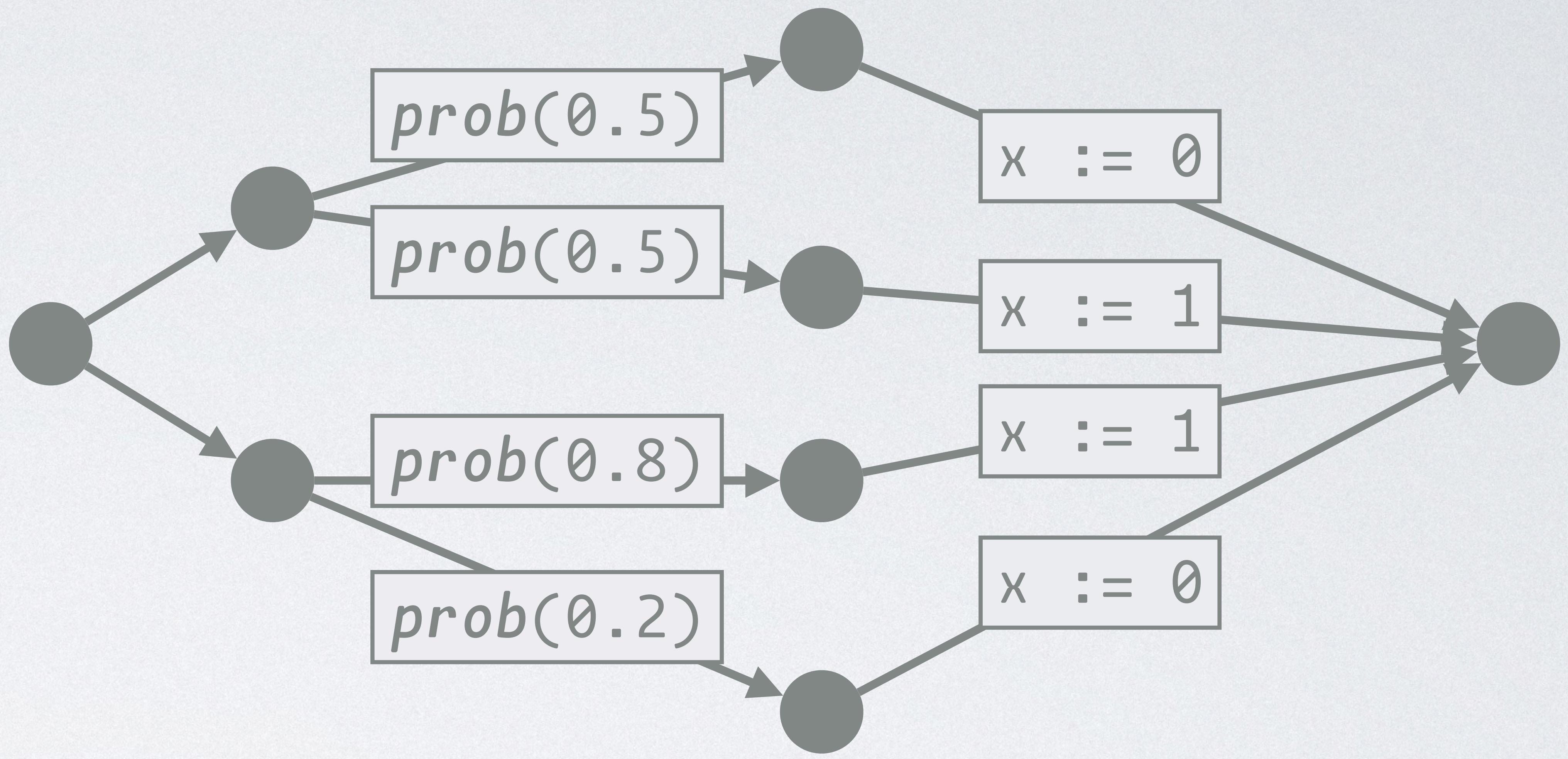
```
if ⋆ then
  if prob(0.5) then
    x := 0
  else
    x := 1
  fi
else
  if prob(0.8) then
    x := 1
  else
    x := 0
  fi
fi
```



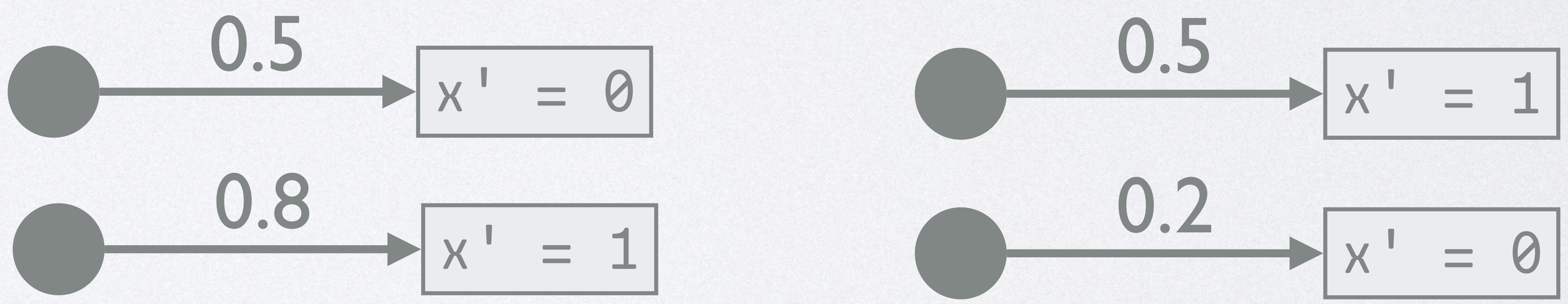
CONTROL-FLOW GRAPHS AND PATH SEMANTICS

⋆ denotes nondeterministic-choice

```
if * then
  if prob(0.5) then
    x := 0
  else
    x := 1
  fi
else
  if prob(0.8) then
    x := 1
  else
    x := 0
  fi
fi
```



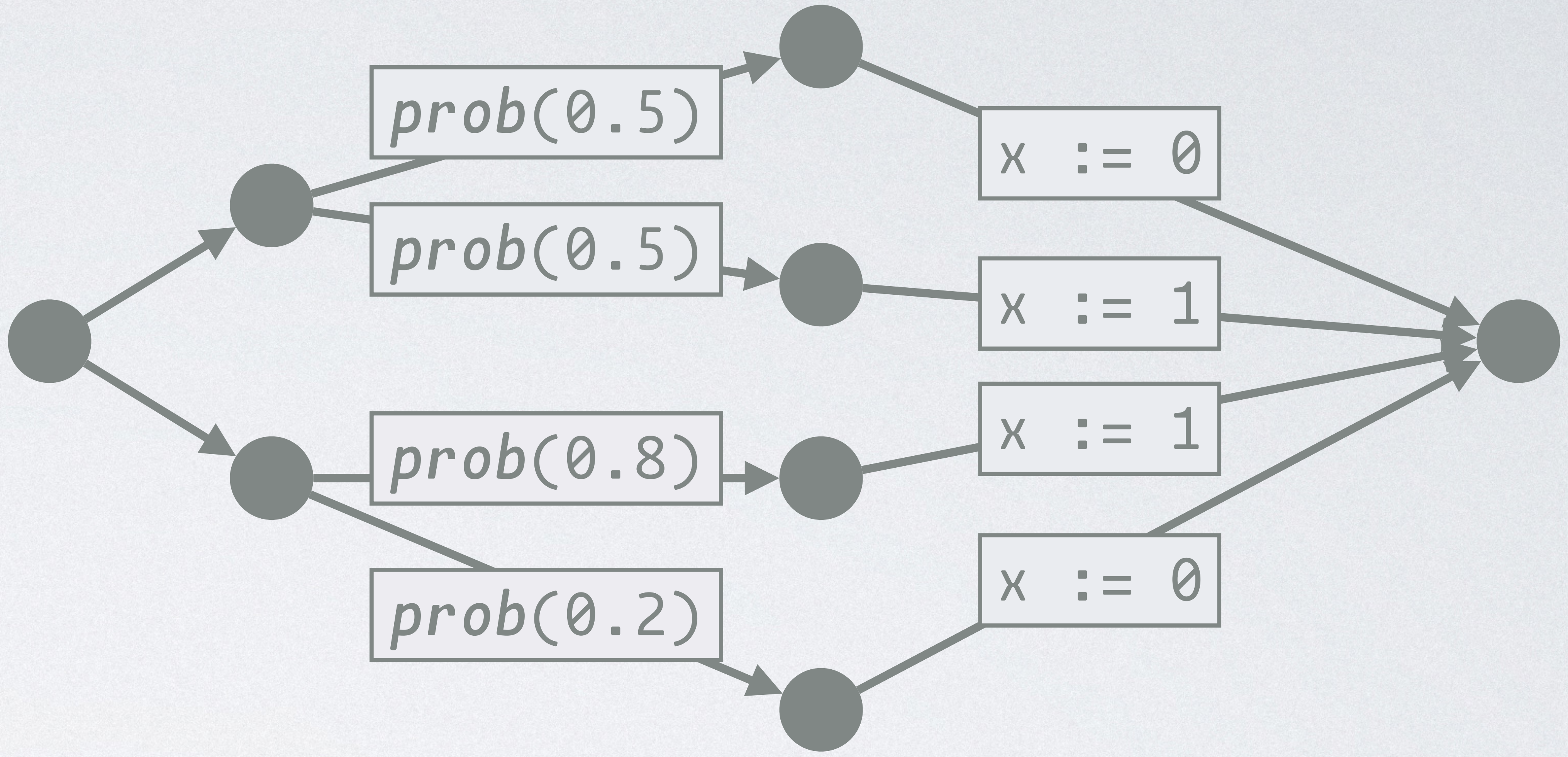
- Paths annotated with probabilities:



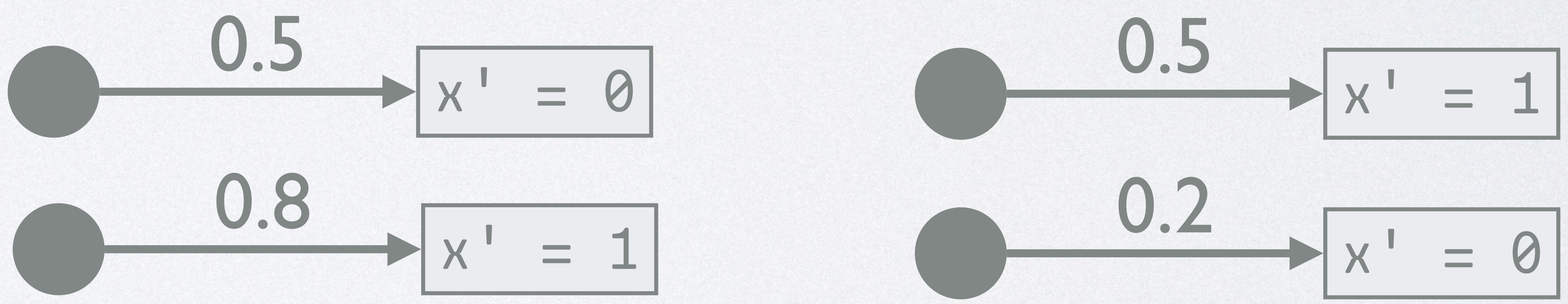
Control-Flow Graphs and Path Semantics

⋆ denotes nondeterministic-choice

```
if * then
  if prob(0.5) then
    x := 0
  else
    x := 1
  fi
else
  if prob(0.8) then
    x := 1
  else
    x := 0
  fi
fi
```



- Paths annotated with probabilities:



- $\mathbf{Prob}[x' = 1] = 1.3\,?$

Hyper-Path Semantics

⋆ denotes nondeterministic-choice

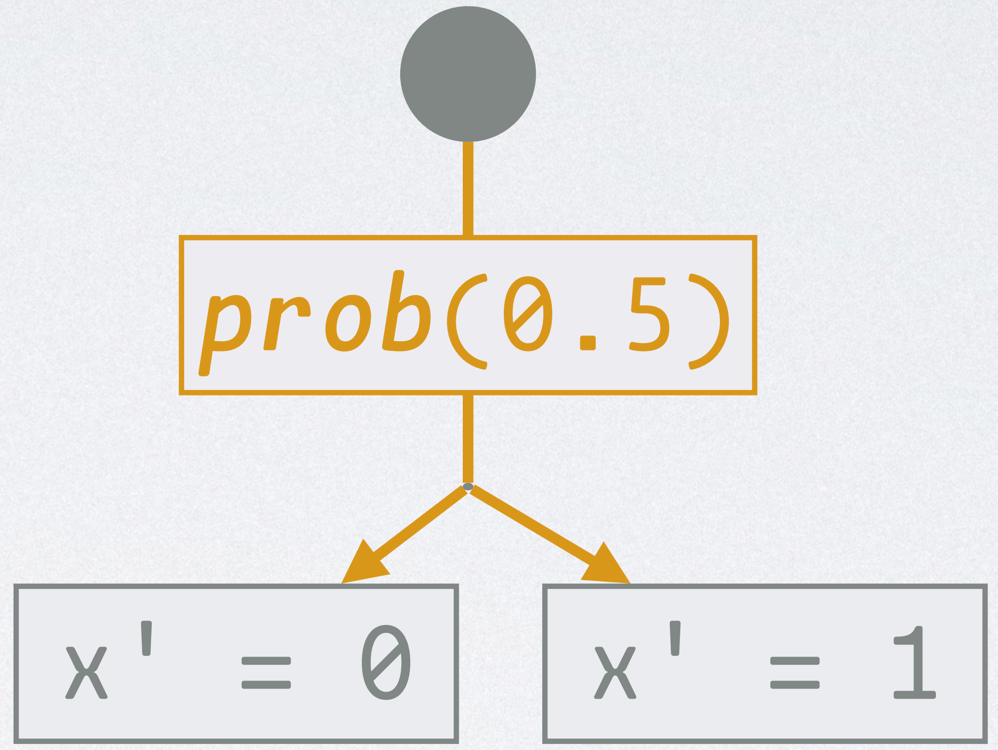
```
if ⋆ then
  if prob(0.5) then
    x := 0
  else
    x := 1
  fi
else
  if prob(0.8) then
    x := 1
  else
    x := 0
  fi
fi
```

