Introduction to

Dynamic Programming for Generative Models

Andreas Stuhlmüller

IPAM GSS 2011

Overview

- Motivation
  - A simple coordination game
  - Blue-eyed islanders puzzle
  - Hidden Markov models
- DP for HMMs: Forward and Viterbi algorithm
- DP for probabilistic programs: UDP algorithm

A simple coordination game

```
(define (sample-location)
  (if (flip .55)
      'popular-bar
      'unpopular-bar))
```

```
(define (bob)
  (sample-location))
```

(Schelling 1960)

A simple coordination game

```
(define (sample-location)
  (if (flip .55)
      'popular-bar
      'unpopular-bar))

(define (alice)
  (query
   (define alice-location (sample-location))
   alice-location
   (equal? alice-location (bob))))

(define (bob)
  (sample-location))
```

(Schelling 1960)

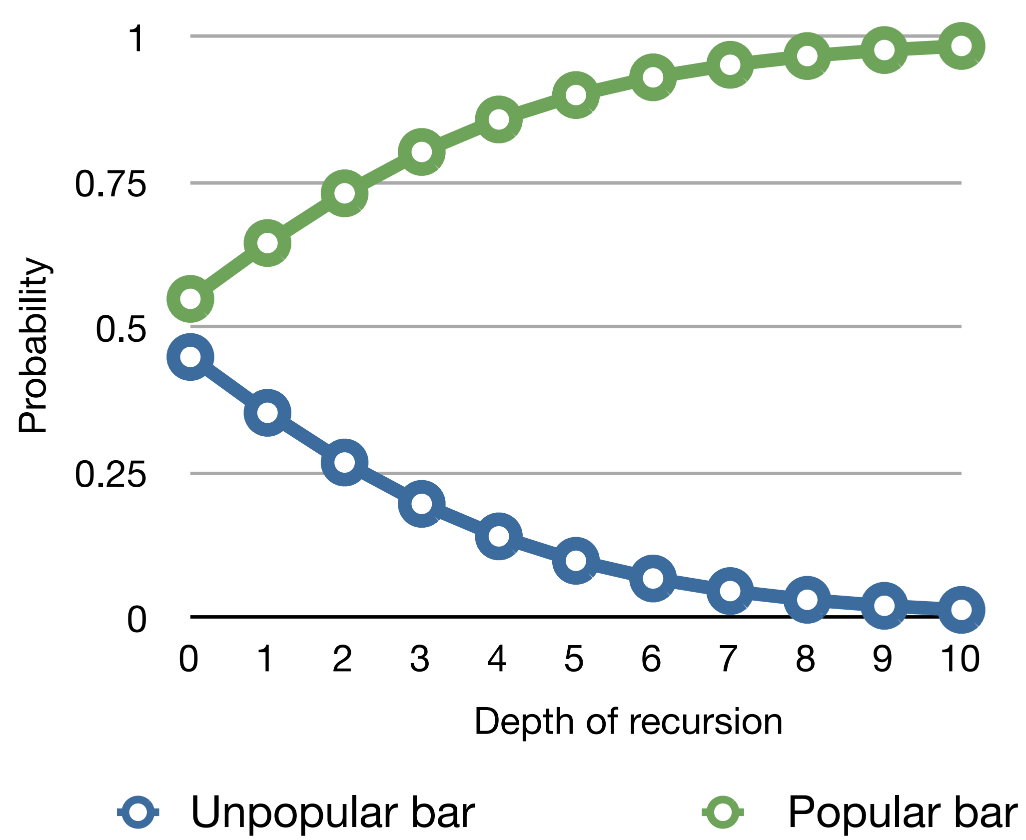
A simple coordination game

```
(define (sample-location)
  (if (flip .55)
      'popular-bar
      'unpopular-bar))

(define (alice depth)
  (query
   (define alice-location (sample-location))
   alice-location
   (equal? alice-location (bob (- depth 1)))))

(define (bob depth)
  (query
   (define bob-location (sample-location))
   bob-location
   (or (= depth 0)
       (equal? bob-location (alice depth)))))
```

(Schelling 1960)

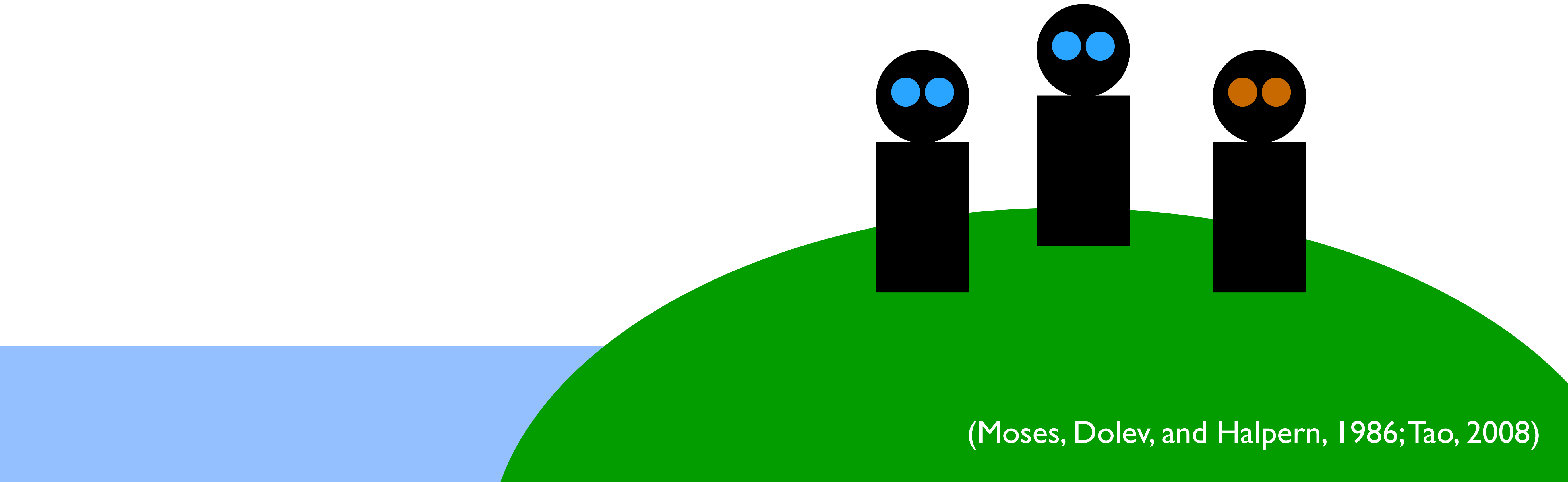
A simple coordination game

```
(define (sample-location)
  (if (flip .55)
      'popular-bar
      'unpopular-bar))

(define (alice depth)
  (query
   (define alice-location (sample-location))
   alice-location
   (equal? alice-location (bob (- depth 1)))))

(define (bob depth)
  (query
   (define bob-location (sample-location))
   bob-location
   (or (= depth 0)
       (equal? bob-location (alice depth)))))
```



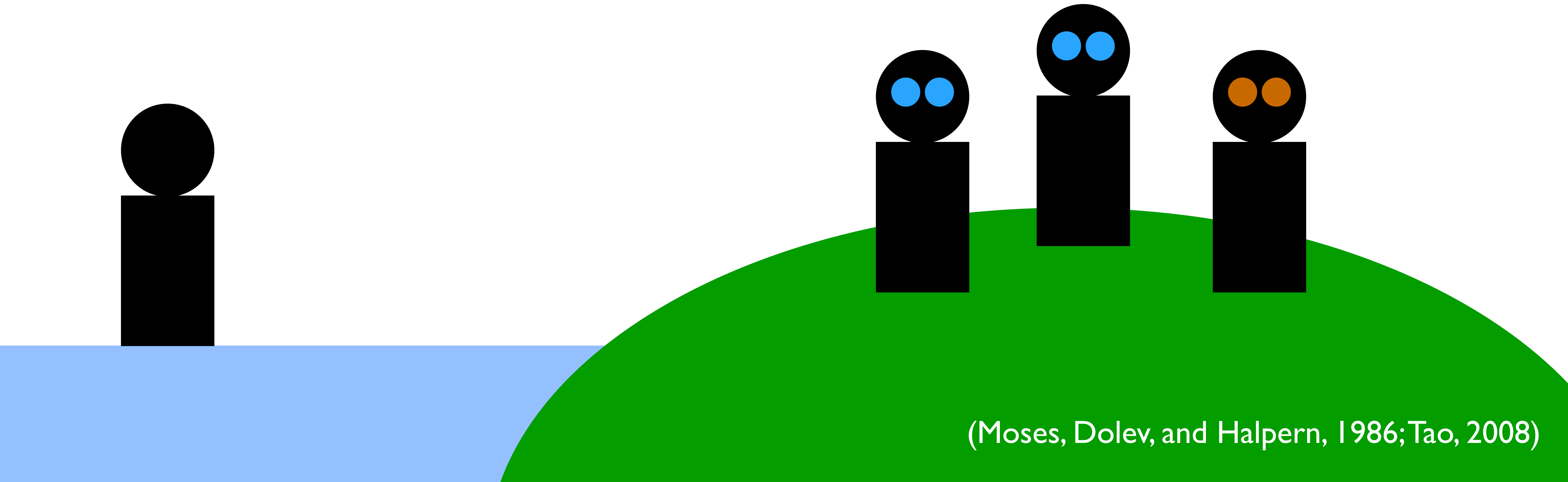
(Schelling 1960)

The blue-eyed islanders puzzle



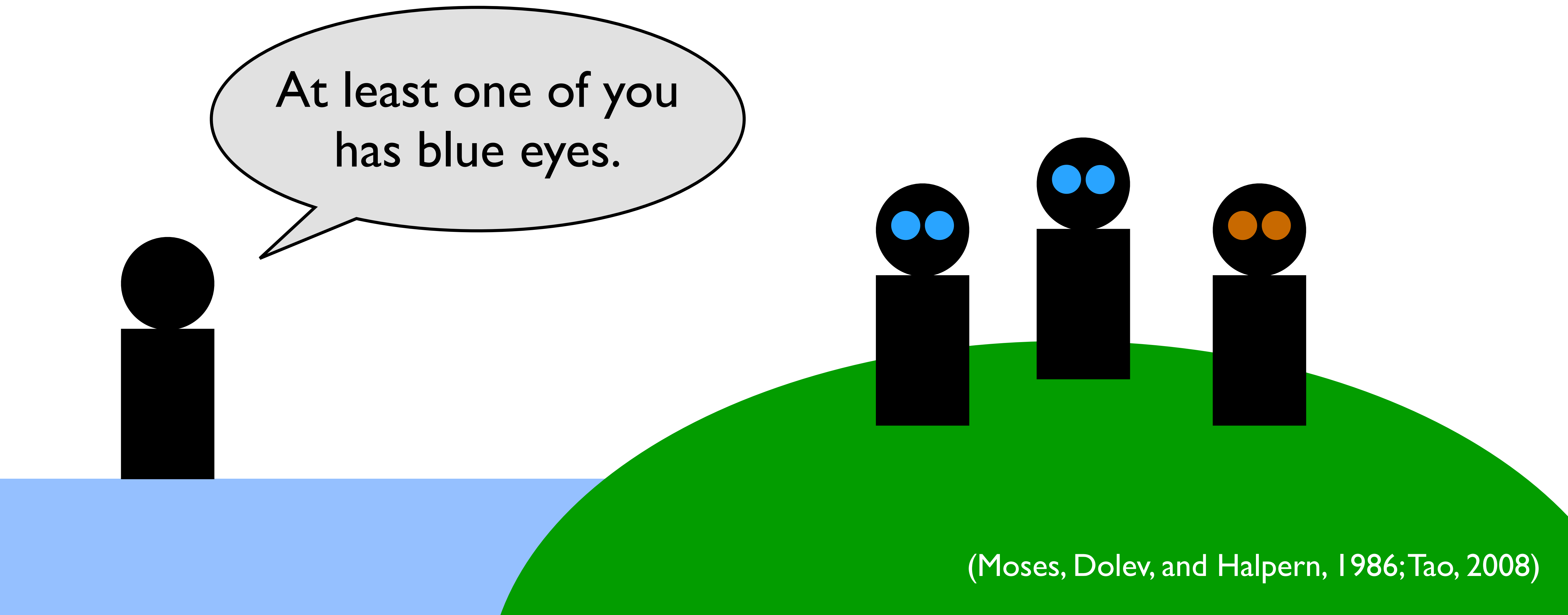
(Moses, Dolev, and Halpern, 1986; Tao, 2008)

The blue-eyed islanders puzzle



(Moses, Dolev, and Halpern, 1986; Tao, 2008)