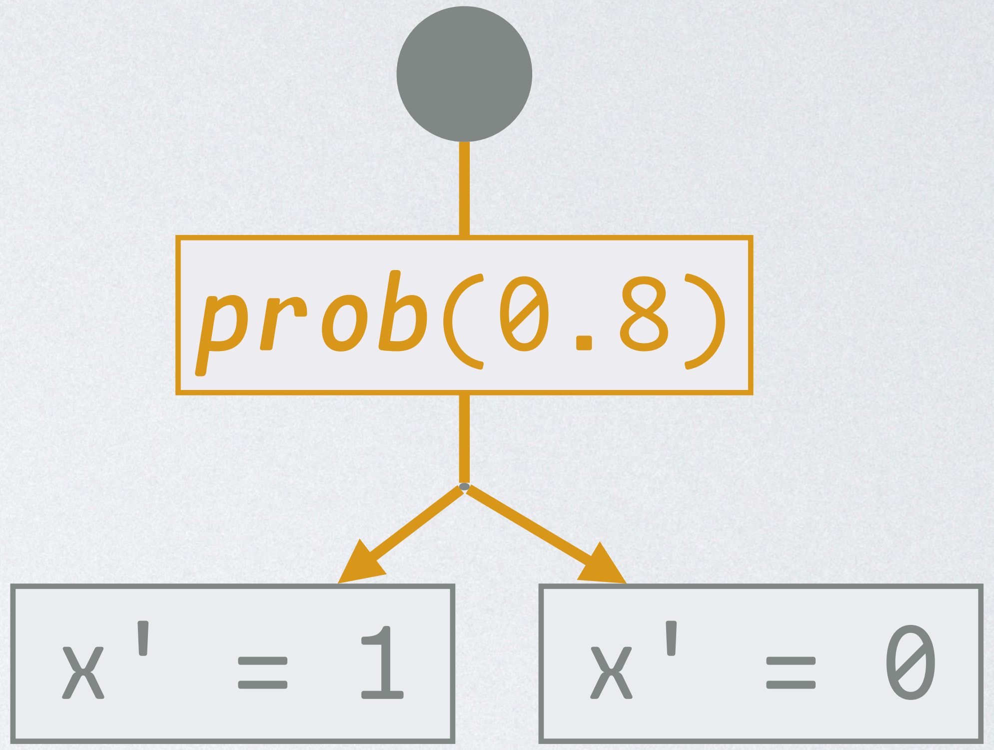
Hyper-Path Semantics

* denotes nondeterministic-choice

```
if * then
  if prob(0.5) then
    x := 0
  else
    x := 1
  fi
else
  if prob(0.8) then
    x := 1
  else
    x := 0
  fi
fi
```

- *prob*() introduces distributions on executions, and * collects such distributions



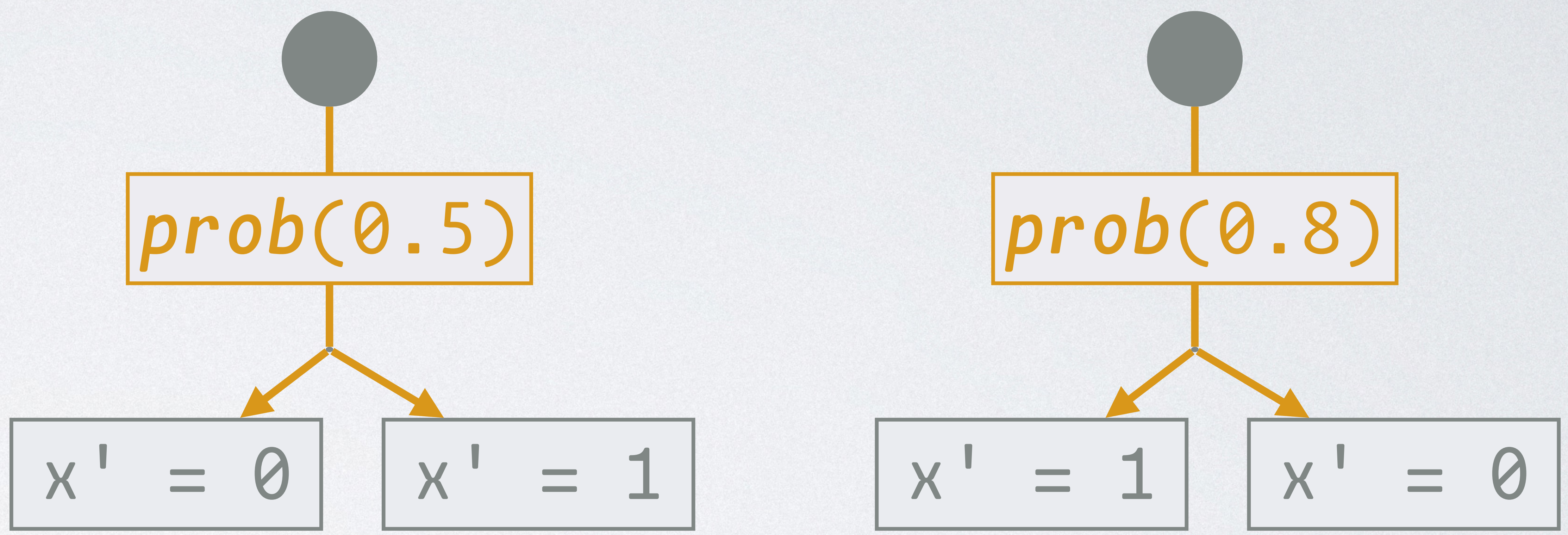


Hyper-Path Semantics

* denotes nondeterministic-choice

```
if * then
  if prob(0.5) then
    x := 0
  else
    x := 1
  fi
else
  if prob(0.8) then
    x := 1
  else
    x := 0
  fi
fi
```

- *prob*() introduces distributions on executions, and * collects such distributions



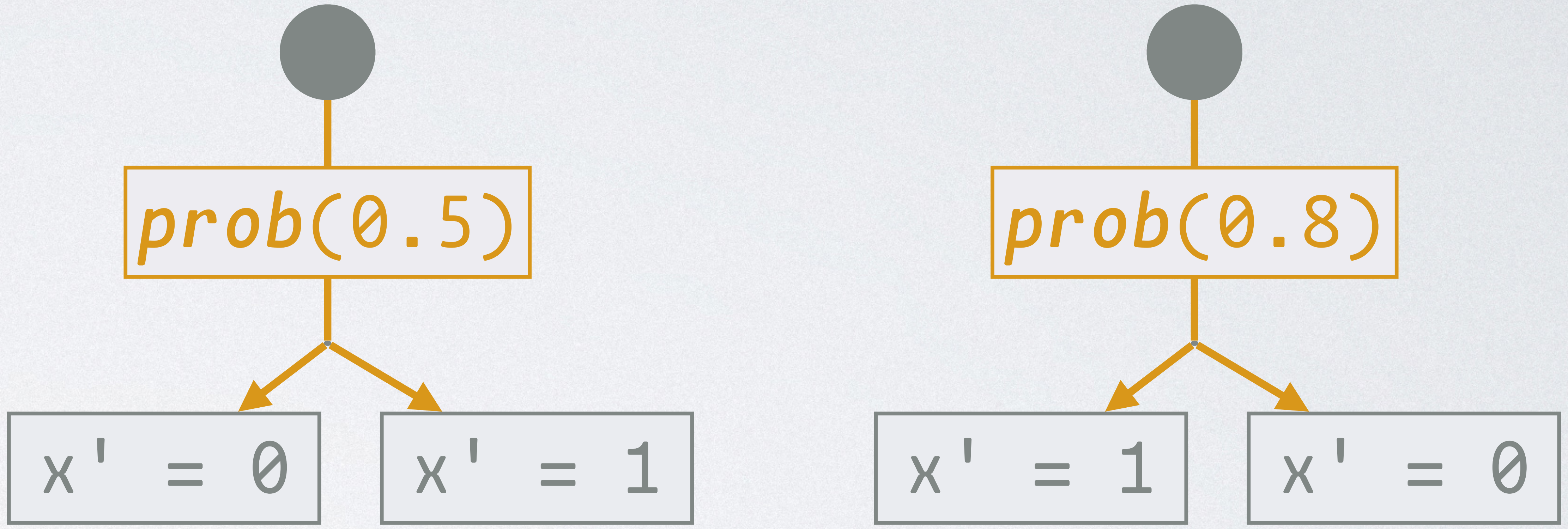
- A **hyper-path** represents a distribution on paths

Hyper-Path Semantics

* denotes nondeterministic-choice

```
if * then
  if prob(0.5) then
    x := 0
  else
    x := 1
  fi
else
  if prob(0.8) then
    x := 1
  else
    x := 0
  fi
fi
```

- *prob*() introduces distributions on executions, and * collects such distributions



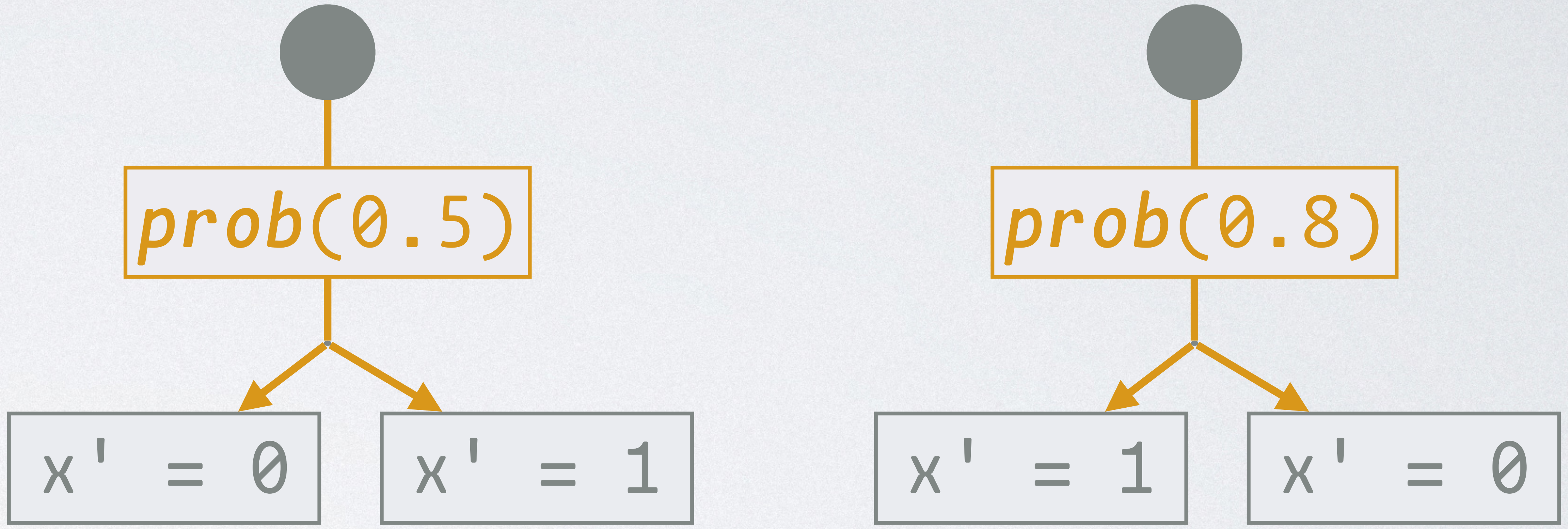
- A **hyper-path** represents a distribution on paths
- **Nondeterminism** is modeled by **collections** of hyper-paths

Hyper-Path Semantics

⋆ denotes nondeterministic-choice

```
if ⋆ then
  if prob(0.5) then
    x := 0
  else
    x := 1
  fi
else
  if prob(0.8) then
    x := 1
  else
    x := 0
  fi
fi
```

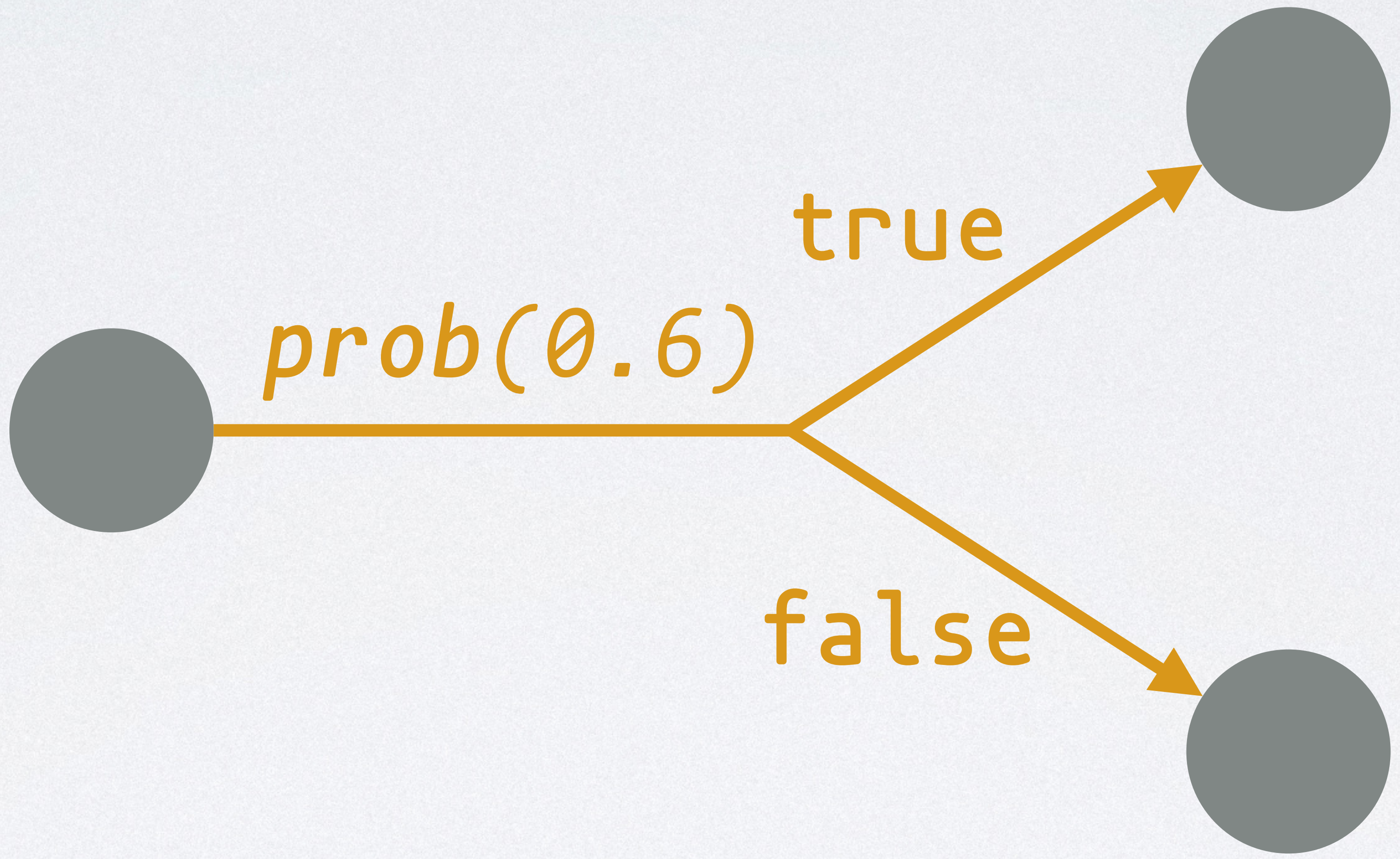
- *prob*() introduces distributions on executions, and ⋆ collects such distributions



- A **hyper-path** represents a distribution on paths
- **Nondeterminism** is modeled by **collections** of hyper-paths
- Kleene algebras are suitable for path semantics, but are ***not*** suitable for hyper-path semantics

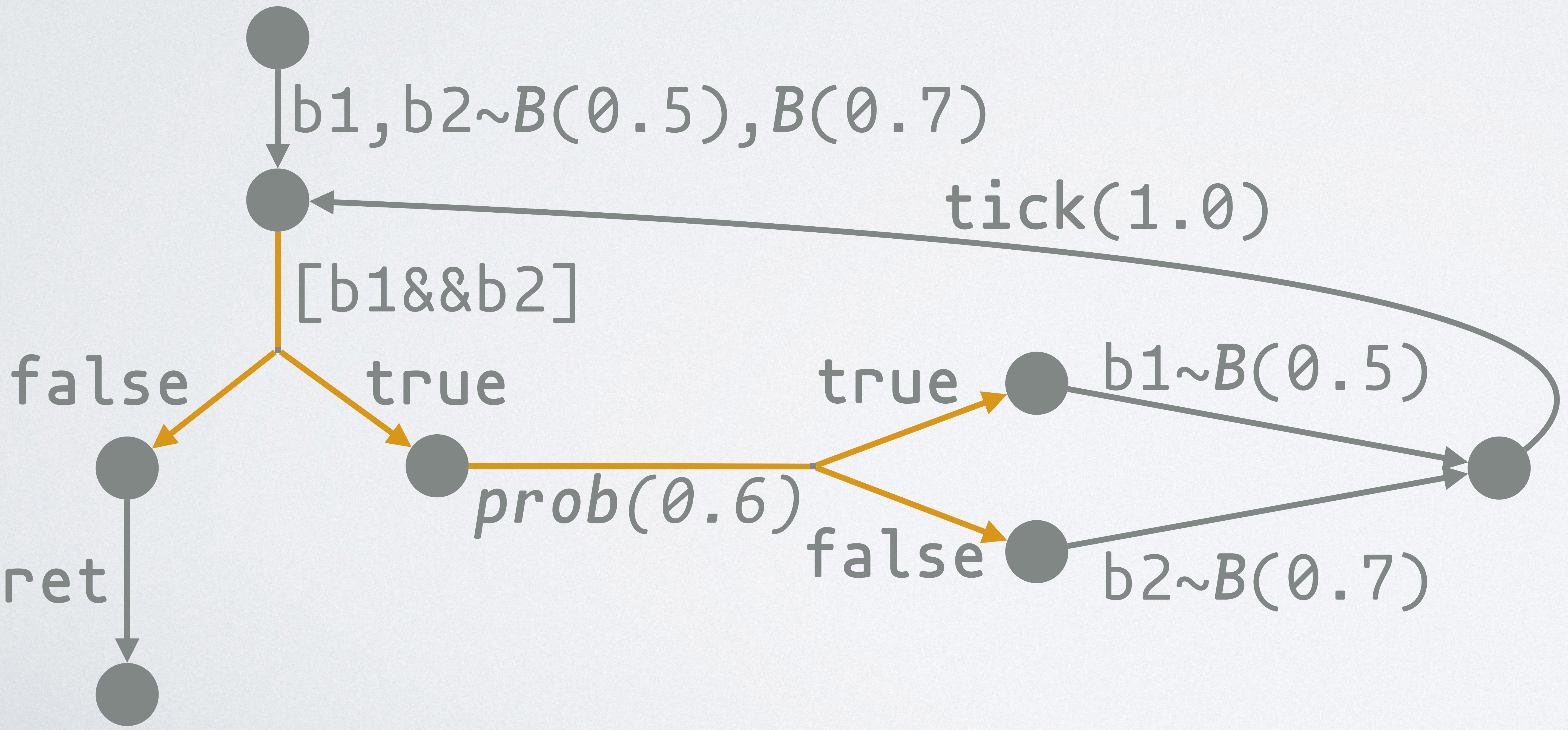
Control-Flow Hyper-Graphs (CFHGs)

A hyper-edge can have multiple destinations



Control-Flow Hyper-Graphs (CFHGs)

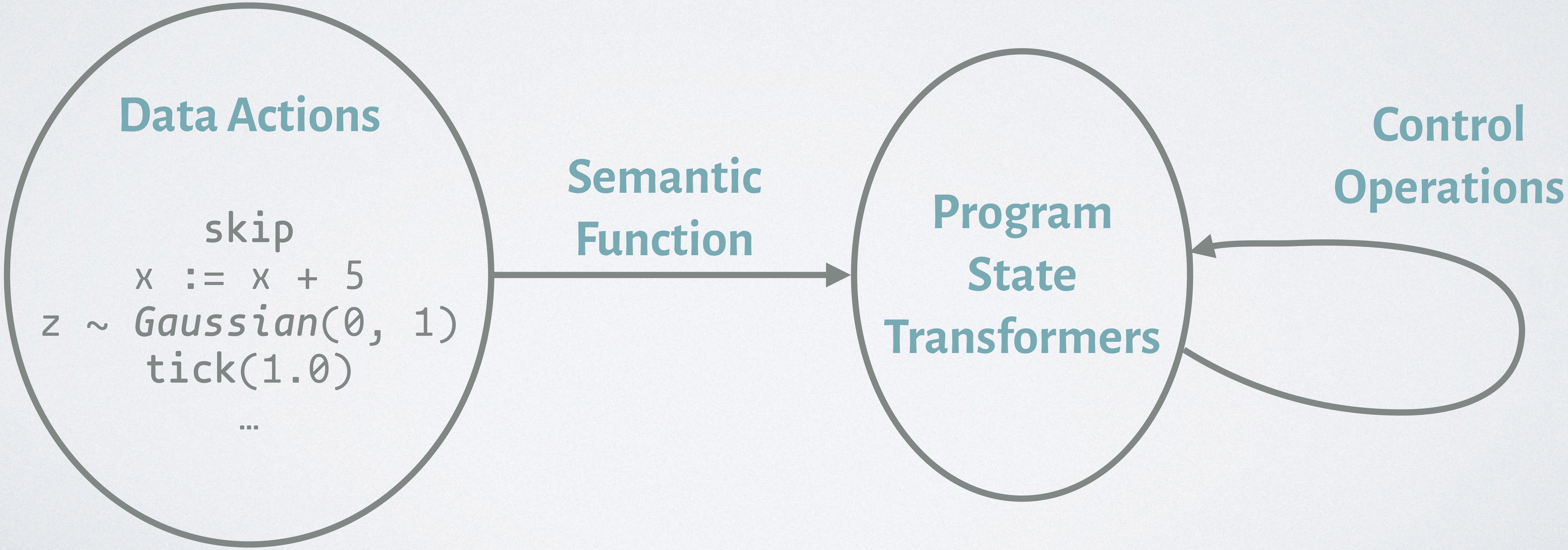
- Directed graphs with hyper-edges
- Branching are hyper-edges
- Support nondeterminism, unstructured control flow, and recursion




```
b1 ~ Bernoulli(0.5);
b2 ~ Bernoulli(0.7);
while (b1 && b2) do
  if prob(0.6) then
    b1 ~ Bernoulli(0.5)
  else
    b2 ~ Bernoulli(0.7)
  fi;
  tick(1.0)
od;
return (b1, b2)
```

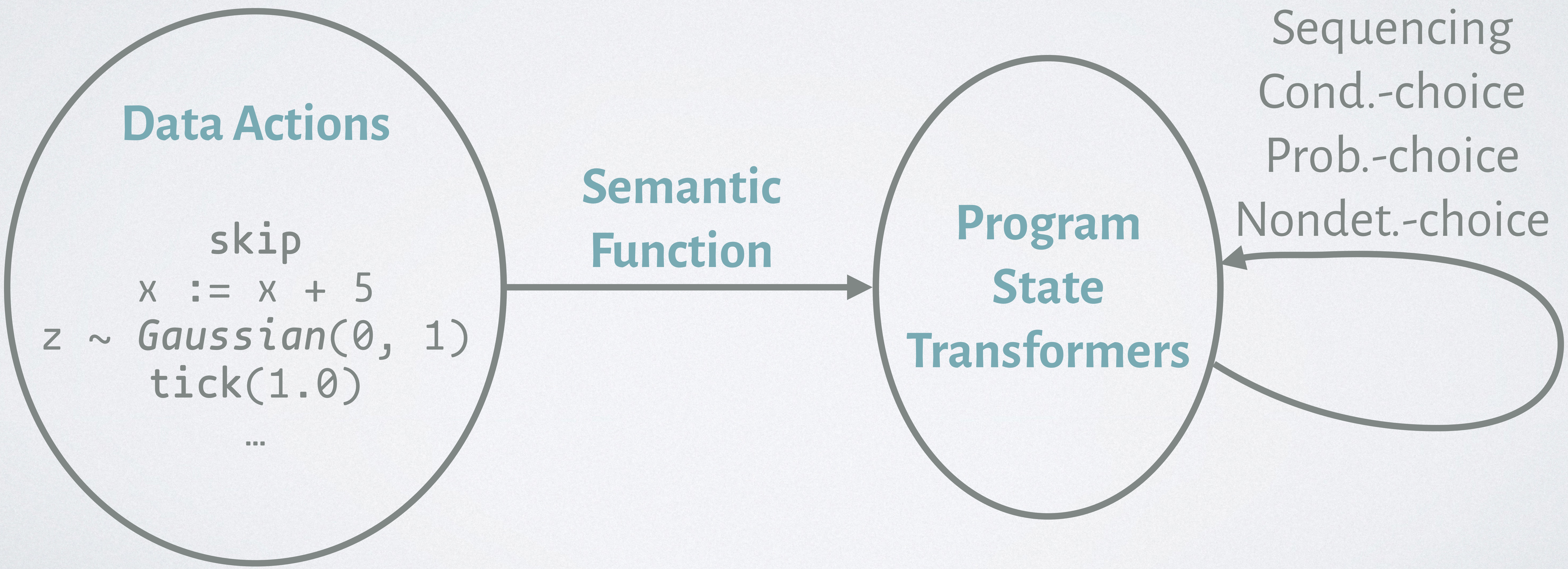
An Algebraic Denotational Semantics

- Perform reasoning in some abstract space of program-state transformers
- The transformers and associated operations obey some algebraic laws



An Algebraic Denotational Semantics

- Perform reasoning in some abstract space of program-state transformers
- The transformers and associated operations obey some algebraic laws



Markov Algebras

- Characterize state transformers and associated operations by algebraic laws

$$\left\langle M, \sqsubseteq, \otimes, {}_{\varphi}\Diamond, \uplus, \perp, 1 \right\rangle$$

Markov Algebras

- Characterize state transformers and associated operations by algebraic laws

$$\left\langle \underline{M, \sqsubseteq}, \otimes, {}_{\varphi}\Diamond, \uplus, \perp, 1 \right\rangle$$

Program state transformers form a complete partial order

Markov Algebras

- Characterize state transformers and associated operations by algebraic laws

$$\left\langle M, \sqsubseteq, \otimes, {}_{\varphi}\Diamond, \uplus, \bot, 1 \right\rangle$$

Program state transformers form a complete partial order

Sequencing, branching (cond. and prob.), and nondet.-choice

Markov Algebras

- Characterize state transformers and associated operations by algebraic laws

$$\left\langle M, \sqsubseteq, \otimes, {}_{\varphi}\Diamond, \uplus, \bot, 1 \right\rangle$$

Program state transformers form a complete partial order

Sequencing, branching (cond. and prob.), and nondet.-choice

The bottom element and the identity element

$\bot$ interprets `abort`

1 interprets `skip`

MARKOV ALGEBRAS

- Characterize state transformers and associated operations by algebraic laws

$$\left\langle \underbrace{M, \sqsubseteq}, \underbrace{\otimes, {}_{\varphi}\diamond, \uplus}, \bot, 1 \right\rangle$$

Program state transformers form a complete partial order

Sequencing, branching (cond. and prob.), and nondet.-choice

Markov Algebras

- Characterize state transformers and associated operations by algebraic laws

$$\left\langle M, \sqsubseteq, \otimes, {}_{\varphi}\Diamond, \uplus, \bot, 1 \right\rangle$$

Program state transformers form a complete partial order

Sequencing, branching (cond. and prob.), and nondet.-choice

$$(a \otimes b) \otimes c = a \otimes (b \otimes c)$$

$$a \otimes 1 = 1 \otimes a$$

$$a \ {}_{\varphi}\Diamond\ b = b \ {}_{\neg\varphi}\Diamond\ a$$

$$a \uplus a = a$$

$$\otimes,\ {}_{\varphi}\Diamond,\ \uplus \text{ continuous w.r.t. } \sqsubseteq$$

$$\ldots$$

A New Model for Resolving Nondeterminism

A New Model for Resolving Nondeterminism

- We want to resolve nondeterminacy among state transformers instead of states

```
if prob(*) then t := t + 1 else t := t - 1 fi
```

A New Model for Resolving Nondeterminism

- We want to resolve nondeterminacy among state transformers instead of states

```
if prob(*) then t := t + 1 else t := t - 1 fi
```