The blue-eyed islanders puzzle



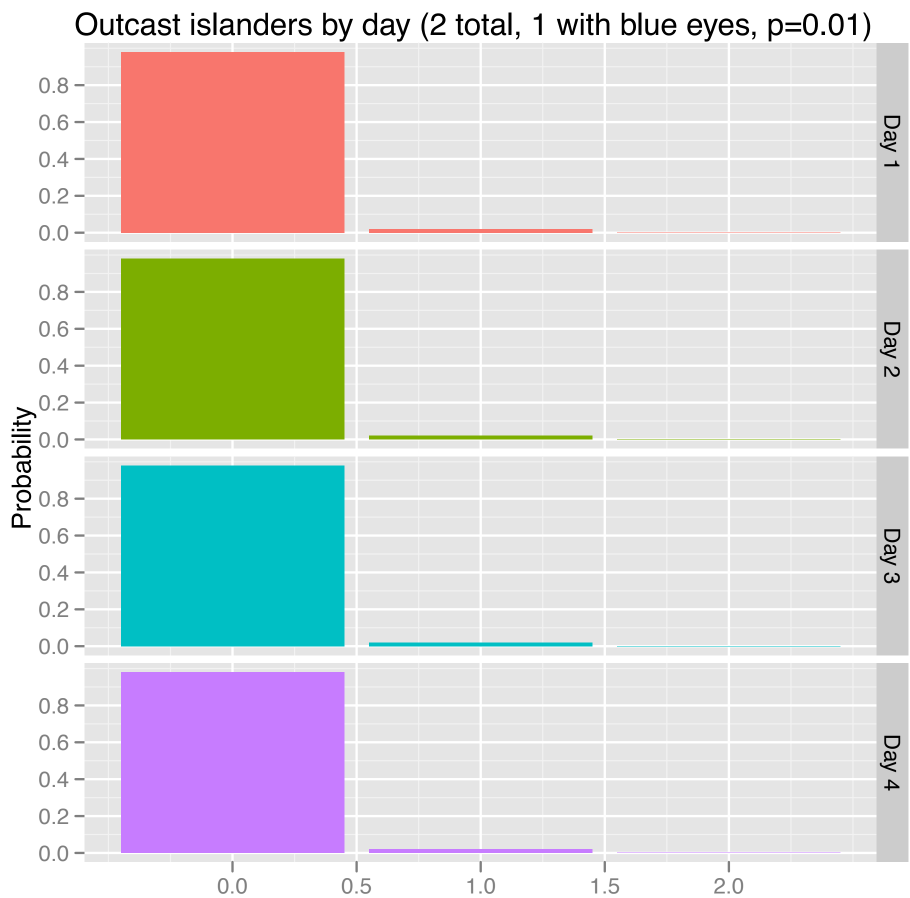
The blue-eyed islanders puzzle

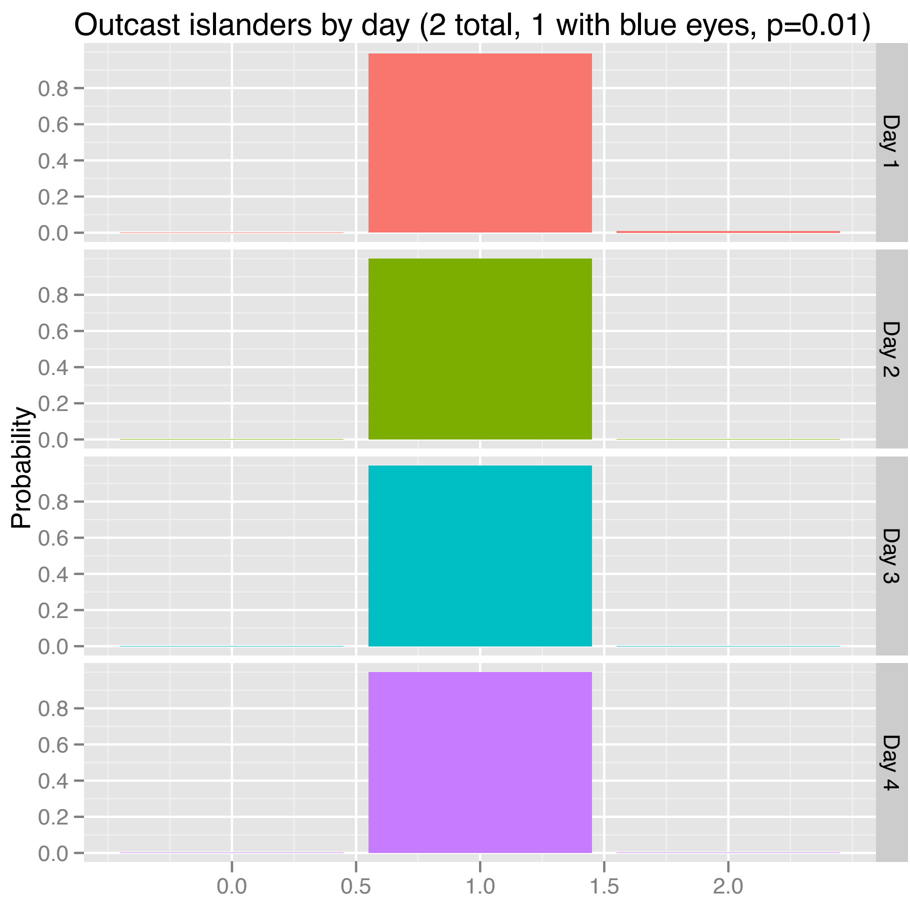
- Two arguments:
  - "The outsider provides no new information, therefore nothing changes."
  - "By induction on the number of blue-eyed people, all *N* blue-eyed people leave on day *N*."

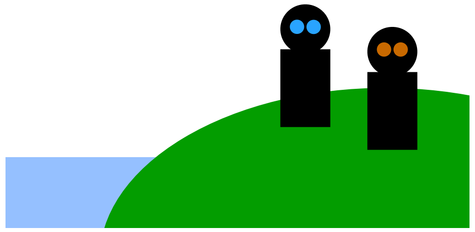
The blue-eyed islanders puzzle

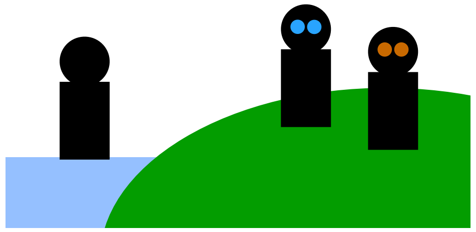
- Difficult for current universal inference algorithms
  - Deeply nested queries
  - Complex (almost-)deterministic dependencies
- But: Potential for sharing of subcomputations