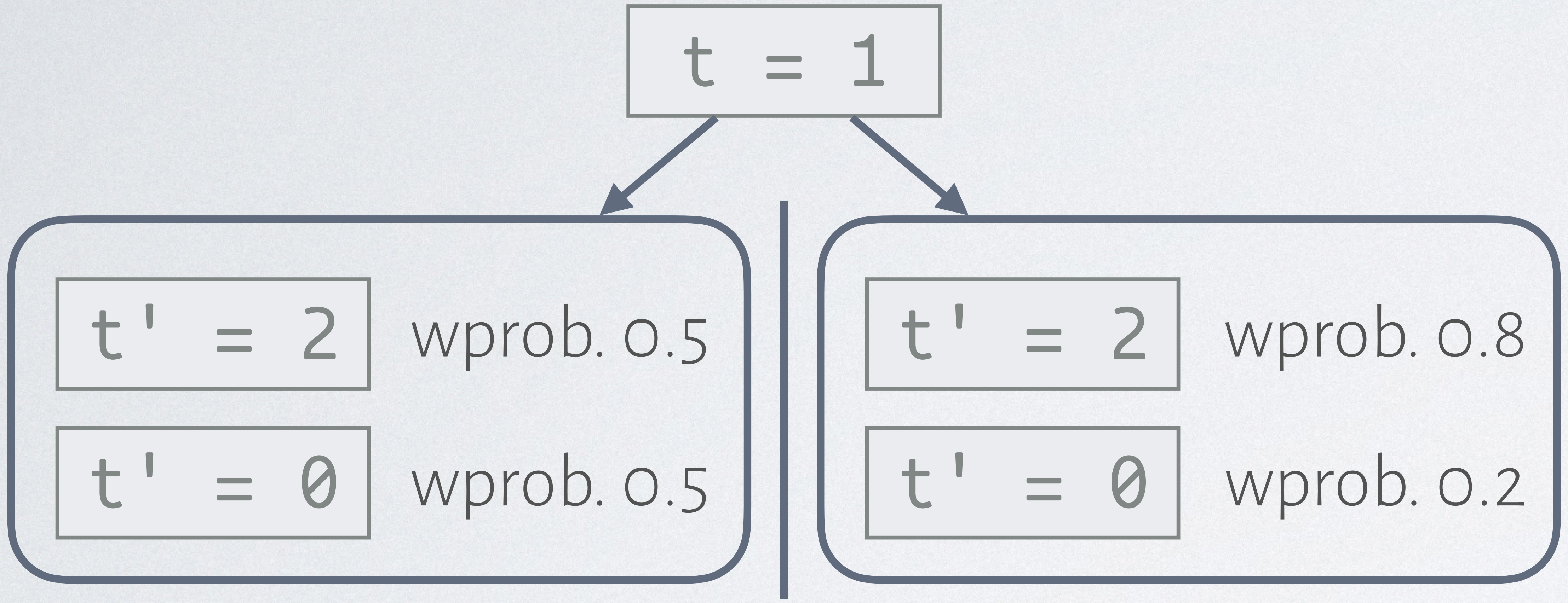
```
t = 1
```

* resolved **after** `t` is given

A New Model for Resolving Nondeterminism

- We want to resolve nondeterminacy among state transformers instead of states

```
if prob(*) then t := t + 1 else t := t - 1 fi
```

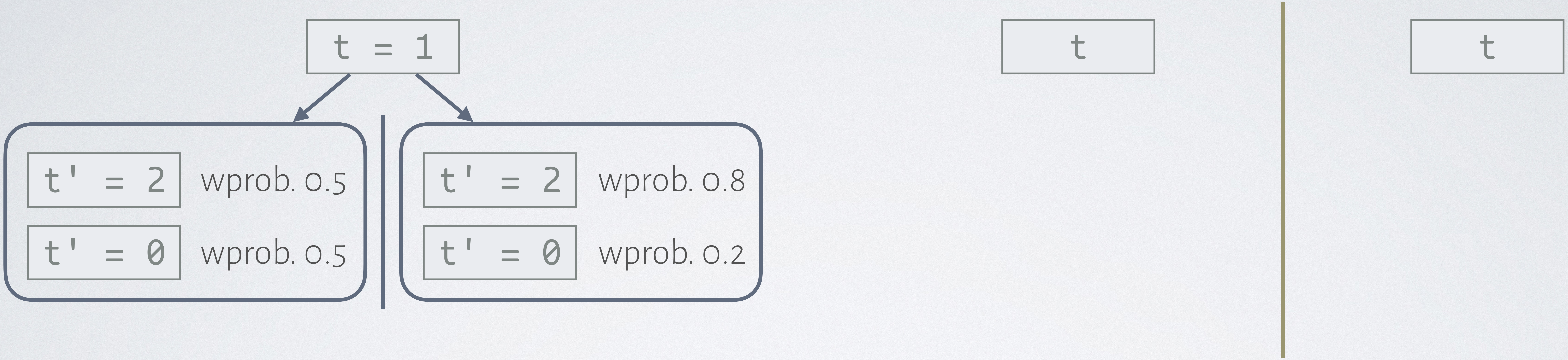


* resolved **after** `t` is given

A New Model for Resolving Nondeterminism

- We want to resolve nondeterminacy **among state transformers** instead of states

```
if prob(⋆) then t := t + 1 else t := t - 1 fi
```



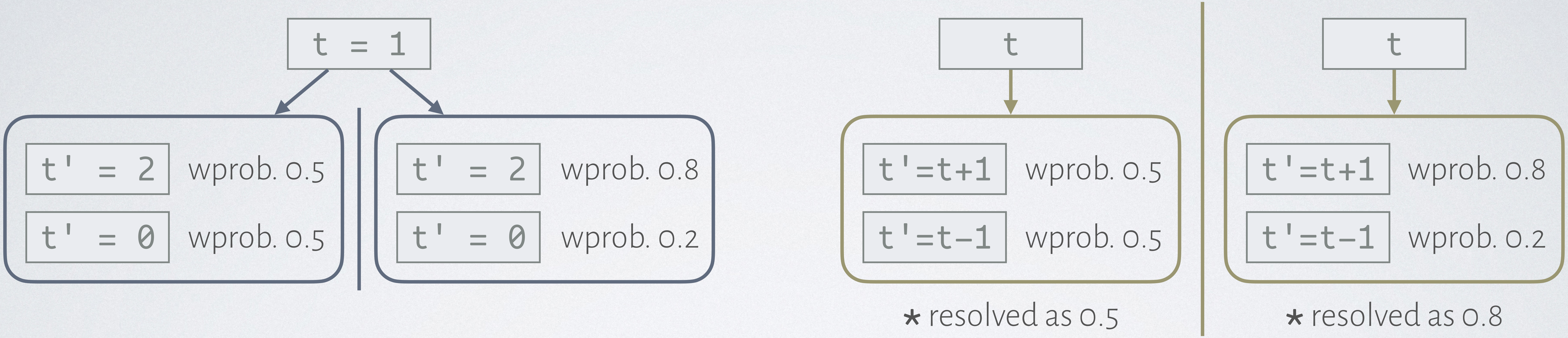
⋆ resolved **after** t is given

⋆ resolved **before** t is given

A New Model for Resolving Nondeterminism

- We want to resolve nondeterminacy **among state transformers** instead of states

```
if prob(*) then t := t + 1 else t := t - 1 fi
```



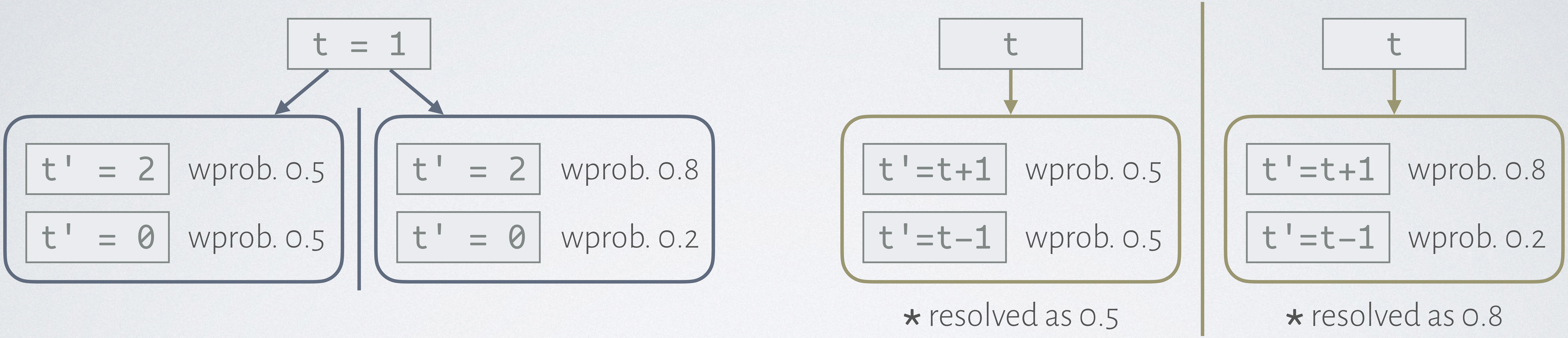
* resolved **after** t is given

* resolved **before** t is given

A New Model for Resolving Nondeterminism

- We want to resolve nondeterminacy **among state transformers** instead of states

```
if prob(*) then t := t + 1 else t := t - 1 fi
```



⋆ resolved **after** t is given

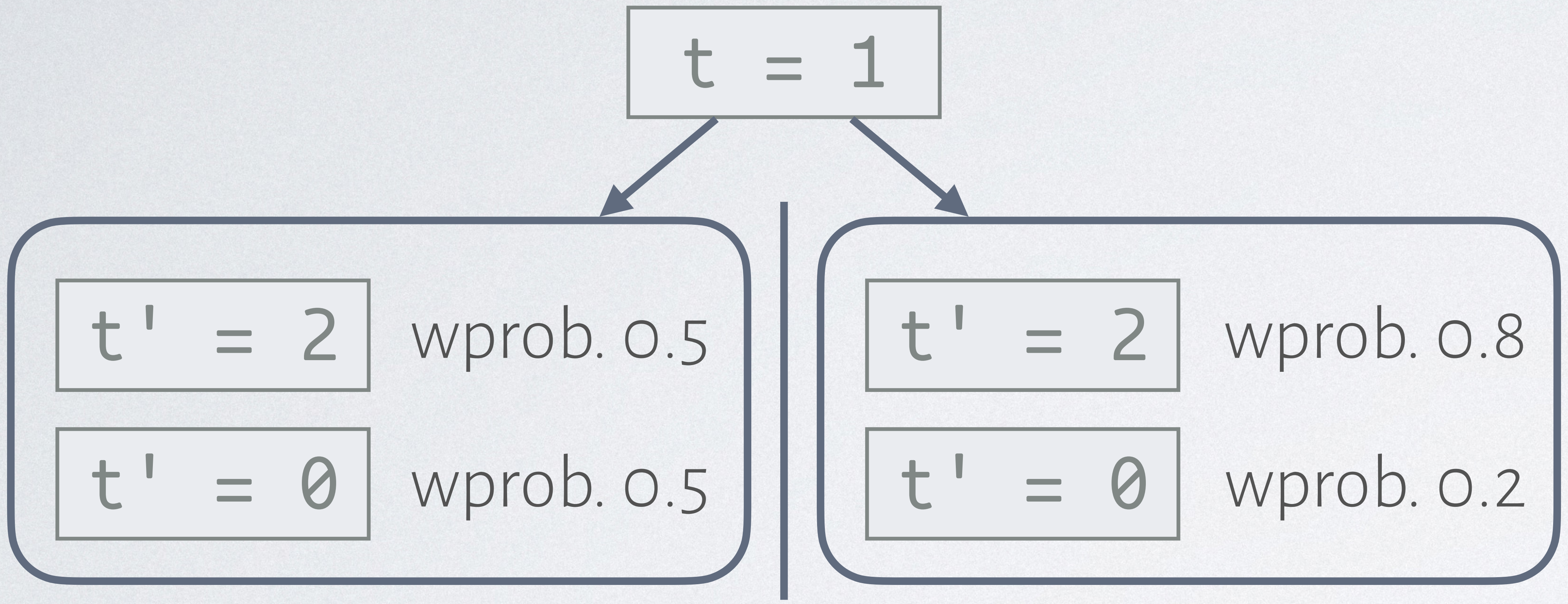
⋆ resolved **before** t is given

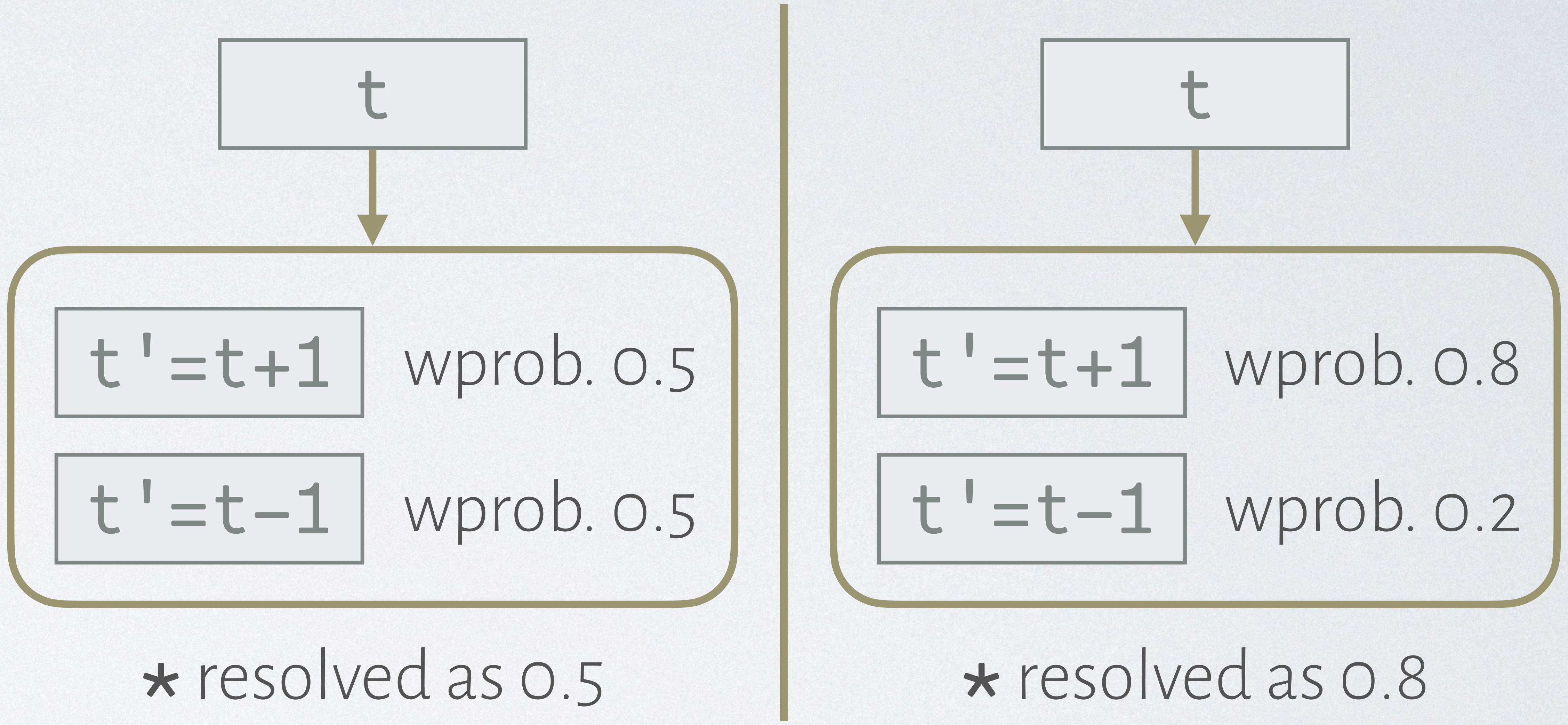
$$M := \text{State} \to \wp(\text{Dist}(\text{State}))$$

A New Model for Resolving Nondeterminism

- We want to resolve nondeterminacy **among state transformers** instead of states

```
if prob(*) then t := t + 1 else t := t - 1 fi
```