The blue-eyed islanders puzzle

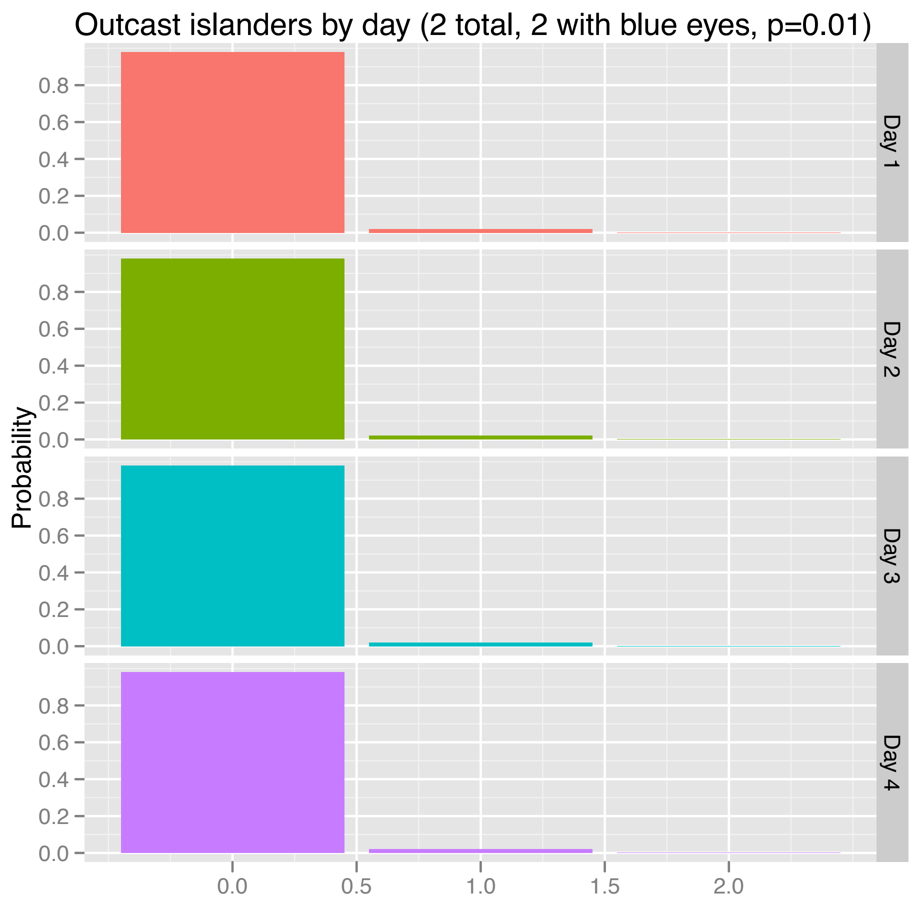


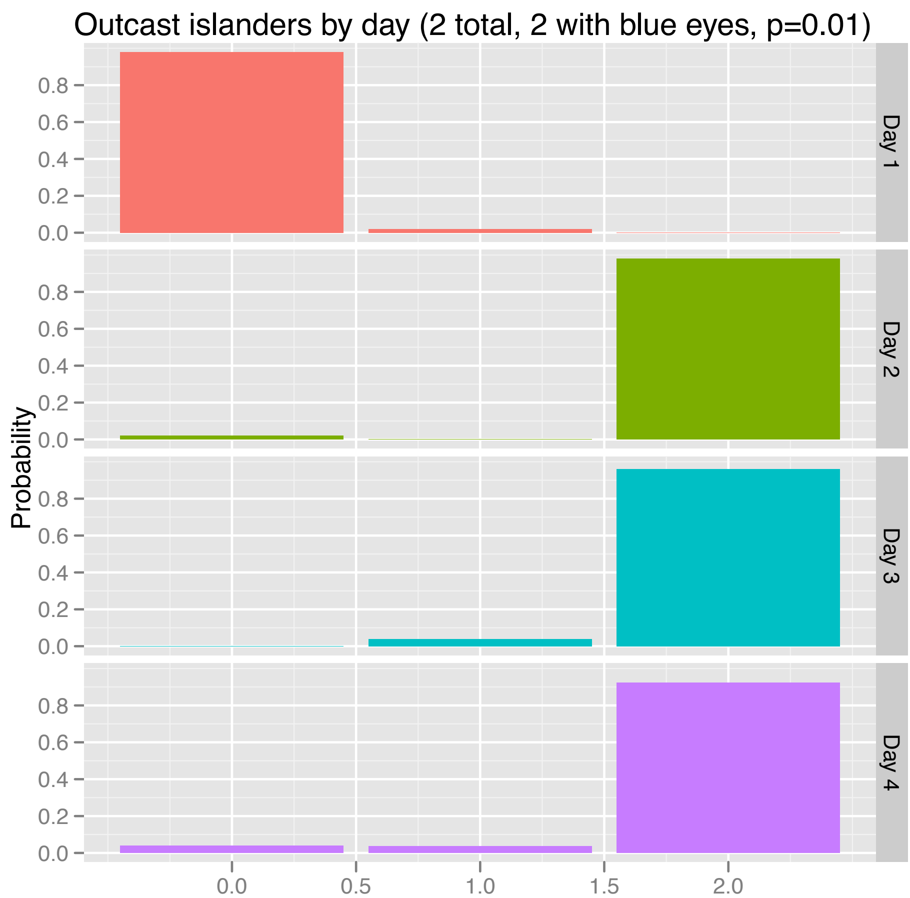


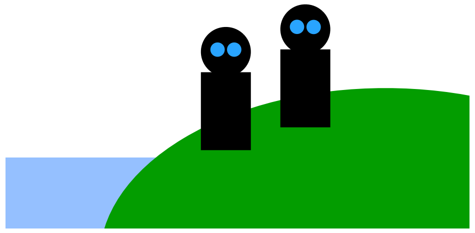


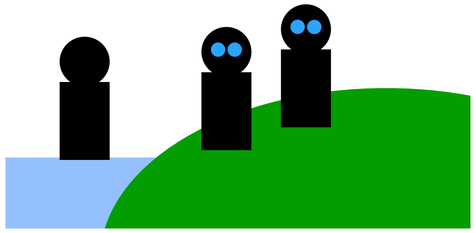


The blue-eyed islanders puzzle

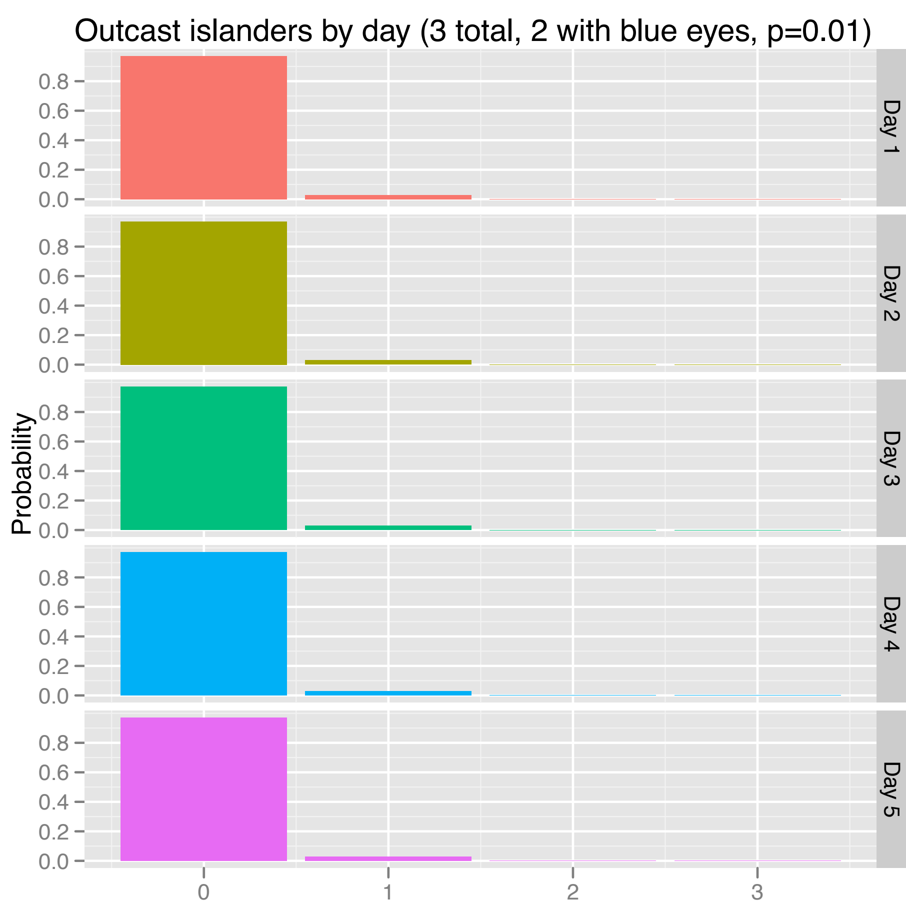


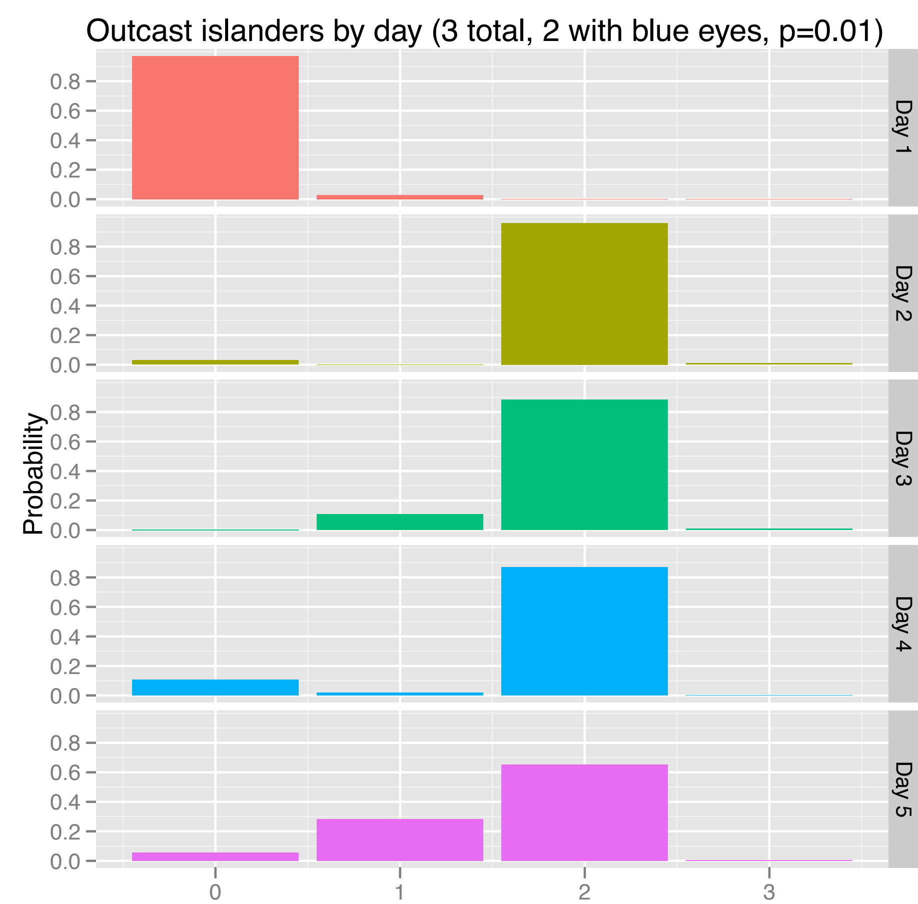


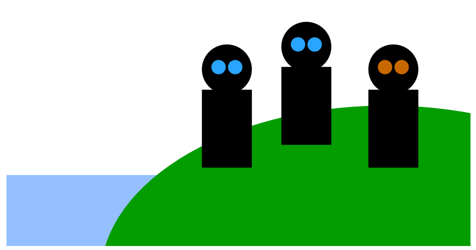


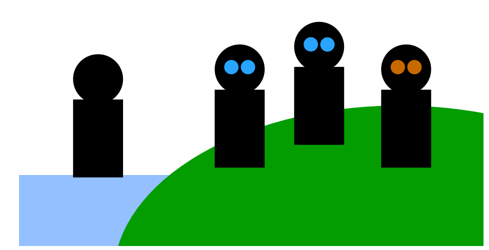


The blue-eyed islanders puzzle

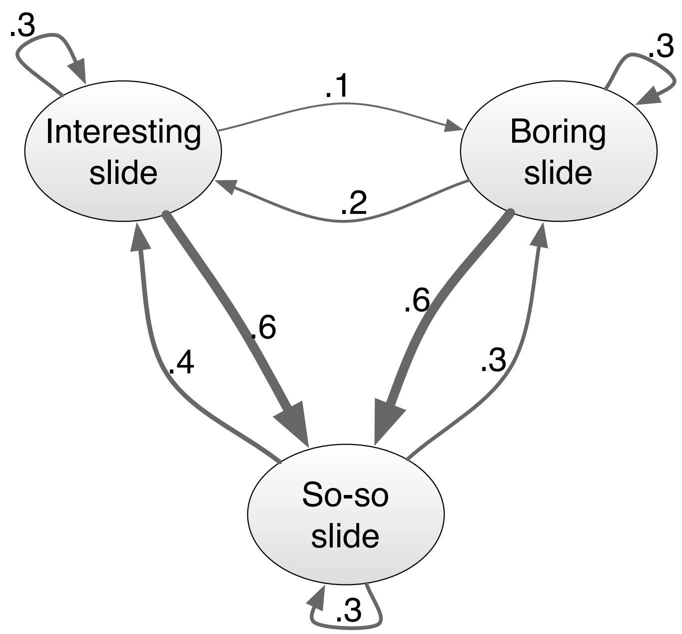






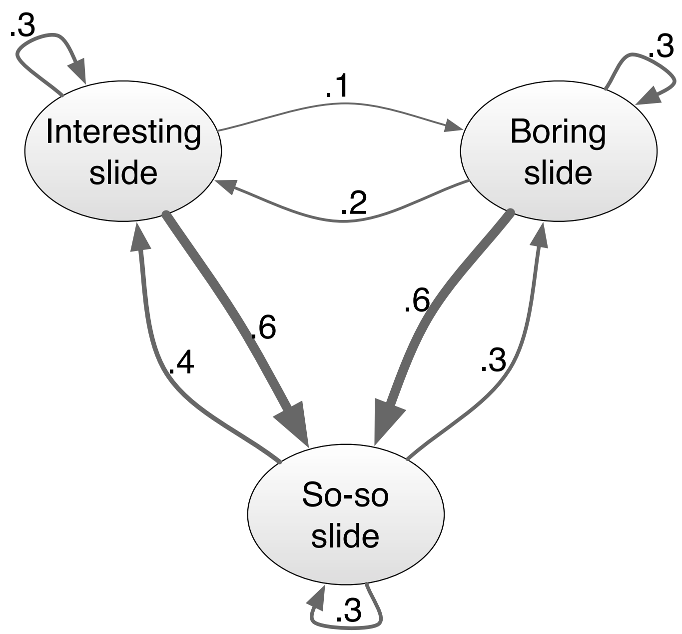


Markov Models



- Set of states $Q = \{S_+, S_0, S_-\}$
- Transition probability matrix A
- Distribution on initial states P_{init}
- Set of end states Q_{end}

Markov Models

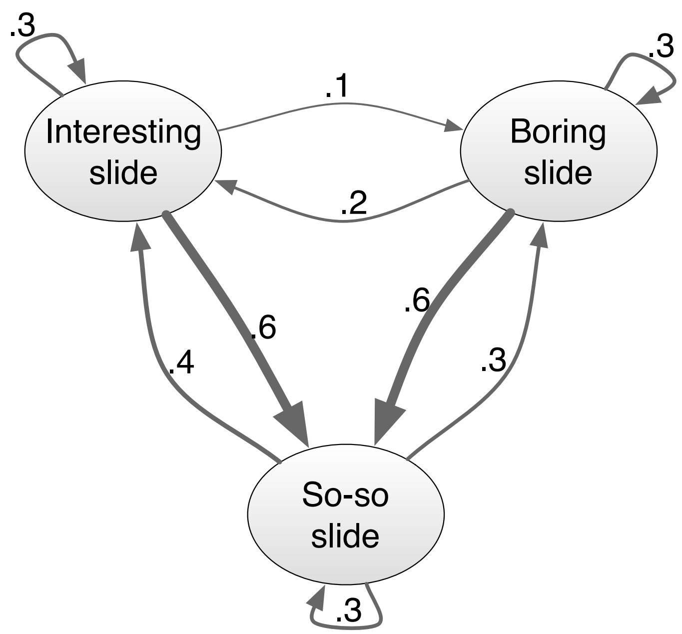


- Set of states $Q = \{S_+, S_0, S_-\}$
- Transition probability matrix A
- Distribution on initial states P_{init}
- Set of end states Q_{end}

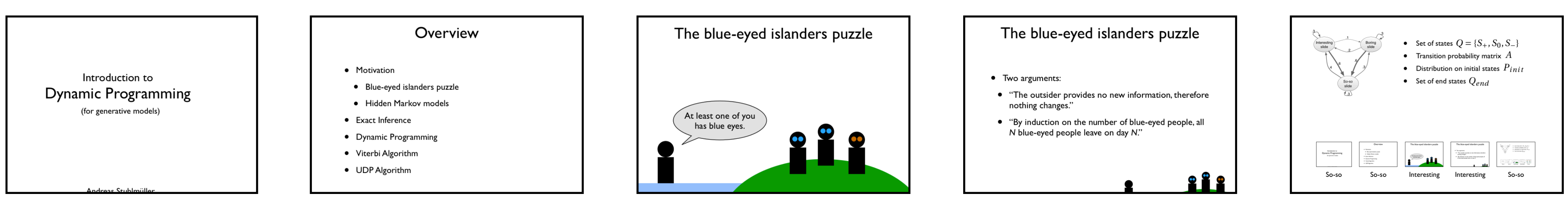
Markov assumption:

$$P(q_i | q_1, \ldots, q_{i-1}) = P(q_i | q_{i-1})$$

Markov Models

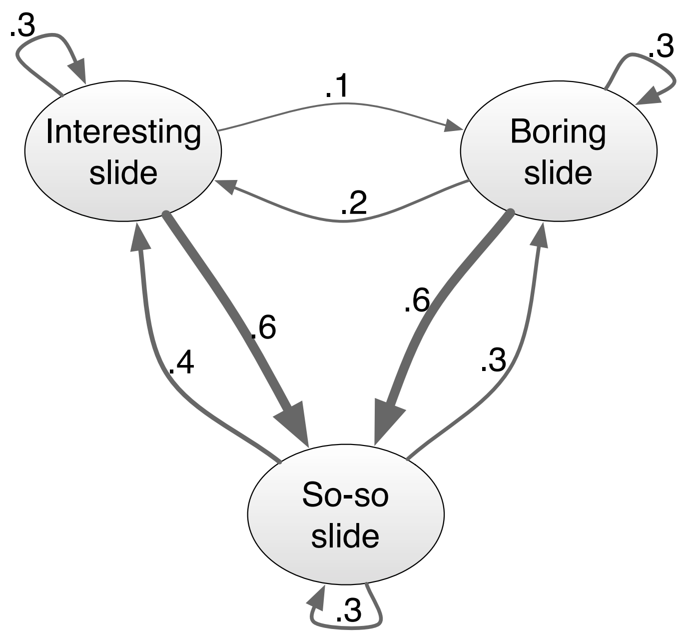


- Set of states $Q = \{S_+, S_0, S_-\}$
- Transition probability matrix A
- Distribution on initial states P_{init}
- Set of end states Q_{end}



So-so | So-so | Interesting | Interesting | So-so

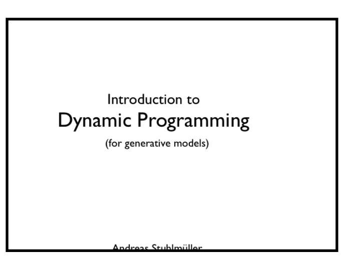
Markov Models



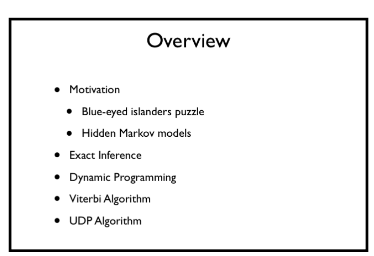
- Set of states $Q = \{S_+, S_0, S_-\}$
- Transition probability matrix A
- Distribution on initial states P_{init}
- Set of end states Q_{end}

$$P([S_0, S_0, S_+, S_+, S_0]) =$$

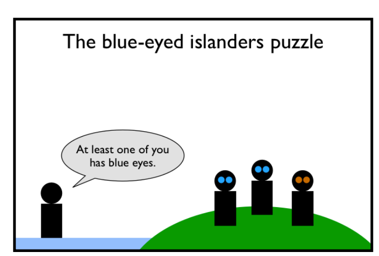
$$P_{init}(S_0) \quad * \quad P(S_0|S_0) \quad * \quad P(S_+|S_0) \quad * \quad P(S_+|S_+) \quad * \quad P(S_0|S_+)$$



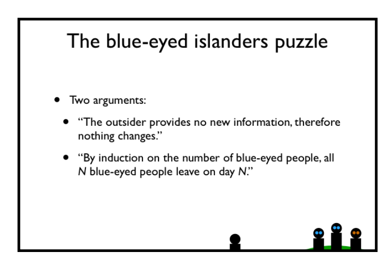
So-so



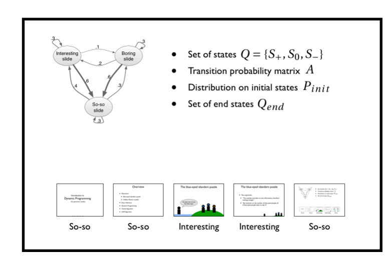
So-so



Interesting

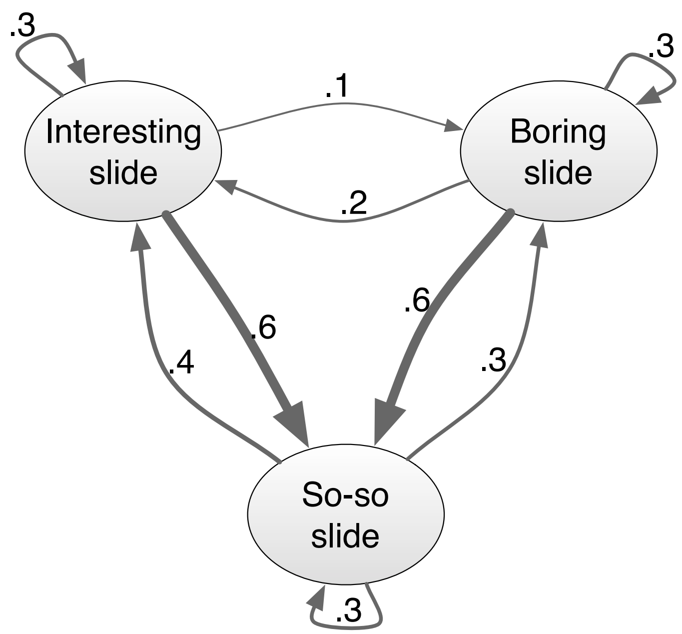


Interesting



So-so

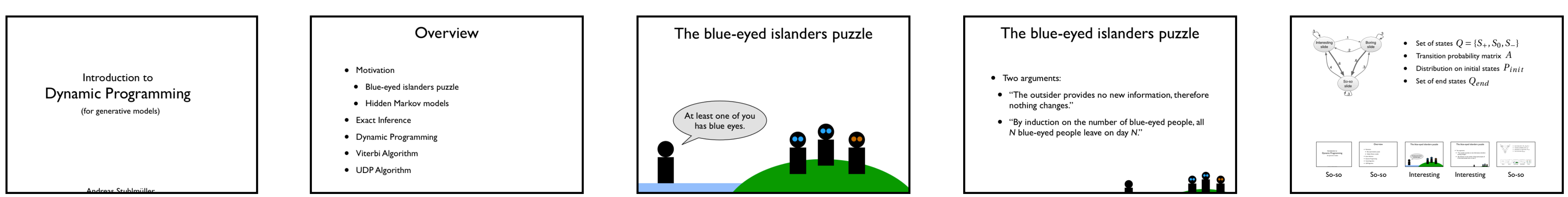
Markov Models



- Set of states $Q = \{S_+, S_0, S_-\}$
- Transition probability matrix A
- Distribution on initial states P_{init}
- Set of end states Q_{end}