⋆ resolved **after** t is given

$$M := \mathrm{State} \to \wp(\mathrm{Dist}(\mathrm{State}))$$

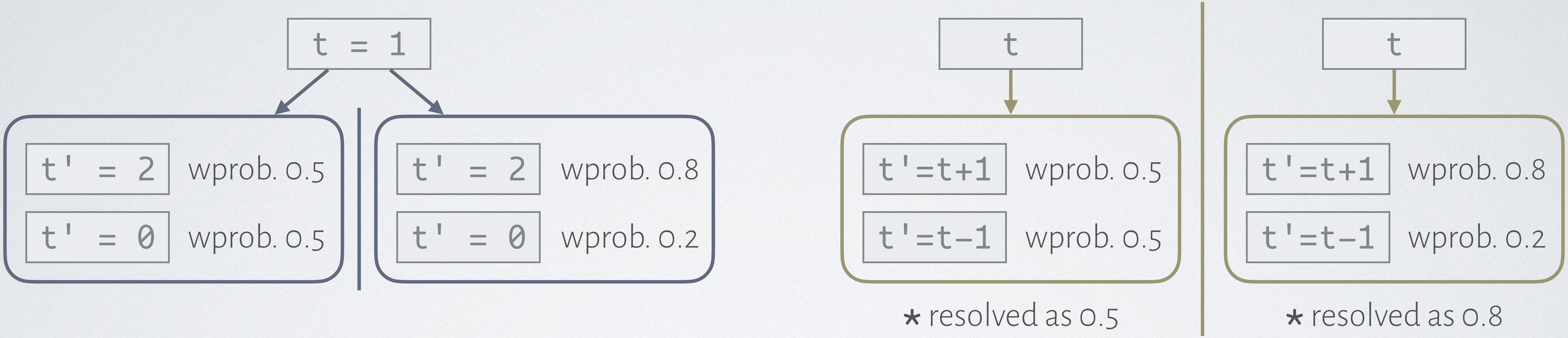
⋆ resolved **before** t is given

$$M := \wp(\mathrm{State} \to \mathrm{Dist}(\mathrm{State}))$$

A New Model for Resolving Nondeterminism

- We want to resolve nondeterminacy among state transformers instead of states

```
if prob(*) then t := t + 1 else t := t - 1 fi
```



⋆ resolved **after** t is given

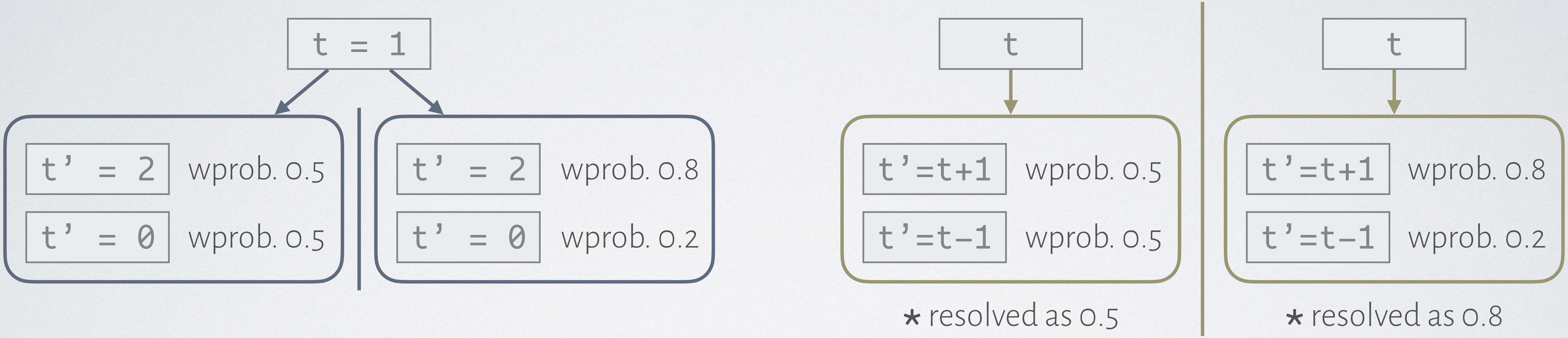
⋆ resolved **before** t is given

- Relational reasoning: for any concretization P of the program, for any inputs t_1, t_2, it holds that $\mathbb{E}_{t_1' \sim P(t_1), t_2' \sim P(t_2)}[t_1' - t_2'] = t_1 - t_2$

A New Model for Resolving Nondeterminism

- We want to resolve nondeterminacy among state transformers instead of states

```
if prob(*) then t := t + 1 else t := t - 1 fi
```



* resolved **after** t is given

* resolved **before** t is given

- We developed a domain-theoretic characterization of the nondeterminism-first resolution

Overview

- ☑ Motivation
- ☑ An Algebraic Denotational Semantics
- ☐ Pre-Markov Algebra Framework (PMAF)

Pre-Markov Algebra Framework (PMAF)

Contributions

- An **algebraic framework** for **interprocedural dataflow analysis** of **first-order probabilistic programs**
- Published as PMAF: An Algebraic Framework for Static Analysis of Probabilistic Programs in *PLDI'18*

PMAF

Recursion
Unstructured control-flow
Divergence
Nondeterminism
…

Pre-Markov Algebra Framework (PMAF)

Contributions

- An **algebraic framework** for **interprocedural dataflow analysis** of **first-order probabilistic programs**
- Published as PMAF: An Algebraic Framework for Static Analysis of Probabilistic Programs in *PLDI'18*

PMAF

Pre-Markov Algebra Framework (PMAF)

Contributions

- An **algebraic framework** for **interprocedural dataflow analysis** of **first-order probabilistic programs**
- Published as PMAF: An Algebraic Framework for Static Analysis of Probabilistic Programs in *PLDI'18*

PMAF

Pre-Markov Algebra Framework (PMAF)

Contributions

- An **algebraic framework** for **interprocedural dataflow analysis** of **first-order probabilistic programs**
- Published as PMAF: An Algebraic Framework for Static Analysis of Probabilistic Programs in *PLDI'18*

PMAF

Pre-Markov Algebra Framework (PMAF)

Contributions

- An **algebraic framework** for **interprocedural dataflow analysis** of **first-order probabilistic programs**
- Published as PMAF: An Algebraic Framework for Static Analysis of Probabilistic Programs in *PLDI'18*

PMAF

Existing

- Reachability Prob. Analysis
- Markov Decision Problem

Pre-Markov Algebra Framework (PMAF)

Contributions

- An **algebraic framework** for **interprocedural dataflow analysis** of **first-order probabilistic programs**
- Published as PMAF: An Algebraic Framework for Static Analysis of Probabilistic Programs in *PLDI'18*

PMAF

Existing

- Reachability Prob. Analysis
- Markov Decision Problem

New

- Expectation-Invariant Analysis

Why not Probabilistic Abstract Interpretation?

⋆ denotes nondeterministic-choice

`tick(q)` increases T by `q`

```
if ⋆ then
  if prob(0.5) then tick(1.0) else tick(2.0) fi
else
  if prob(0.5) then tick(1.0) else tick(2.0) fi
fi
```

Why not Probabilistic Abstract Interpretation?

⋆ denotes nondeterministic-choice

`tick(q)` increases T by `q`

```
if ⋆ then
  if prob(0.5) then tick(1.0) else tick(2.0) fi
else
  if prob(0.5) then tick(1.0) else tick(2.0) fi
fi
```

with prob. $\frac{1}{4}$

Why not Probabilistic Abstract Interpretation?

⋆ denotes nondeterministic-choice

`tick(q)` increases T by `q`

```
if ⋆ then
  if prob(0.5) then tick(1.0) else tick(2.0) fi
else
  if prob(0.5) then tick(1.0) else tick(2.0) fi
fi
```

Why not Probabilistic Abstract Interpretation?

$\star$ denotes nondeterministic-choice

`tick(q)` increases T by `q`

```
if * then
  if prob(0.5) then tick(1.0) else tick(2.0) fi
else
  if prob(0.5) then tick(1.0) else tick(2.0) fi
fi
```

with prob. $\frac{1}{4}$

Why not Probabilistic Abstract Interpretation?

⋆ denotes nondeterministic-choice

`tick(q)` increases T by `q`

```
if ⋆ then
  if prob(0.5) then tick(1.0) else tick(2.0) fi
else
  if prob(0.5) then tick(1.0) else tick(2.0) fi
fi
```

Why not Probabilistic Abstract Interpretation?

⋆ denotes nondeterministic-choice

`tick(q)` increases T by `q`

```
if ⋆ then
  if prob(0.5) then tick(1.0) else tick(2.0) fi
else
  if prob(0.5) then tick(1.0) else tick(2.0) fi
fi
```

with prob. $\frac{1}{4}$

Why not Probabilistic Abstract Interpretation?

⋆ denotes nondeterministic-choice

`tick(q)` increases T by `q`

```
if ⋆ then
  if prob(0.5) then tick(1.0) else tick(2.0) fi
else
  if prob(0.5) then tick(1.0) else tick(2.0) fi
fi
```

WHY NOT PROBABILISTIC ABSTRACT INTERPRETATION?

* denotes nondeterministic-choice

`tick(q)` increases T by `q`

```
if * then
  if prob(0.5) then tick(1.0) else tick(2.0) fi
else
  if prob(0.5) then tick(1.0) else tick(2.0) fi
fi
```

with prob. $\frac{1}{4}$

Why not Probabilistic Abstract Interpretation?

⋆ denotes nondeterministic-choice

`tick(q)` increases T by `q`

```
if ⋆ then
  if prob(0.5) then tick(1.0) else tick(2.0) fi
else
  if prob(0.5) then tick(1.0) else tick(2.0) fi
fi
```

Why not Probabilistic Abstract Interpretation?

⋆ denotes nondeterministic-choice

`tick(q)` increases T by `q`

```
if ⋆ then
  if prob(0.5) then tick(1.0) else tick(2.0) fi
else
  if prob(0.5) then tick(1.0) else tick(2.0) fi
fi
```

- Their concrete semantics yields

$$\mathbb{E}[T] \in \frac{1}{4} \cdot \{1\} + \frac{1}{4} \cdot \{2\} + \frac{1}{4} \cdot \{1,2\} + \frac{1}{4} \cdot \{1,2\} = \{1.25, 1.5, 1.75\}$$

Why not Probabilistic Abstract Interpretation?

⋆ denotes nondeterministic-choice

`tick(q)` increases T by `q`

```
if ⋆ then
  if prob(0.5) then tick(1.0) else tick(2.0) fi
else
  if prob(0.5) then tick(1.0) else tick(2.0) fi
fi
```

Identical!

- Their concrete semantics yields

$$\mathbb{E}[T] \in \frac{1}{4} \cdot \{1\} + \frac{1}{4} \cdot \{2\} + \frac{1}{4} \cdot \{1,2\} + \frac{1}{4} \cdot \{1,2\} = \{1.25, 1.5, 1.75\}$$

Why not Probabilistic Abstract Interpretation?

⋆ denotes nondeterministic-choice

`tick(q)` increases T by `q`

```
if ⋆ then
  if prob(0.5) then tick(1.0) else tick(2.0) fi
else
  if prob(0.5) then tick(1.0) else tick(2.0) fi
fi
```

Identical!

- Their concrete semantics yields

$$\mathbb{E}[T] \in \frac{1}{4} \cdot \{1\} + \frac{1}{4} \cdot \{2\} + \frac{1}{4} \cdot \{1,2\} + \frac{1}{4} \cdot \{1,2\} = \{1.25, 1.5, 1.75\}$$

whereas our semantics yields $\mathbb{E}[T] = 1.5$

Example Analyses

```
b1 ~ Bernoulli(0.5);
b2 ~ Bernoulli(0.7);
while (b1 && b2) do
  if prob(0.6) then
    b1 ~ Bernoulli(0.5)
  else
    b2 ~ Bernoulli(0.7)
  fi;
  tick(1.0)
od;
return (b1, b2)
```

Example Analyses

- Our framework can be instantiated to prove:

```
b1 ~ Bernoulli(0.5);
b2 ~ Bernoulli(0.7);
while (b1 && b2) do
  if prob(0.6) then
    b1 ~ Bernoulli(0.5)
  else
    b2 ~ Bernoulli(0.7)
  fi;
  tick(1.0)
od;
return (b1, b2)
```

Example Analyses

- Our framework can be instantiated to prove:

- the probability that `b1` and `b2` are both `false` at the end of the program = 0.15

```
b1 ~ Bernoulli(0.5);
b2 ~ Bernoulli(0.7);
while (b1 && b2) do
  if prob(0.6) then
    b1 ~ Bernoulli(0.5)
  else
    b2 ~ Bernoulli(0.7)
  fi;
  tick(1.0)
od;
return (b1, b2)
```

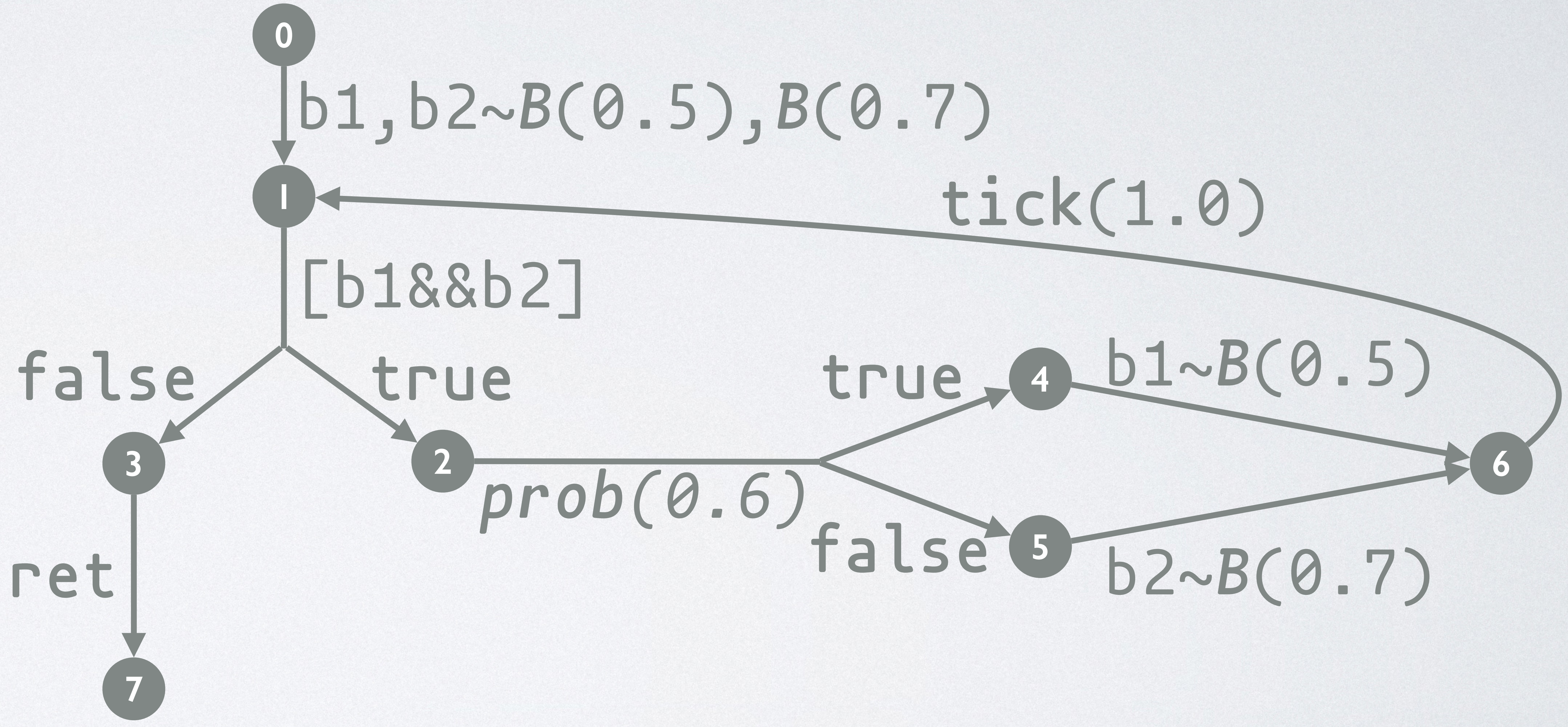
Example Analyses

- Our framework can be instantiated to prove:
- the probability that `b1` and `b2` are both `false` at the end of the program = 0.15
- the expected termination time (ticks) = 5/6

```
b1 ~ Bernoulli(0.5);
b2 ~ Bernoulli(0.7);
while (b1 && b2) do
  if prob(0.6) then
    b1 ~ Bernoulli(0.5)
  else
    b2 ~ Bernoulli(0.7)
  fi;
  tick(1.0)
od;
return (b1, b2)
```