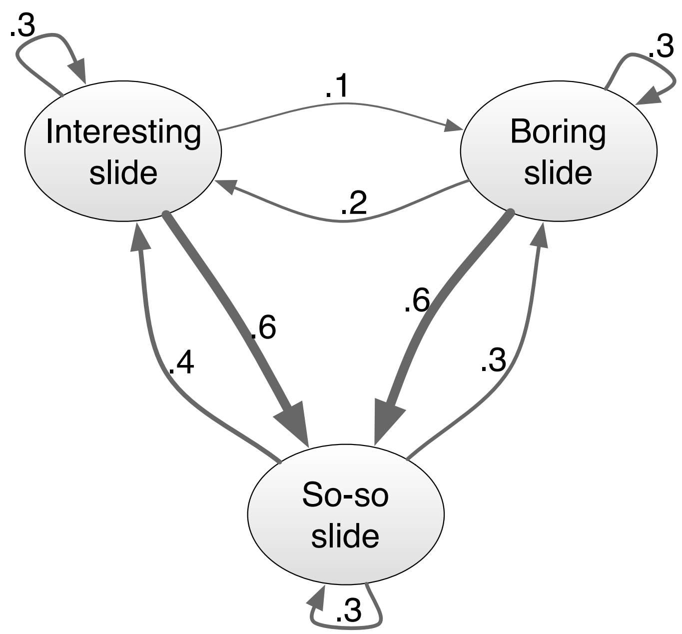
$$P([S_0, S_0, S_+, S_+, S_0]) =$$

$$.33 \quad * \quad .3 \quad * \quad .4 \quad * \quad .3 \quad * \quad .6 \quad = \quad .0071$$

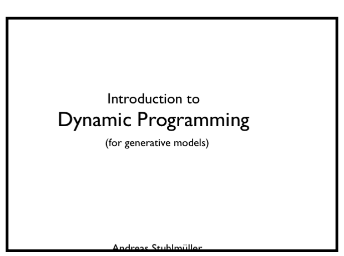


So-so | So-so | Interesting | Interesting | So-so

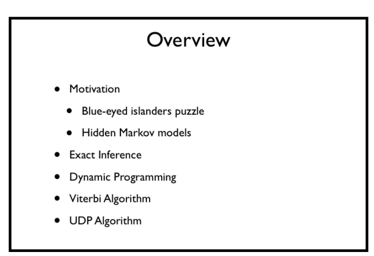
Markov Models



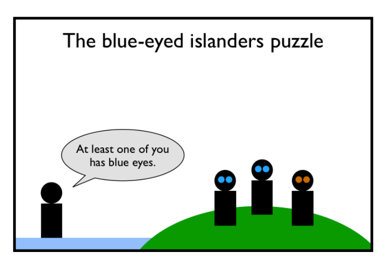

```
(define transition-dists
  '((interesting . (.3 .6 .1))
    (so-so . (.4 .3 .3))
    (boring . (.2 .6 .3))))

(define (transition state)
  (multinomial '(interesting so-so boring)
               (rest (assoc state transition-dists))))

(define (mm state n)
  (if (= n 0)
      '()
      (pair state
            (mm (transition state) (- n 1)))))
```



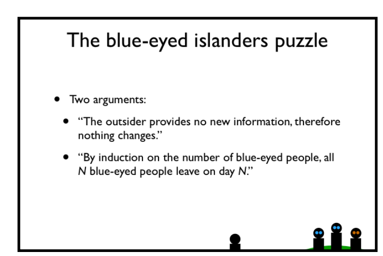
So-so



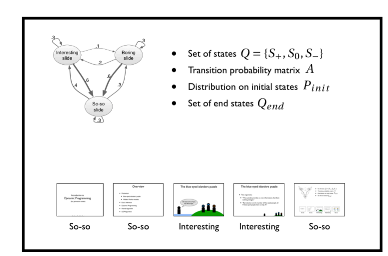
So-so



Interesting

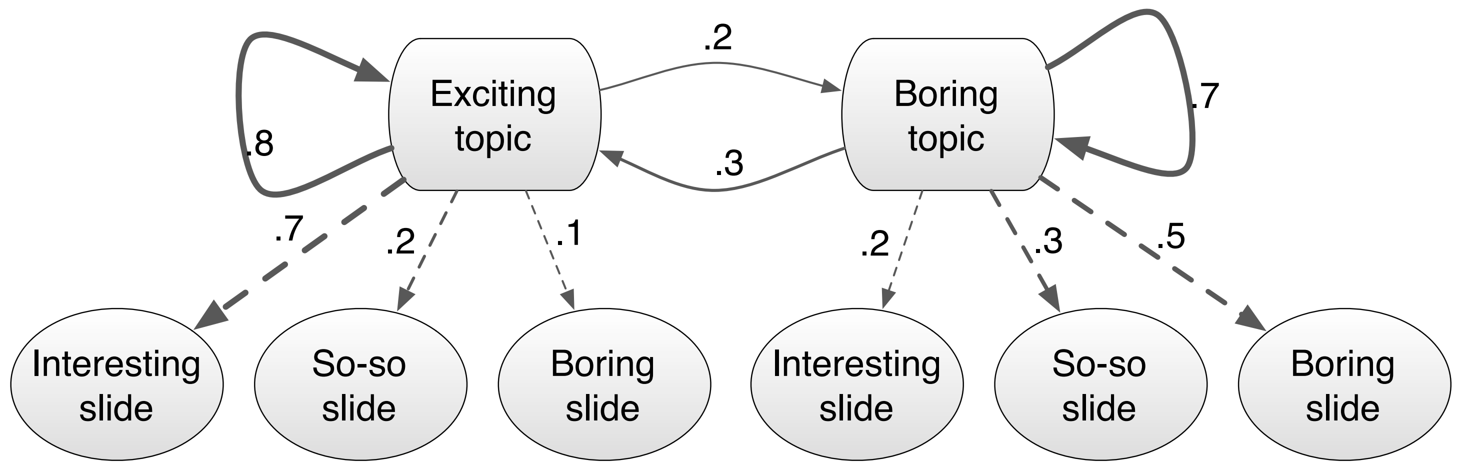


Interesting



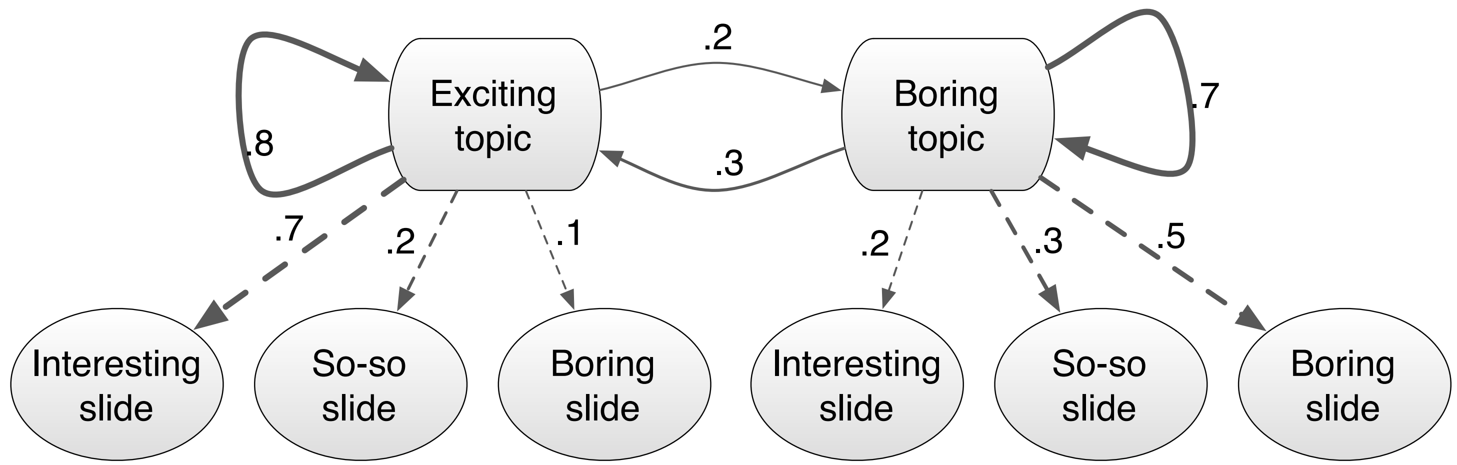
So-so

Hidden Markov Models



- Set of states $Q = \{E, \neg E\}$
- Transition probability matrix A
- Distribution on initial states P_{init}
- Set of end states Q_{end}
- Set of possible observations $V = \{S_+, S_0, S_-\}$
- Observation probability matrix O

Hidden Markov Models

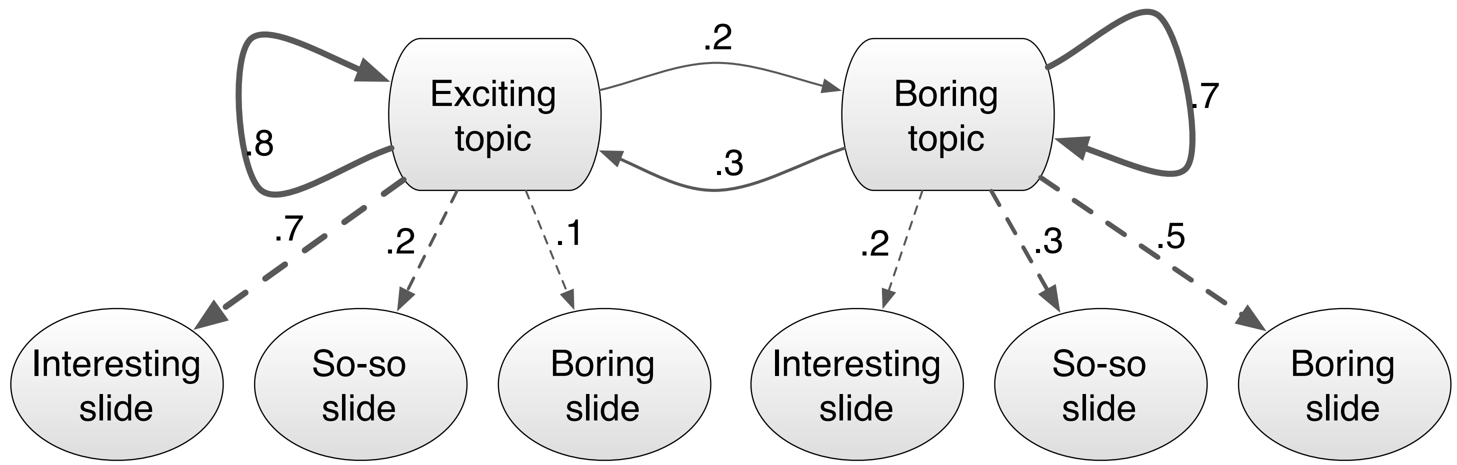


- Set of states $Q = \{E, \neg E\}$
- Transition probability matrix A
- Distribution on initial states P_{init}
- Set of end states Q_{end}
- Set of possible observations $V = \{S_+, S_0, S_-\}$
- Observation probability matrix O

Markov assumption: $P(q_i|q_1,\ldots,q_{i-1}) = P(q_i|q_{i-1})$

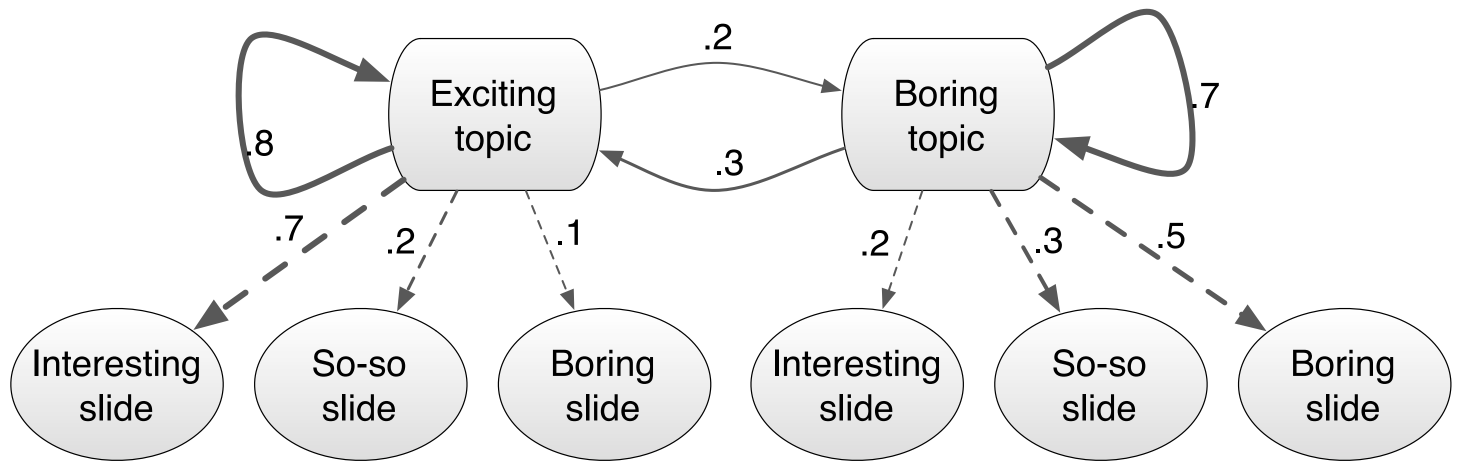
Output independence: $P(o_i|q_1,\ldots,q_i,\ldots,q_T) = P(o_i|q_i)$