Hyper-Graph Semantics

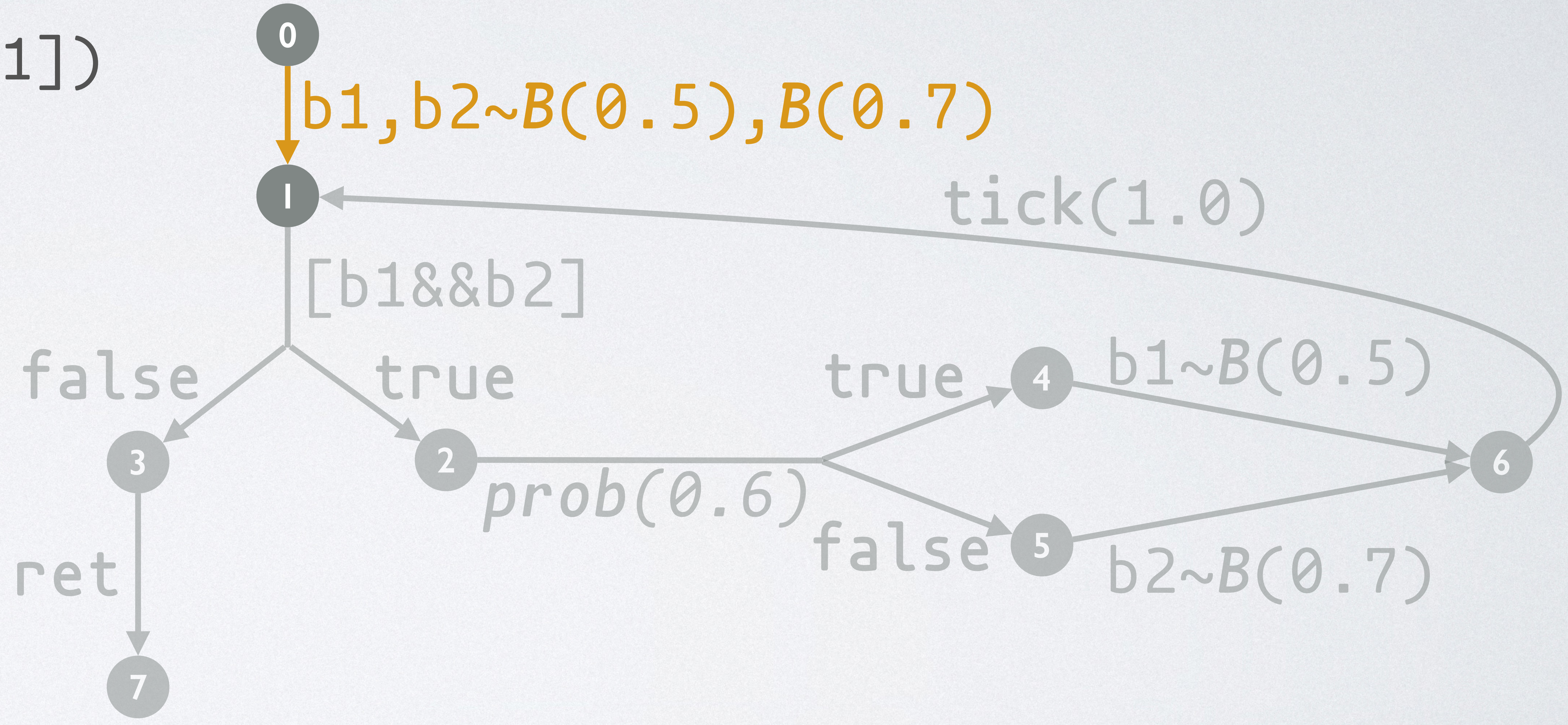
- The hyper-graph semantics is formulated by an equation system



Hyper-Graph Semantics

- The hyper-graph semantics is formulated by an equation system

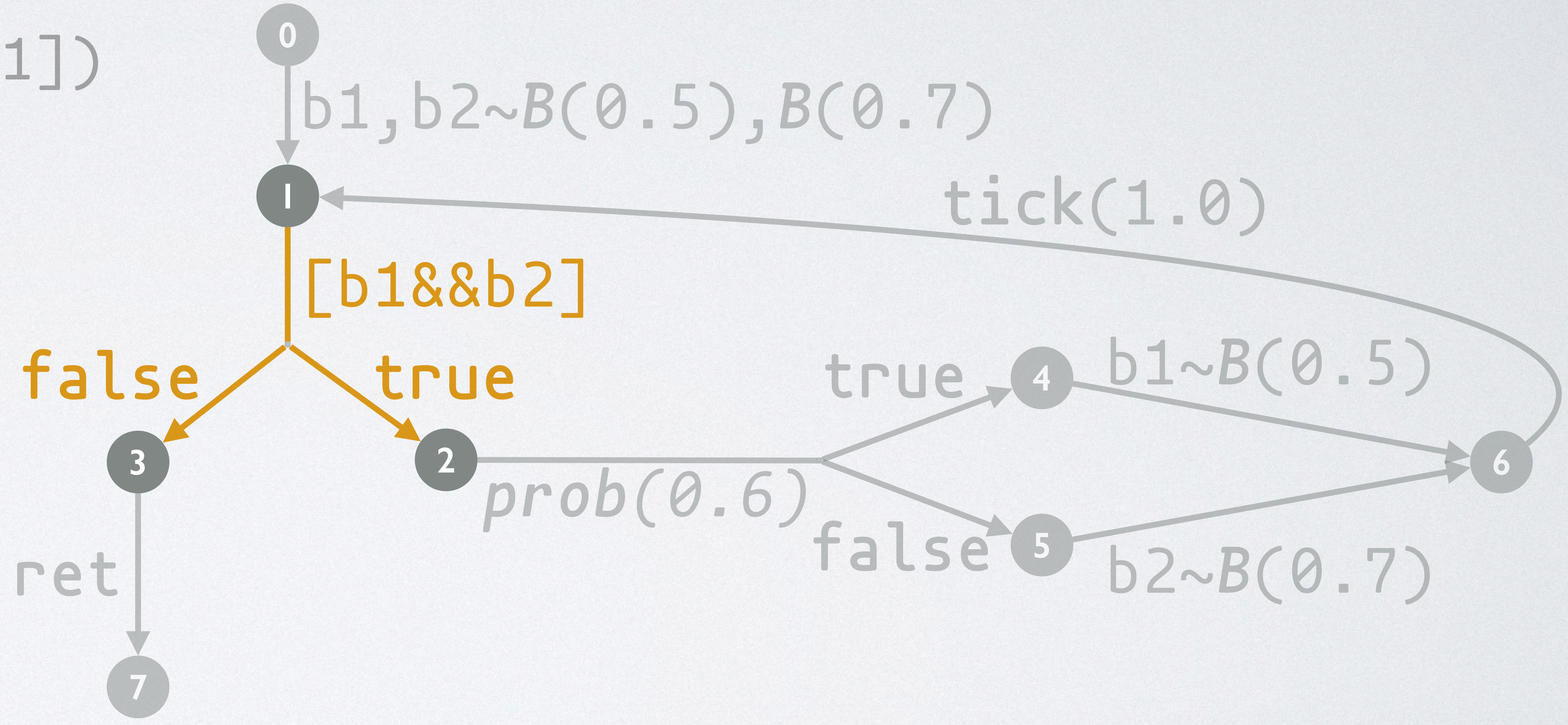
```
S[0]=seq[b1,b2~B(0.5),B(0.7)](S[1])
```



Hyper-Graph Semantics

- The hyper-graph semantics is formulated by an equation system

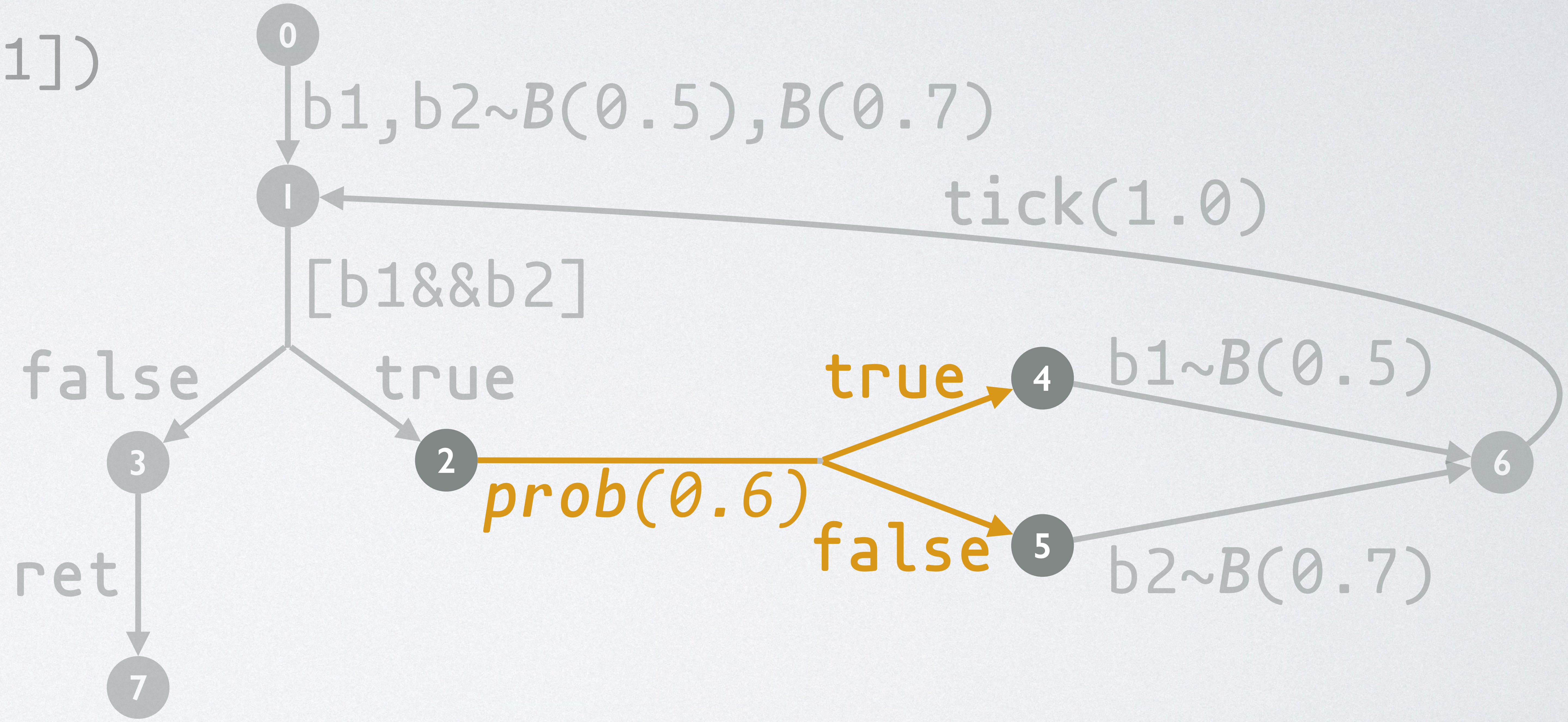
```
S[0]=seq[b1,b2~B(0.5),B(0.7)](S[1])
S[1]=cond[b1&&b2](S[2],S[3])
```



Hyper-Graph Semantics

- The hyper-graph semantics is formulated by an equation system

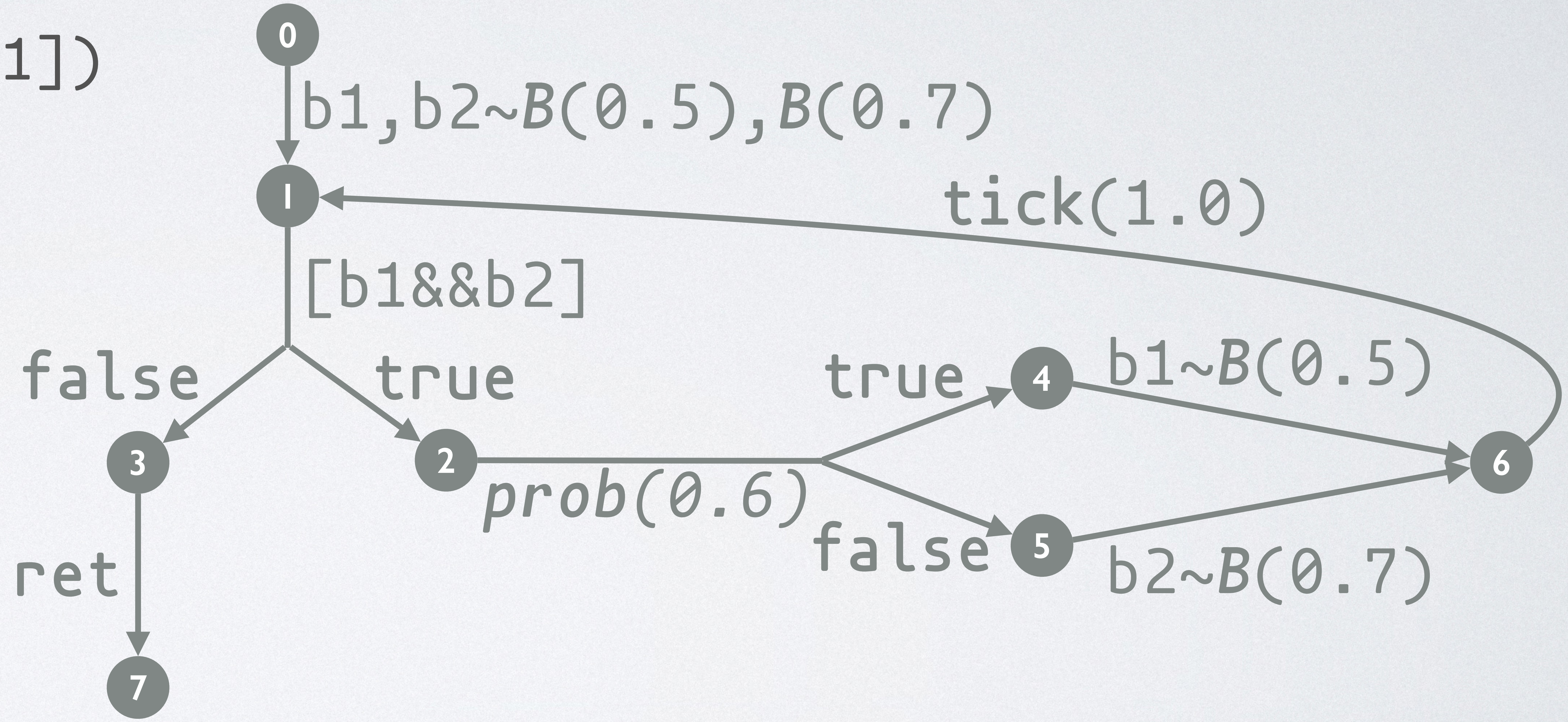
```
S[0]=seq[b1,b2~B(0.5),B(0.7)](S[1])
S[1]=cond[b1&&b2](S[2],S[3])
S[2]=prob[0.6](S[4],S[5])
```



HYPER-GRAPH SEMANTICS

- The hyper-graph semantics is formulated by an equation system

```
S[0]=seq[b1,b2~B(0.5),B(0.7)](S[1])
S[1]=cond[b1&&b2](S[2],S[3])
S[2]=prob[0.6](S[4],S[5])
```

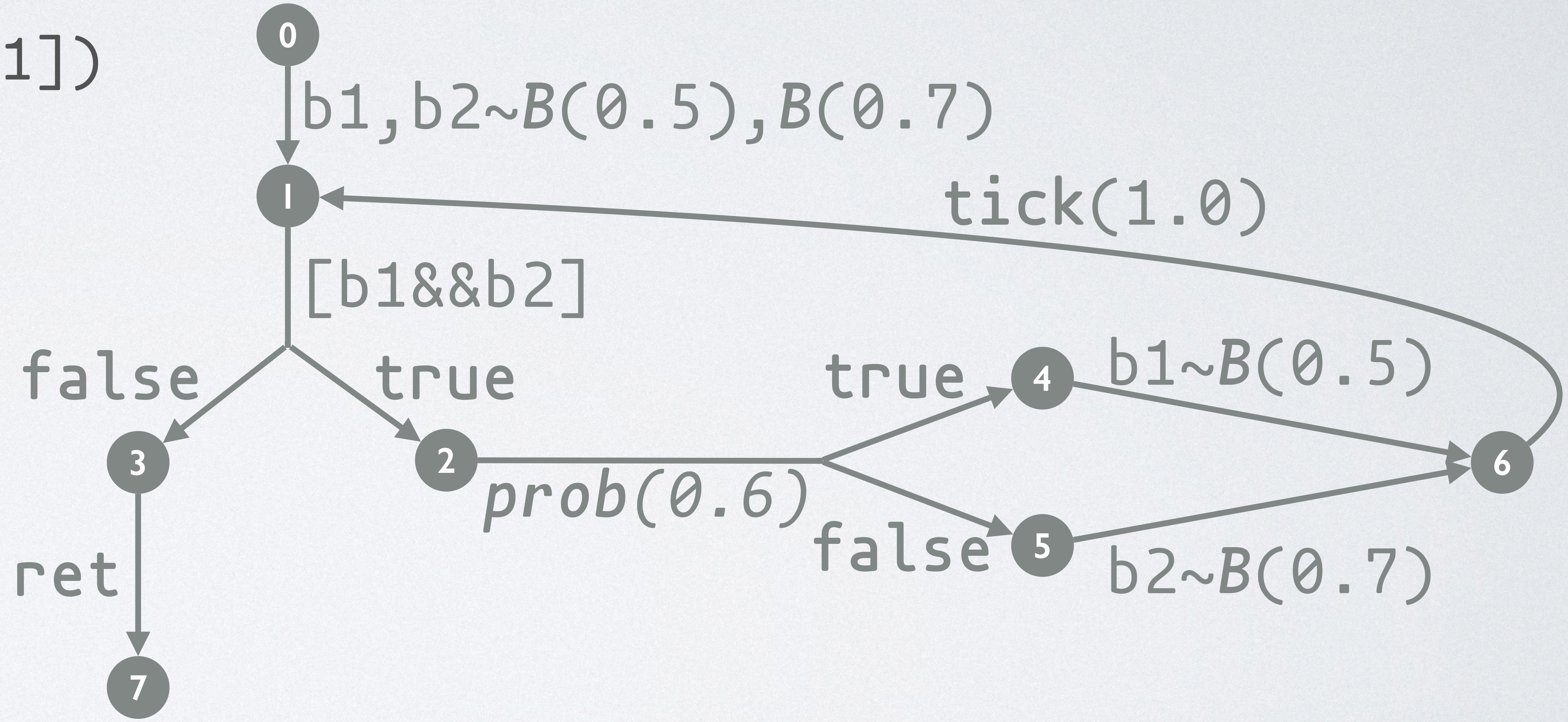


Hyper-Graph Semantics

- The hyper-graph semantics is formulated by an equation system

```
S[0]=seq[b1,b2~B(0.5),B(0.7)](S[1])
S[1]=cond[b1&&b2](S[2],S[3])
S[2]=prob[0.6](S[4],S[5])
```

Use a **Markov algebra** to interpret ***seq***, ***cond***, ***prob***



Hyper-Graph Semantics

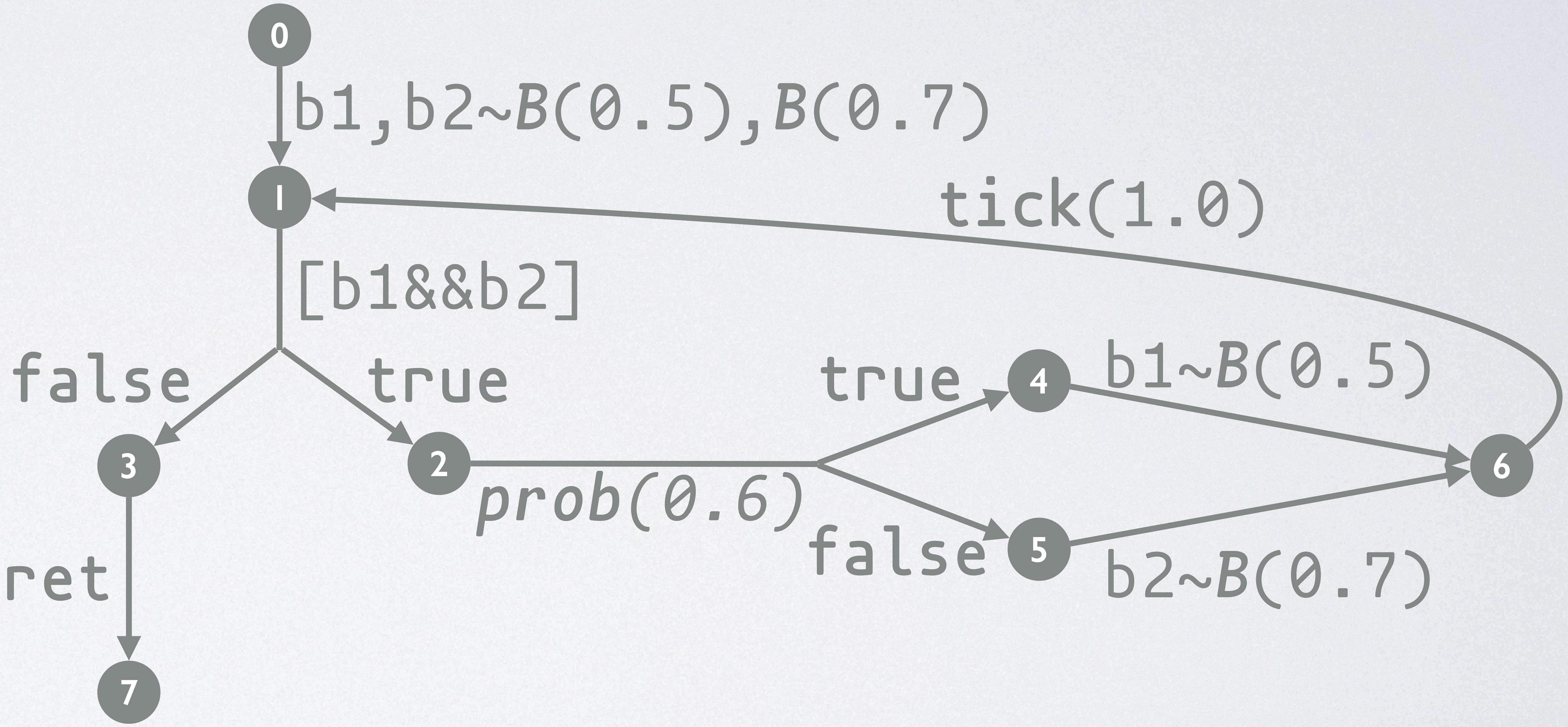
- The hyper-graph semantics is formulated by an equation system

$S[0] = [\mathrm{b1}, \mathrm{b2} \sim \mathrm{B}(0.5), \mathrm{B}(0.7)] \otimes S[1]$

$S[1] = S[2]_{\mathrm{b1\&\&b2}} \diamond S[3]$

$S[2] = S[4]_{0.6} \oplus S[5]$

Use a **Markov algebra** to interpret ***seq***, ***cond***, ***prob***



HYPER-GRAPH SEMANTICS

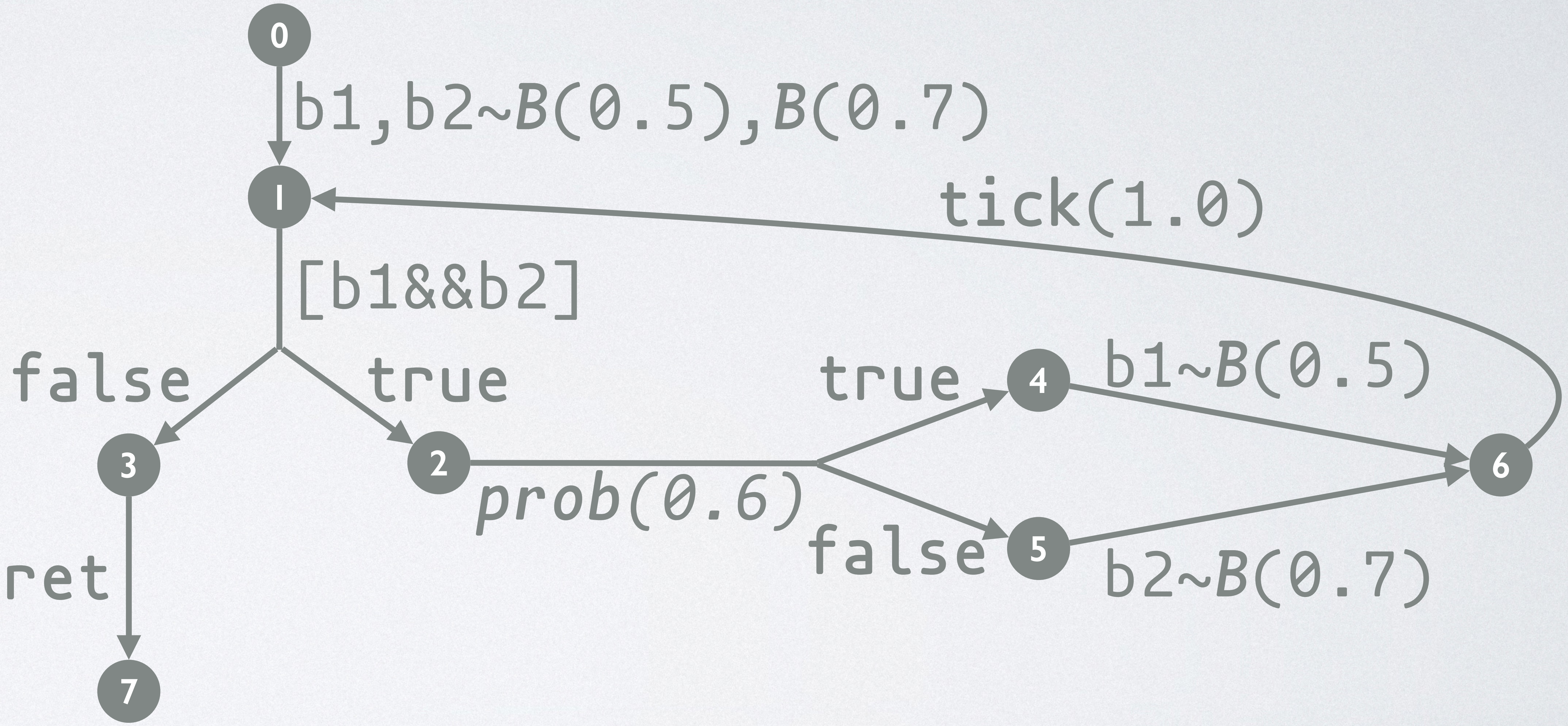
- The hyper-graph semantics is formulated by an equation system

$S[0] = [\mathrm{b1}, \mathrm{b2} \sim \mathrm{B}(0.5), \mathrm{B}(0.7)] \otimes S[1]$