Hidden Markov Models



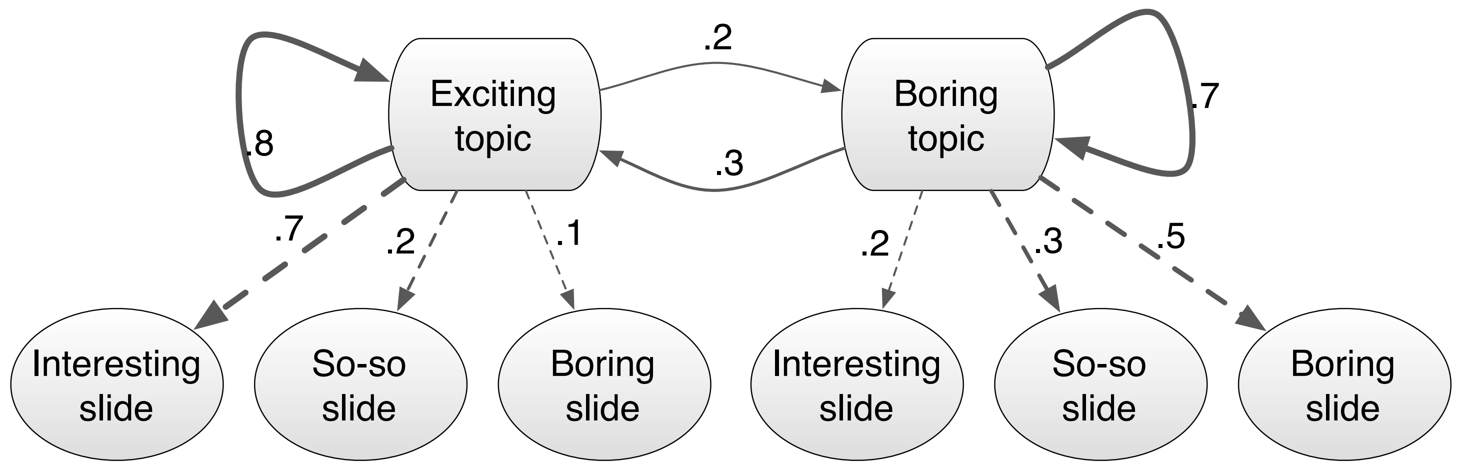

```
(define (mm state n)
  (if (= n 0)
      '()
      (pair state
            (mm (transition state) (- n 1)))))
```

Hidden Markov Models



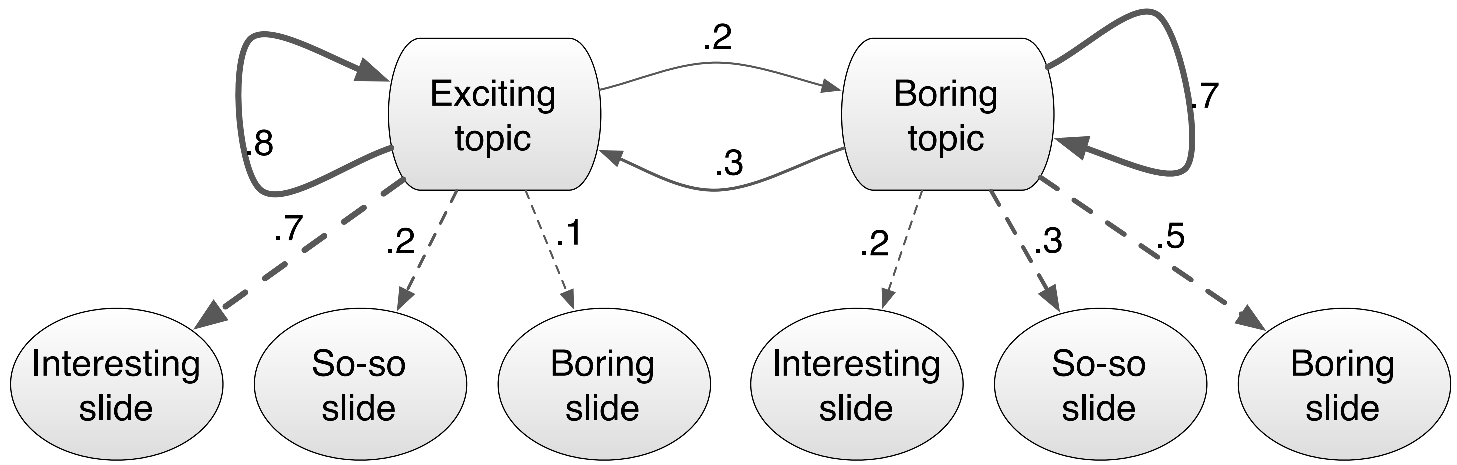

```
(define (mm state n)
  (if (= n 0)
      '()
      (pair state
            (mm (transition state) (- n 1)))))


(define (hmm state n)
  (if (= n 0)
      '()
      (pair (observe state)
            (mm (transition state) (- n 1)))))
```

Hidden Markov Models

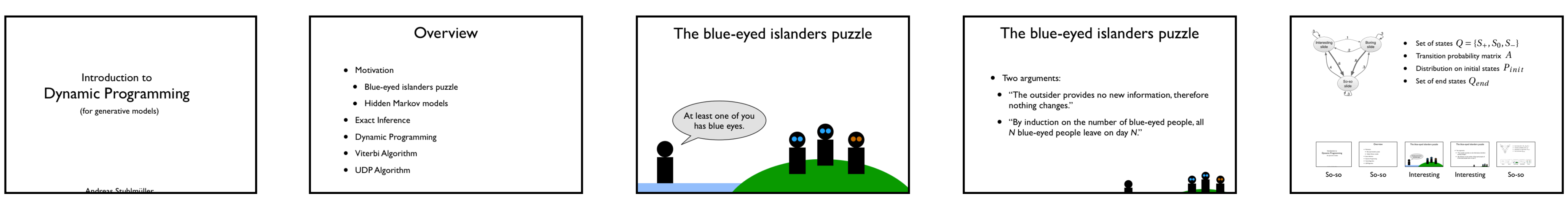



```
(define (mm state n)
  (if (= n 0)
      '()
      (pair state
            (mm (transition state) (- n 1)))))


(define (hmm state n)
  (if (= n 0)
      '()
      (pair (observe state)
            (mm (transition state) (- n 1)))))
```

Hidden Markov Models

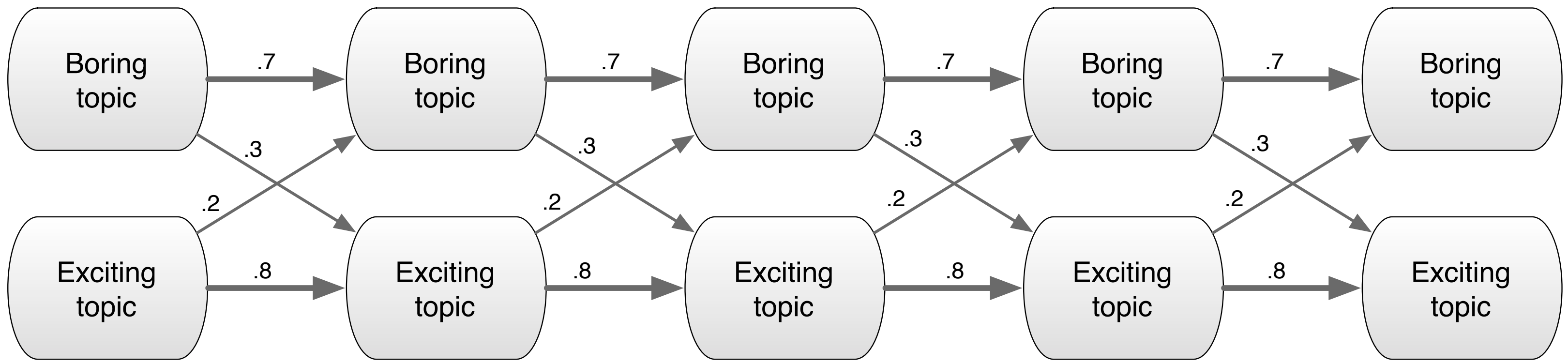


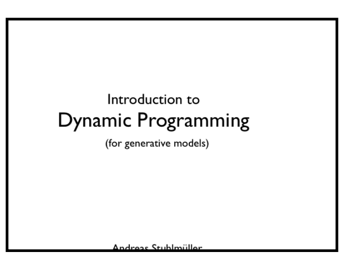
- Set of states $Q = \{E, \neg E\}$
- Transition probability matrix A
- Distribution on initial states P_{init}
- Set of end states Q_{end}
- Set of possible observations $V = \{S_+, S_0, S_-\}$
- Observation probability matrix O



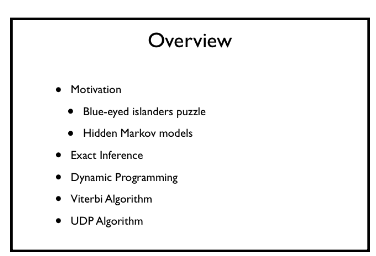
So-so　So-so　Interesting　Interesting　So-so

Inference by Enumeration

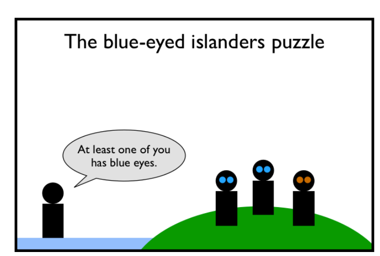




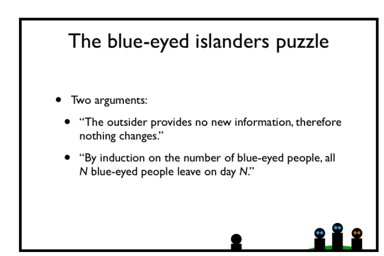
So-so



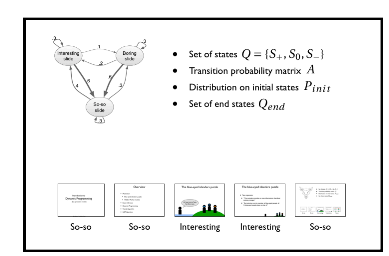
So-so



Interesting

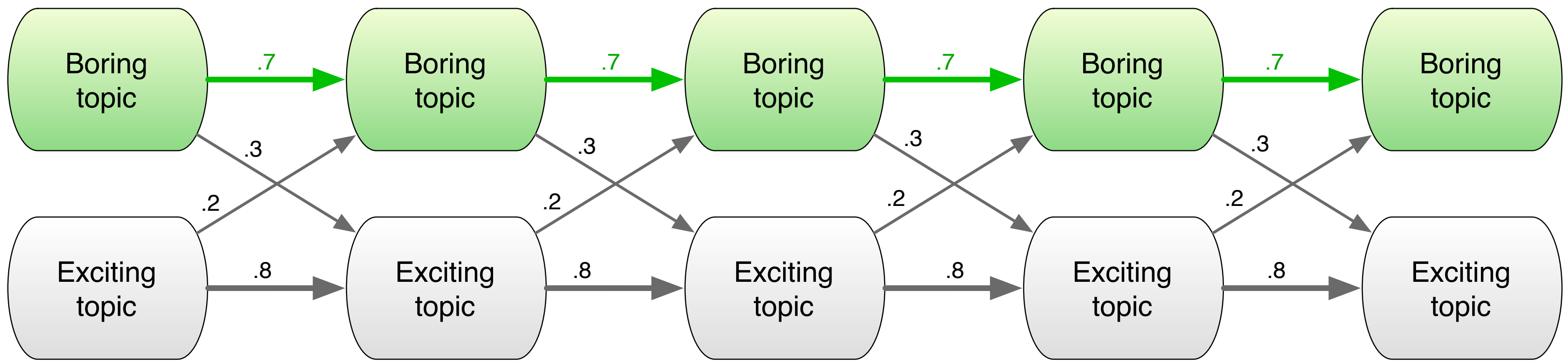


Interesting

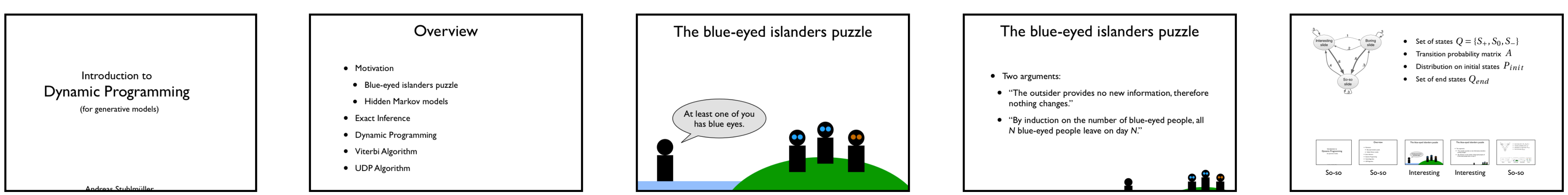


So-so

Inference by Enumeration

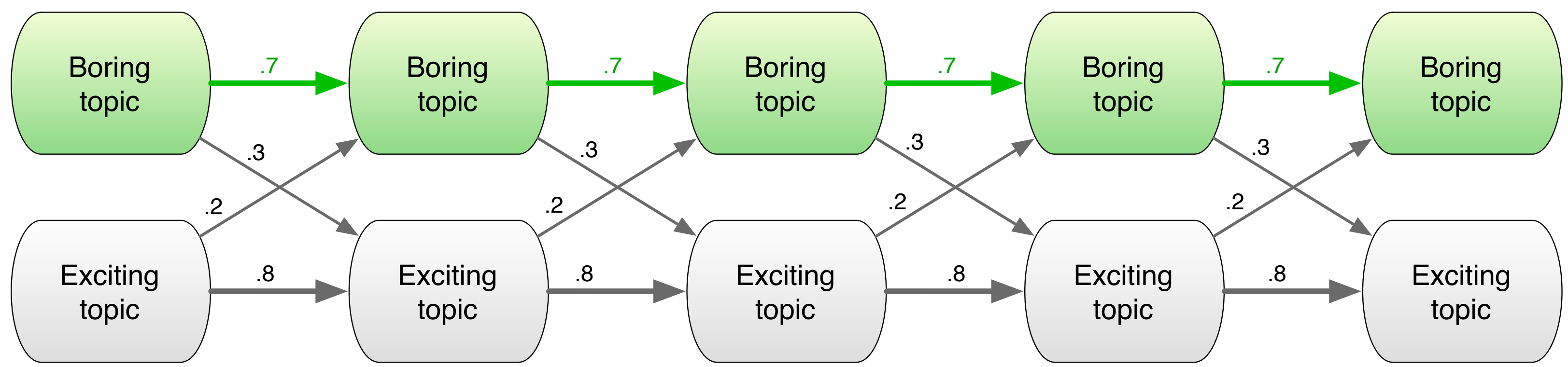


$$P(o_{1:n}|q_{1:n}) = P(o_n|q_n)\ldots P(o_1|q_1)$$

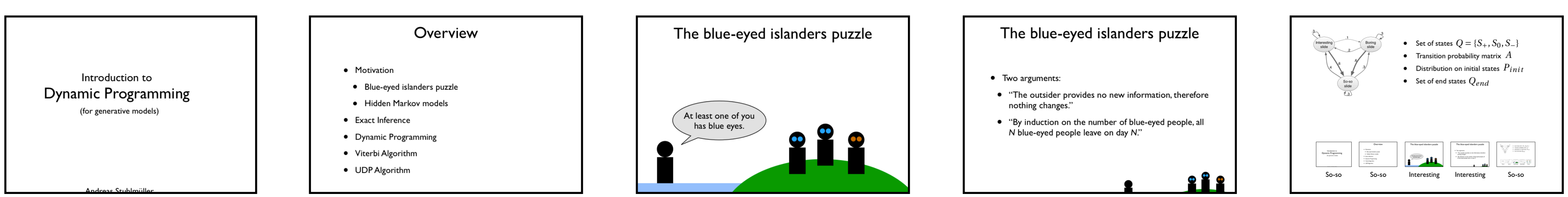


So-so | So-so | Interesting | Interesting | So-so

Inference by Enumeration

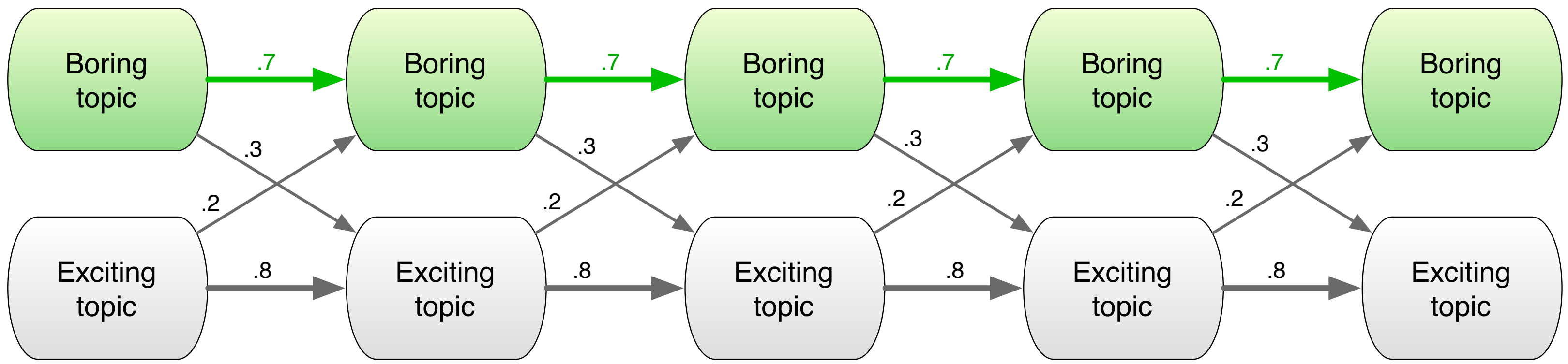


$$P(q_{1:n}) = P(q_n|q_{n-1})\ldots P(q_2|q_1)P_{init}(q_1)$$



So-so So-so Interesting Interesting So-so

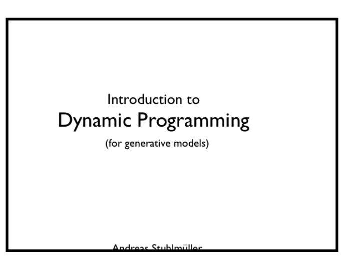
Inference by Enumeration



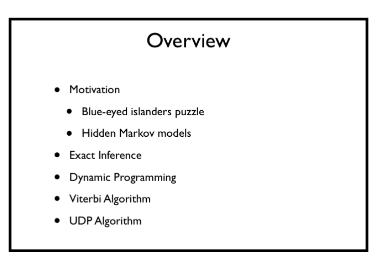
$$P(o_{1:n}|q_{1:n}) = P(o_n|q_n)\ldots P(o_1|q_1)$$

$$P(q_{1:n}) = P(q_n|q_{n-1})\ldots P(q_2|q_1)P_{init}(q_1)$$

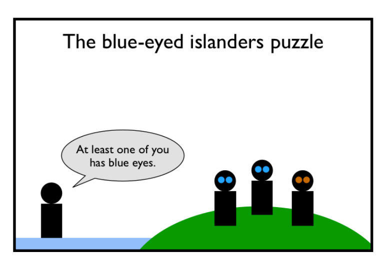
$$P(o_{1:n}) = \sum_{q_{1:n}\in Q^n} P(o_{1:n}|q_{1:n})P(q_{1:n})$$



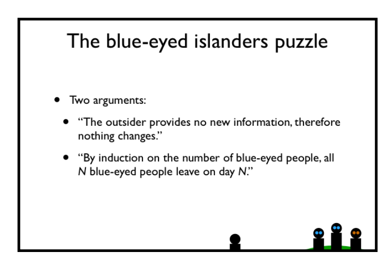
So-so



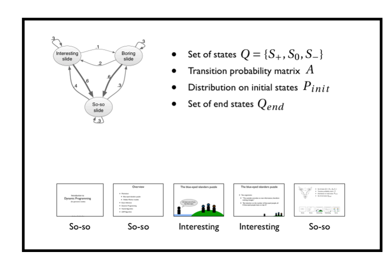
So-so



Interesting

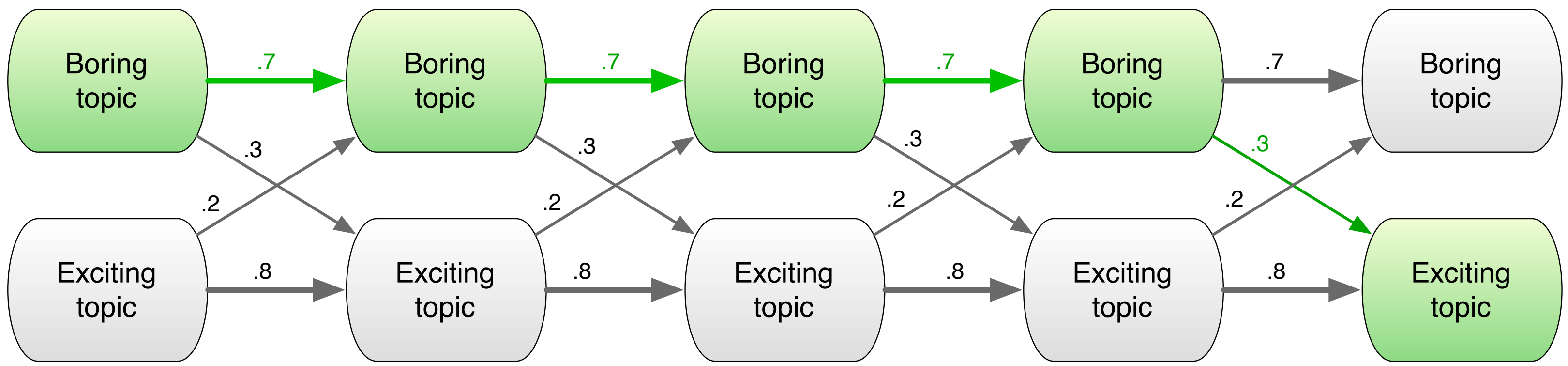


Interesting



So-so

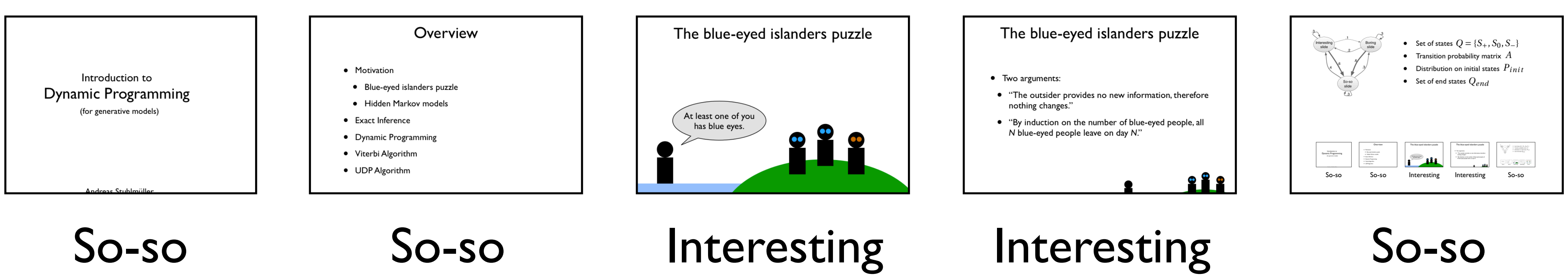
Inference by Enumeration



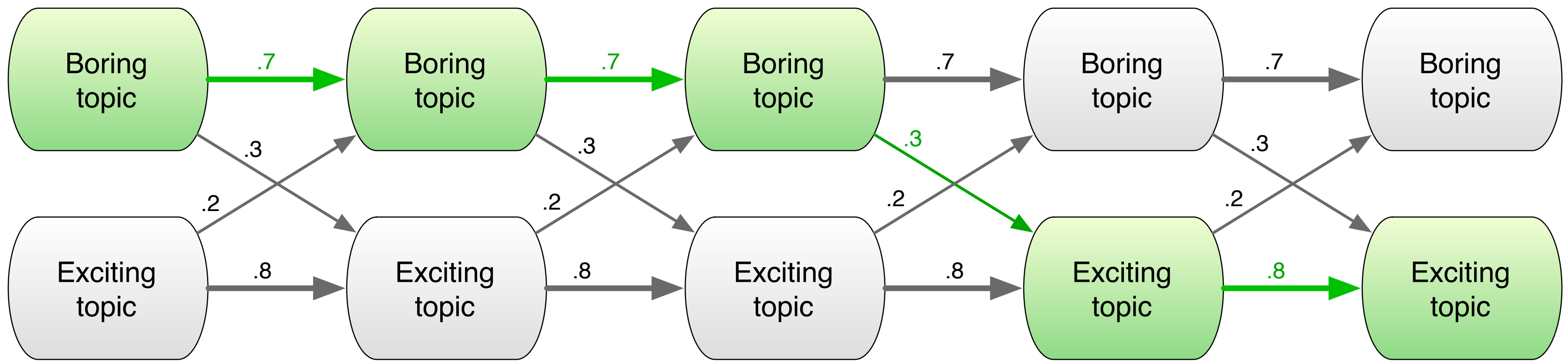
$$P(o_{1:n}|q_{1:n}) = P(o_n|q_n)\dots P(o_1|q_1)$$

$$P(q_{1:n}) = P(q_n|q_{n-1})\dots P(q_2|q_1)P_{init}(q_1)$$

$$P(o_{1:n}) = \sum_{q_{1:n}\in Q^n} P(o_{1:n}|q_{1:n})P(q_{1:n})$$



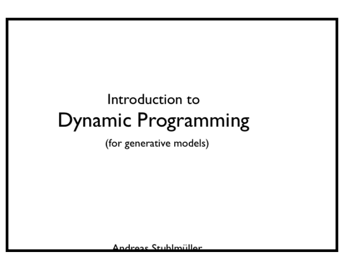
Inference by Enumeration



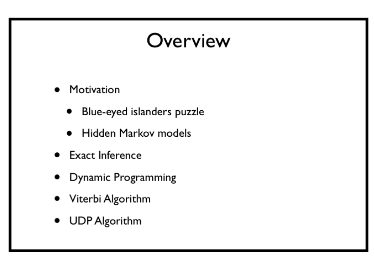
$$P(o_{1:n}|q_{1:n}) = P(o_n|q_n)\dots P(o_1|q_1)$$

$$P(q_{1:n}) = P(q_n|q_{n-1})\dots P(q_2|q_1)P_{init}(q_1)$$

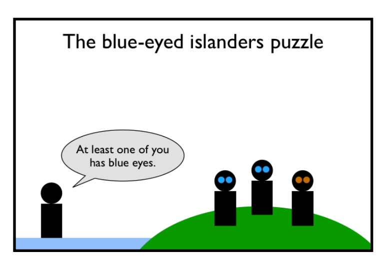
$$P(o_{1:n}) = \sum_{q_{1:n}\in Q^n} P(o_{1:n}|q_{1:n})P(q_{1:n})$$