$S[1] = S[2]_{\mathrm{b1\&\&b2}} \diamond S[3]$

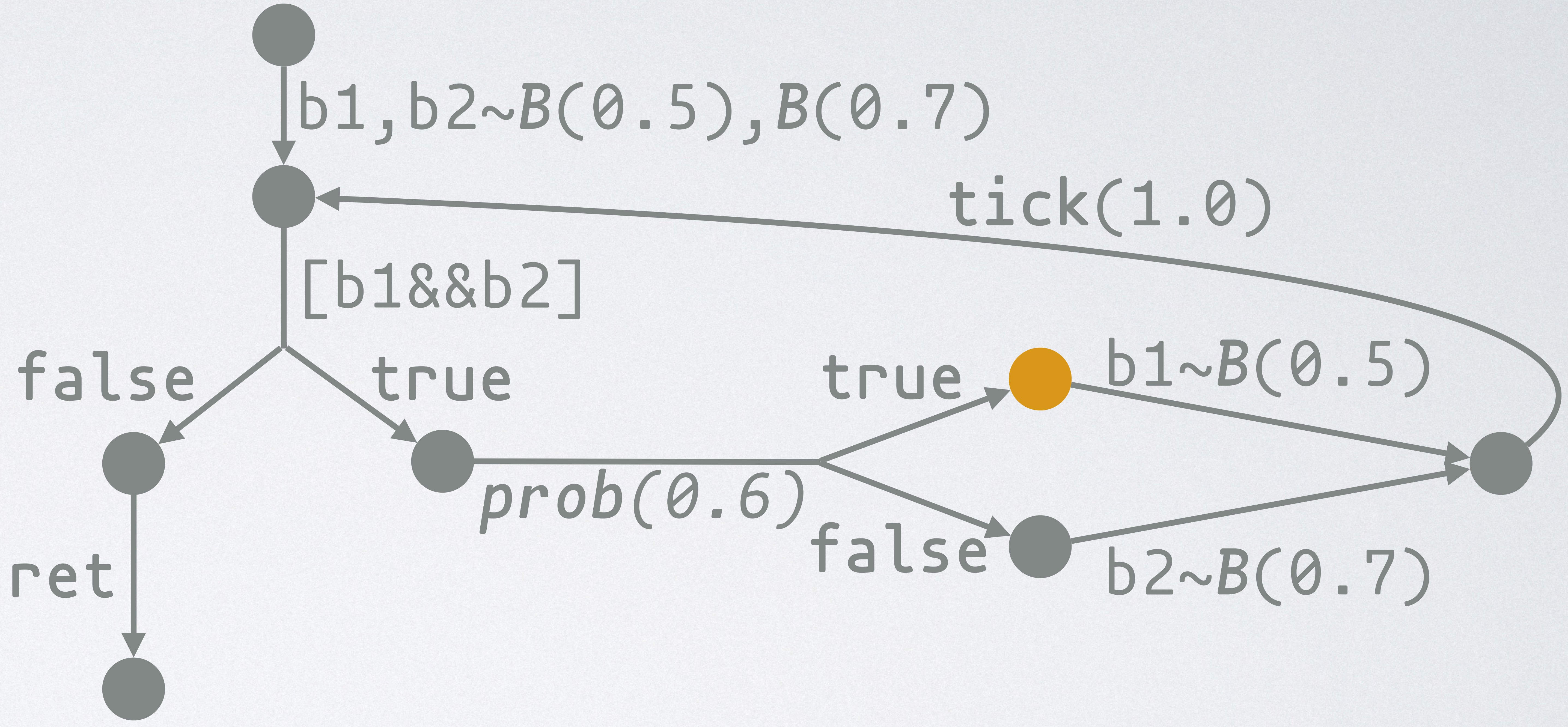
$S[2] = S[4]_{0.6} \oplus S[5]$

If using **abstract** semantics, we obtain an equation system for **static analysis**



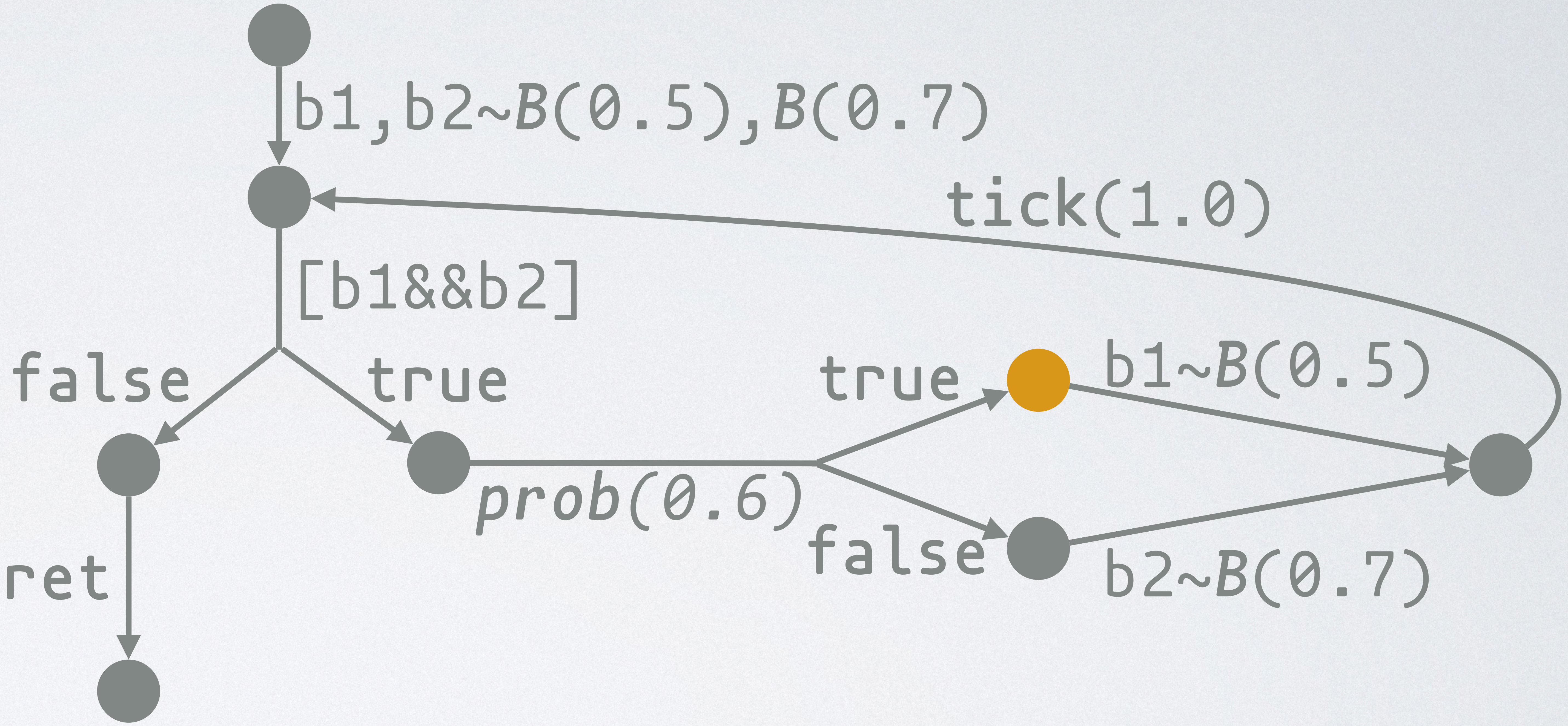
Hyper-Graph Analysis

- **Forward assertions**
- The property of a node is a summary of the computation that continues from the node



Hyper-Graph Analysis

- **Forward assertions**
- The property of a node is a summary of the computation that continues from the node



e.g. the property represented by the node is

$$\lambda(\mathsf{b1},\mathsf{b2})\,.\,\textbf{if}\ \mathsf{b2}\ \textbf{then}\ \frac{1}{7}[\mathsf{b1}'=\mathsf{T},\mathsf{b2}'=\mathsf{F}]+\frac{6}{7}[\mathsf{b1}'=\mathsf{F},\mathsf{b2}'=\mathsf{T}]\ \textbf{else}\ \frac{1}{2}[\mathsf{b1}'=\mathsf{T},\mathsf{b2}'=\mathsf{F}]+\frac{1}{2}[\mathsf{b1}'=\mathsf{F},\mathsf{b2}'=\mathsf{F}]$$

Pre-Markov Algebras (PMA)

- A PMA is basically a Markov algebra, but we assume a different set of axioms that are suitable for formulating static analysis

$$\left\langle M, \sqsubseteq, \otimes, {}_{\varphi}\Diamond, {}_{p}\oplus, \uplus, \bot, 1 \right\rangle$$

Pre-Markov Algebras (PMA)

- A PMA is basically a Markov algebra, but we assume a different set of axioms that are suitable for formulating static analysis

$$\left\langle \underline{M, \sqsubseteq}, \otimes, {}_{\varphi}\Diamond, {}_{p}\oplus, \uplus, \bot, 1 \right\rangle$$

Program properties form a complete lattice

PRE-MARKOV ALGEBRAS (PMA)

- A PMA is basically a Markov algebra, but we assume a different set of axioms that are suitable for formulating static analysis

$$\left\langle \underline{M, \sqsubseteq}, \underline{\otimes, {}_{\varphi}\Diamond, {}_{p}\oplus, \uplus}, \bot, 1 \right\rangle$$

Program properties form a complete lattice

Sequencing, cond.-choice, prob.-choice, and nondet.-choice are monotone operations

Pre-Markov Algebras (PMA)

- A PMA is basically a Markov algebra, but we assume a different set of axioms that are suitable for formulating static analysis

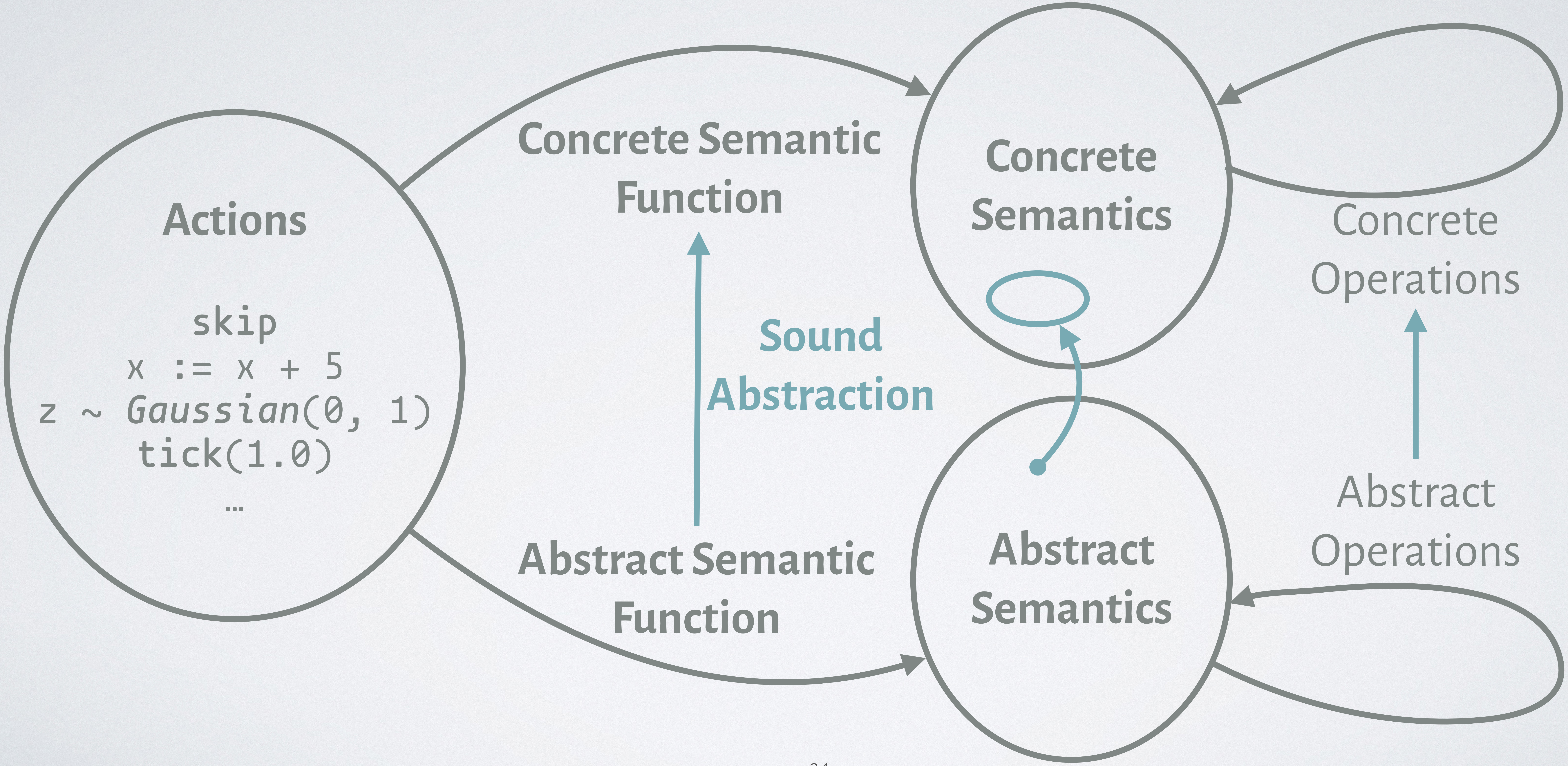
$$\left\langle \underline{M, \sqsubseteq}, \underline{\otimes, {}_{\varphi}\Diamond, {}_{p}\oplus, \uplus}, \underline{\bot, 1} \right\rangle$$

Program properties form a complete lattice

Sequencing, cond.-choice, prob.-choice, and nondet.-choice are monotone operations

The bottom element and the identity element

Soundness via Abstract Interpretation



INSTANTIATIONS

PMAF

INSTANTIATIONS

PMAF

Existing

- Reachability Prob. Analysis
- Markov Decision Problem

INSTANTIATIONS

PMAF

Existing

- Reachability Prob. Analysis
- Markov Decision Problem

New

Interprocedural analyzers!

Instantiations

Existing

PMAF

- Reachability Prob. Analysis
- Markov Decision Problem

Instantiations

PMAF

Existing

- Reachability Prob. Analysis
- Markov Decision Problem

New

- Expectation-Invariant Analysis

Instantiations

PMAF

Existing

- Reachability Prob. Analysis
- Markov Decision Problem

New

- Expectation-Invariant Analysis

Prove invariants among **initial values** and **expected final values**

Expectation-Invariant Analysis

- Benchmark collected from the literature[7,8] and also handcrafted by us
- Derive expectation invariants as least as precise as them in most case

Expectation-Invariant Analysis

Program	#loc	time (sec)	Expectation Invariants
binom-update	14	0.06	E[4x'-n']=4x-n, E[x']<=x+1/4
eg	8	0.89	E[x'+y']=x+y+4, E[z']=1/4z+3/4
recursive	13	0.37	E[x']=x+9
mot-ex	16	0.06	E[2x'-y']=2x-y, E[4x'-3c']=4x-3c, E[x']<=x+3/4

[7] A. Chakarov and S. Sankaranarayanan. Expectation Invariants for Probabilistic Loops as Fixed Points. In *SAS'14*.

[8] J.-P. Katoen, A. K. McIver, L. A. Meinicke, and C. C. Morgan. Linear-Invariant Generation for Probabilistic Programs. In *SAS'10*.