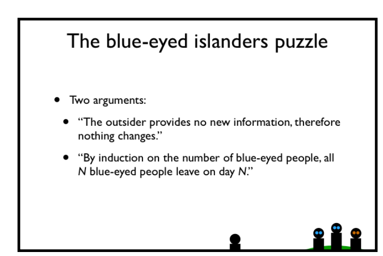
So-so



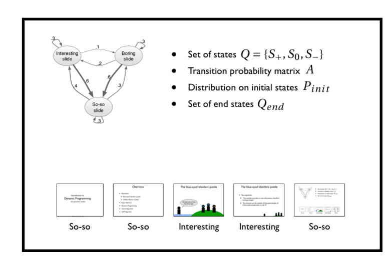
So-so



Interesting

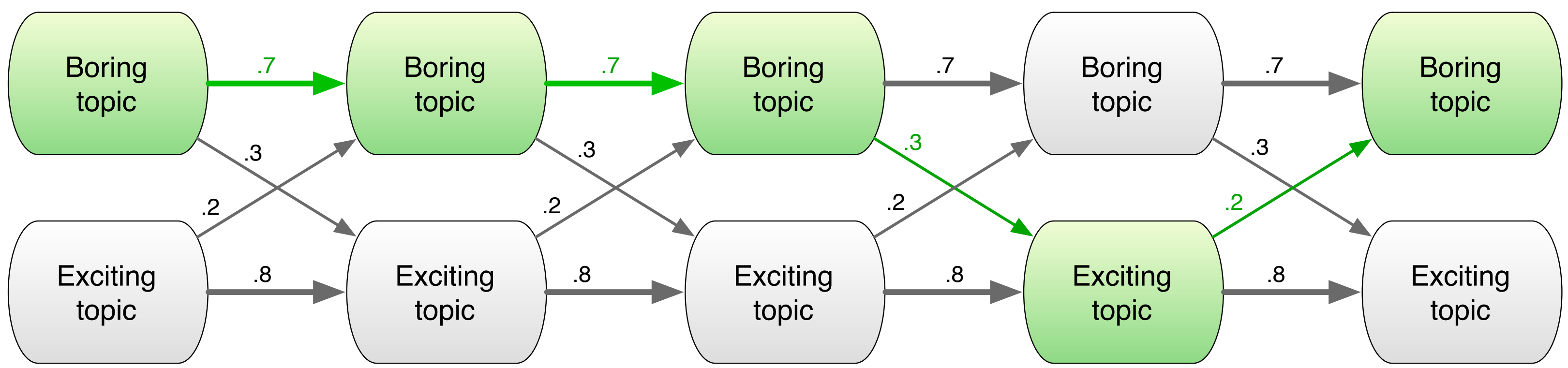


Interesting



So-so

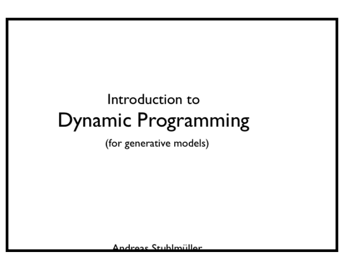
Inference by Enumeration



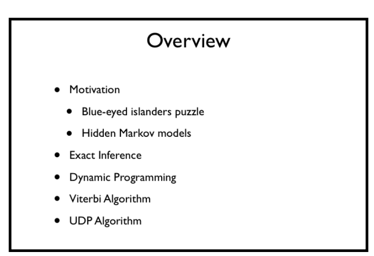
$$P(o_{1:n}|q_{1:n}) = P(o_n|q_n)\ldots P(o_1|q_1)$$

$$P(q_{1:n}) = P(q_n|q_{n-1})\ldots P(q_2|q_1)P_{init}(q_1)$$

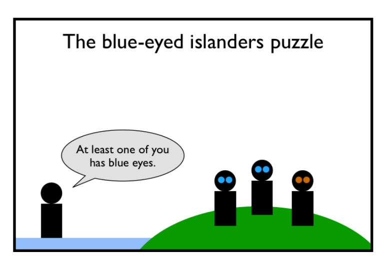
$$P(o_{1:n}) = \sum_{q_{1:n}\in Q^n} P(o_{1:n}|q_{1:n})P(q_{1:n})$$



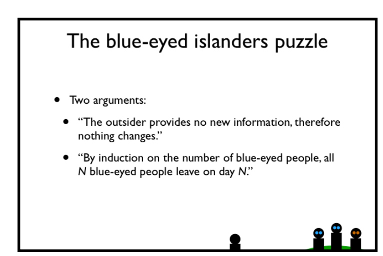
So-so



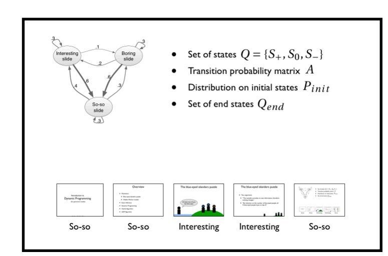
So-so



Interesting

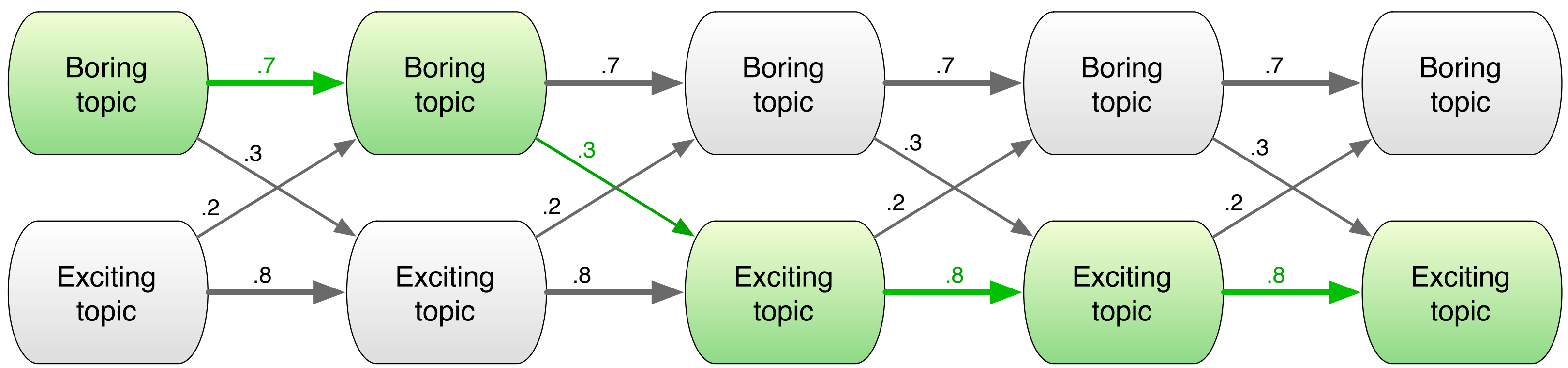


Interesting



So-so

Inference by Enumeration

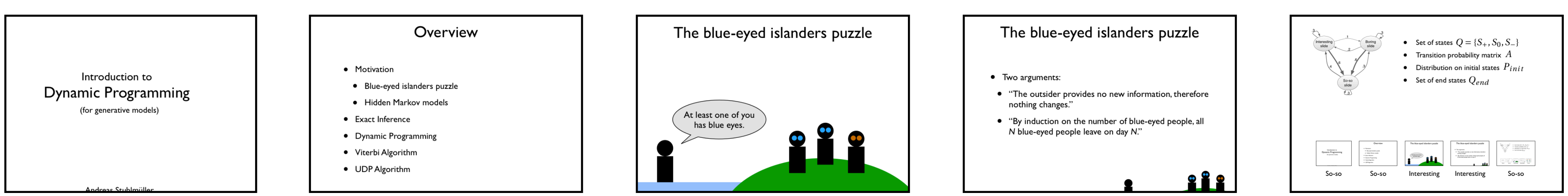


$|Q|^n$ paths

$$P(o_{1:n}|q_{1:n}) = P(o_n|q_n)\ldots P(o_1|q_1)$$

$$P(q_{1:n}) = P(q_n|q_{n-1})\ldots P(q_2|q_1)P_{init}(q_1)$$

$$P(o_{1:n}) = \sum_{q_{1:n}\in Q^n} P(o_{1:n}|q_{1:n})P(q_{1:n})$$



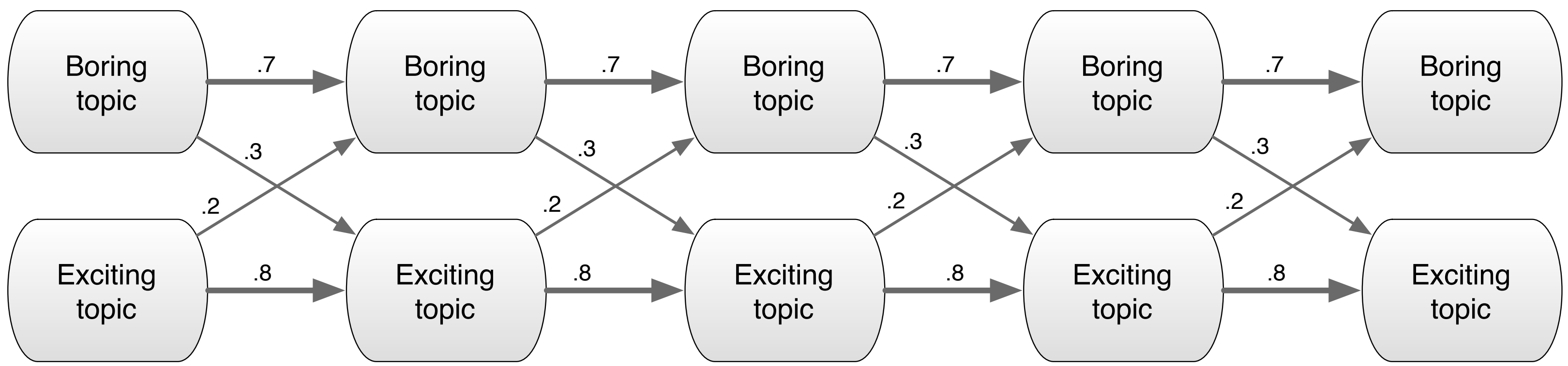
So-so | So-so | Interesting | Interesting | So-so

Dynamic Programming

- Idea
  - Repeatedly reduce problem to smaller problems
  - Cache and reuse solutions to subproblems
- Requirements
  - Optimal substructure: Optimal solution to problem contains optimal solution to subproblems
  - Overlapping subproblems: Within solution to overall problem, need to solve subproblems multiple times

(Cormen, 2001)

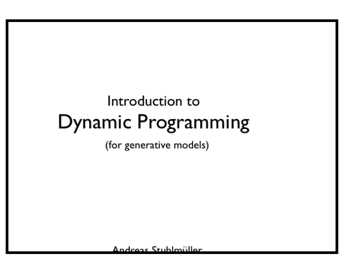
Forward Algorithm



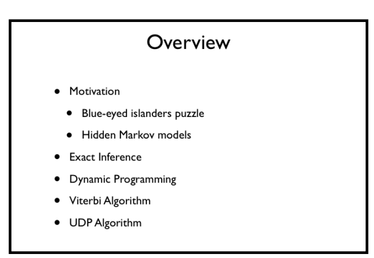
$$O(N|Q|^2)$$

$$P(o_{1:i}, q) = \begin{cases} P_{init}(q) P(o_1|q) & \text{if } i = 1 \\ \sum_{r \in Q} P(o_{1:(i-1)}, r) P(q|r) P(o_i|q) & \text{otherwise} \end{cases}$$

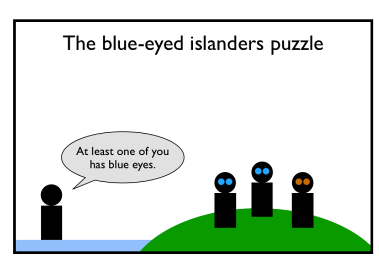
$$P(o_{1:i}) = \sum_{q \in Q} P(o_{1:i}, q)$$



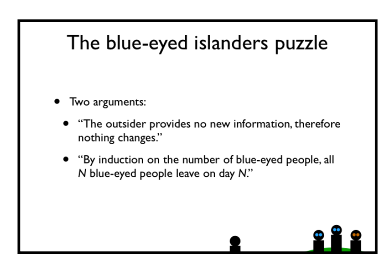
So-so



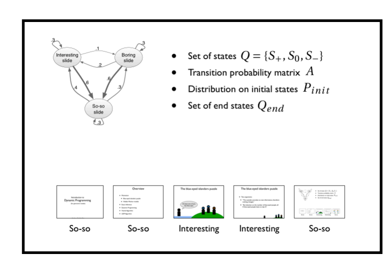
So-so



Interesting

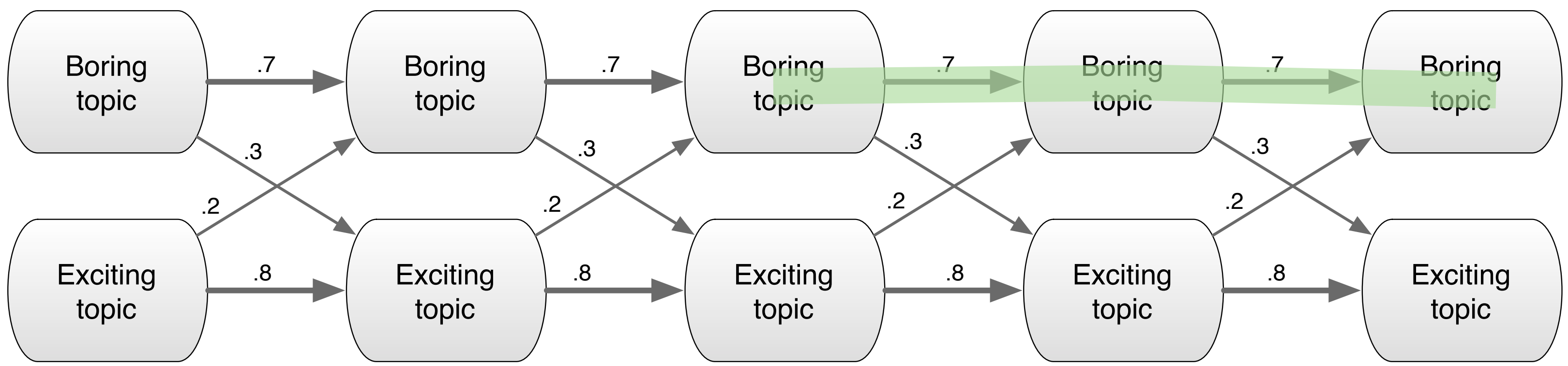


Interesting



So-so

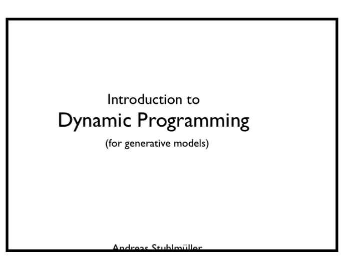
Forward Algorithm



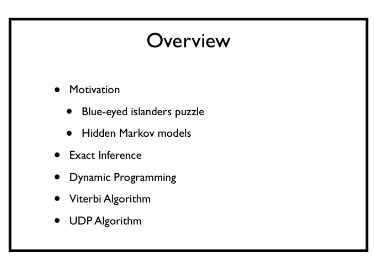
$$O(N|Q|^2)$$

$$P(o_{1:i}, q) = \begin{cases} P_{init}(q)P(o_1|q) & \text{if } i = 1 \\ \sum_{r \in Q} P(o_{1:(i-1)}, r)P(q|r)P(o_i|q) & \text{otherwise} \end{cases}$$

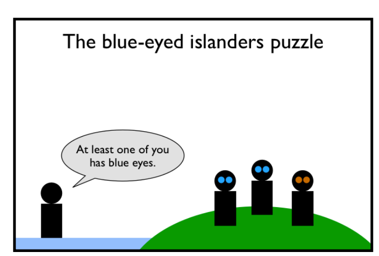
$$P(o_{1:i}) = \sum_{q \in Q} P(o_{1:i}, q)$$



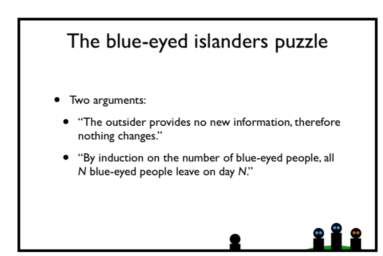
So-so



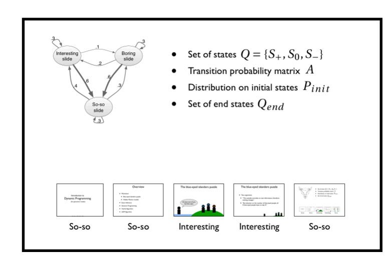
So-so



Interesting

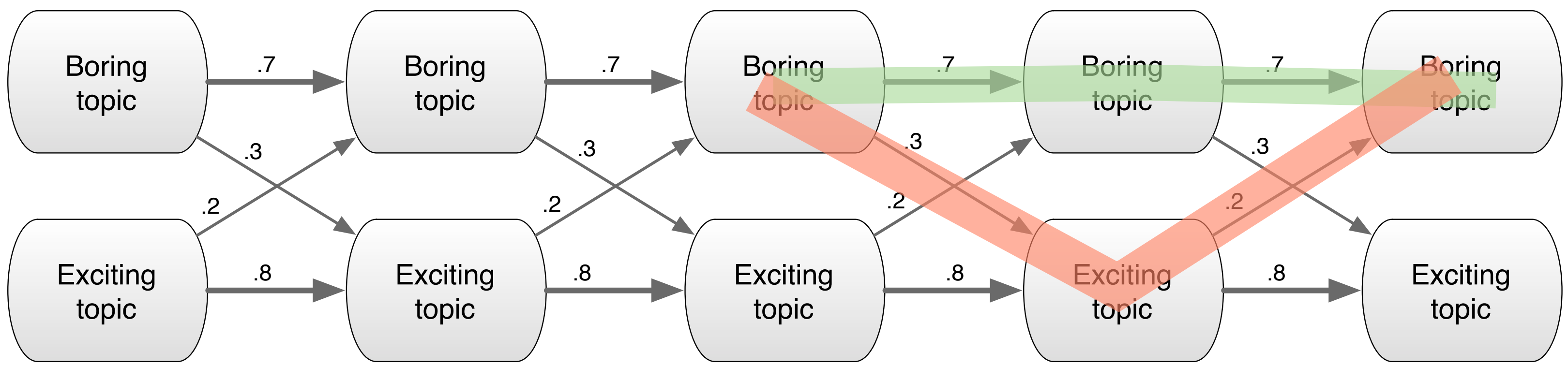


Interesting



So-so

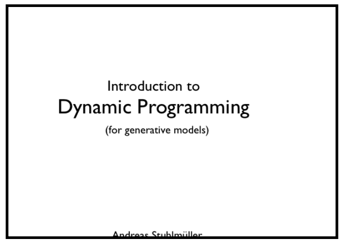
Forward Algorithm



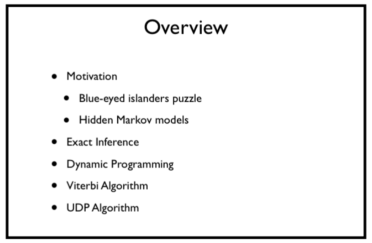
$O(N|Q|^2)$

$$P(o_{1:i}, q) = \begin{cases} P_{init}(q)P(o_1|q) & \text{if } i = 1 \\ \sum_{r \in Q} P(o_{1:(i-1)}, r)P(q|r)P(o_i|q) & \text{otherwise} \end{cases}$$

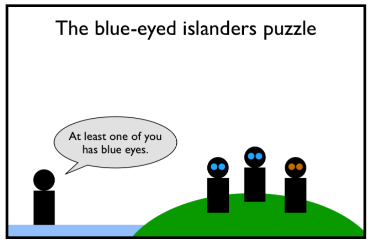
$$P(o_{1:i}) = \sum_{q \in Q} P(o_{1:i}, q)$$



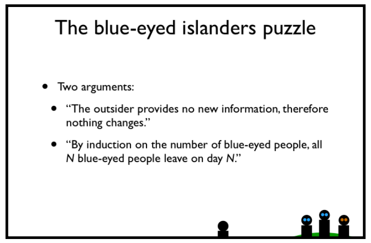
So-so



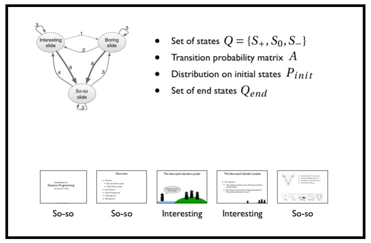
So-so



Interesting

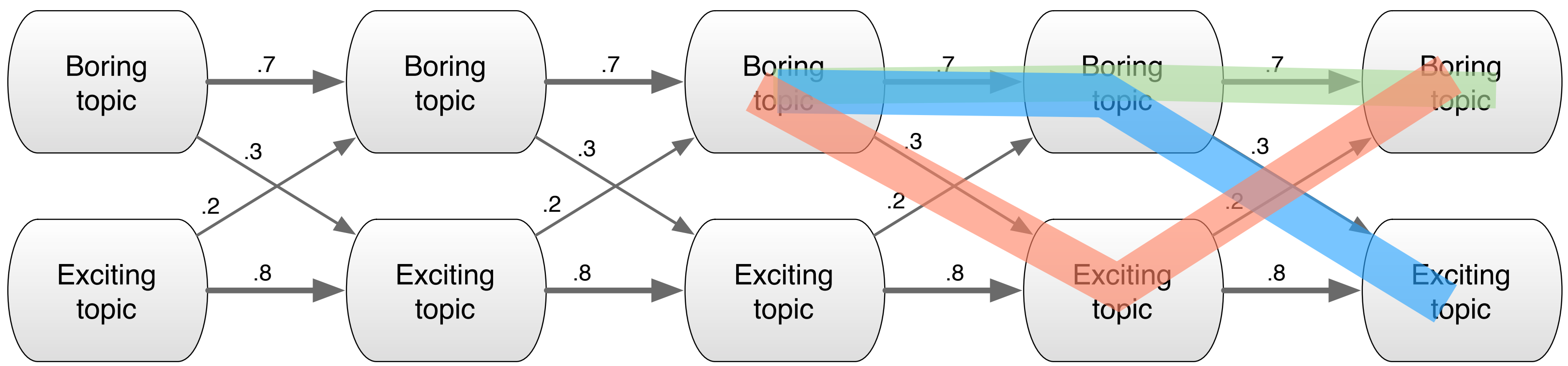


Interesting



So-so

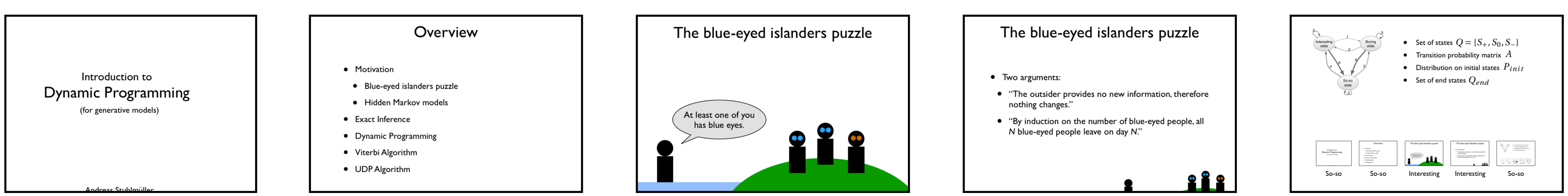
Forward Algorithm



$O(N|Q|^2)$

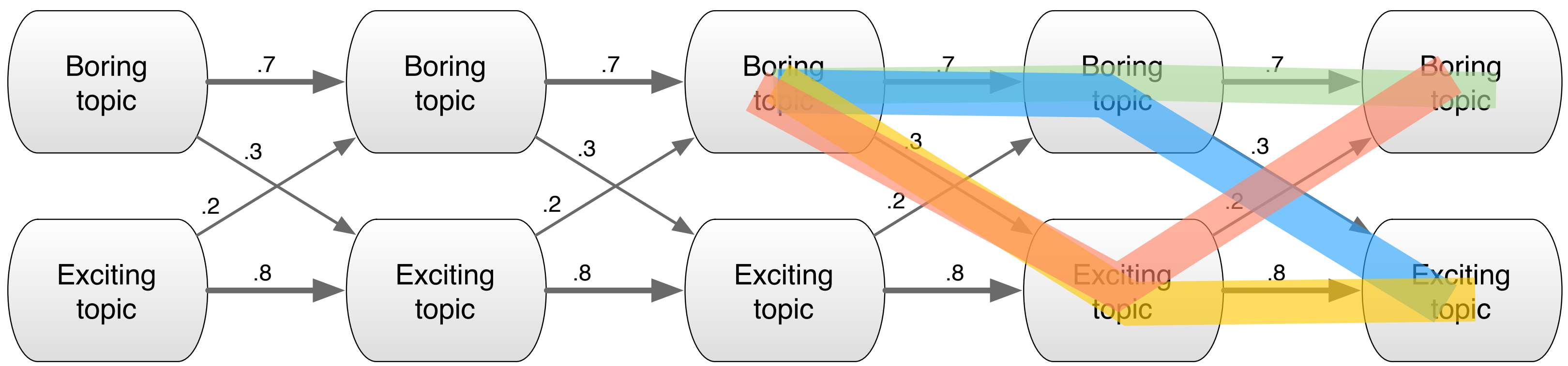
$$P(o_{1:i}, q) = \begin{cases} P_{init}(q)P(o_1|q) & \text{if } i = 1 \\ \sum_{r \in Q} P(o_{1:(i-1)}, r)P(q|r)P(o_i|q) & \text{otherwise} \end{cases}$$

$$P(o_{1:i}) = \sum_{q \in Q} P(o_{1:i}, q)$$



So-so   So-so   Interesting   Interesting   So-so

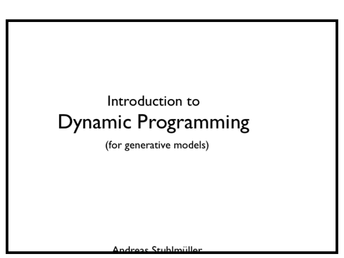
Forward Algorithm



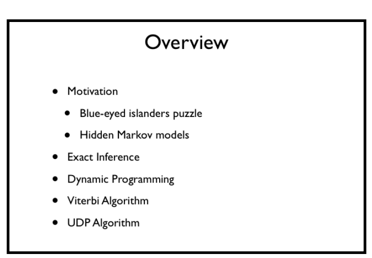
$$O(N|Q|^2)$$

$$P(o_{1:i}, q) = \begin{cases} P_{init}(q)P(o_1|q) & \text{if } i = 1 \\ \sum_{r \in Q} P(o_{1:(i-1)}, r)P(q|r)P(o_i|q) & \text{otherwise} \end{cases}$$

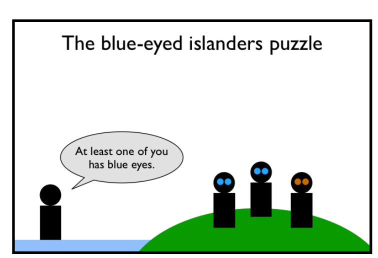
$$P(o_{1:i}) = \sum_{q \in Q} P(o_{1:i}, q)$$



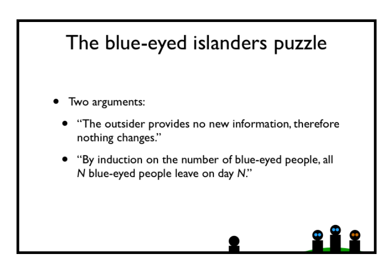
So-so



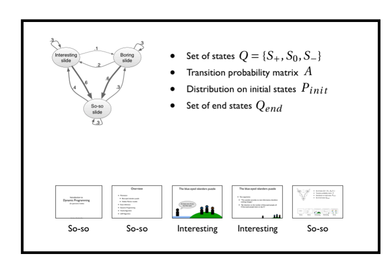
So-so



Interesting

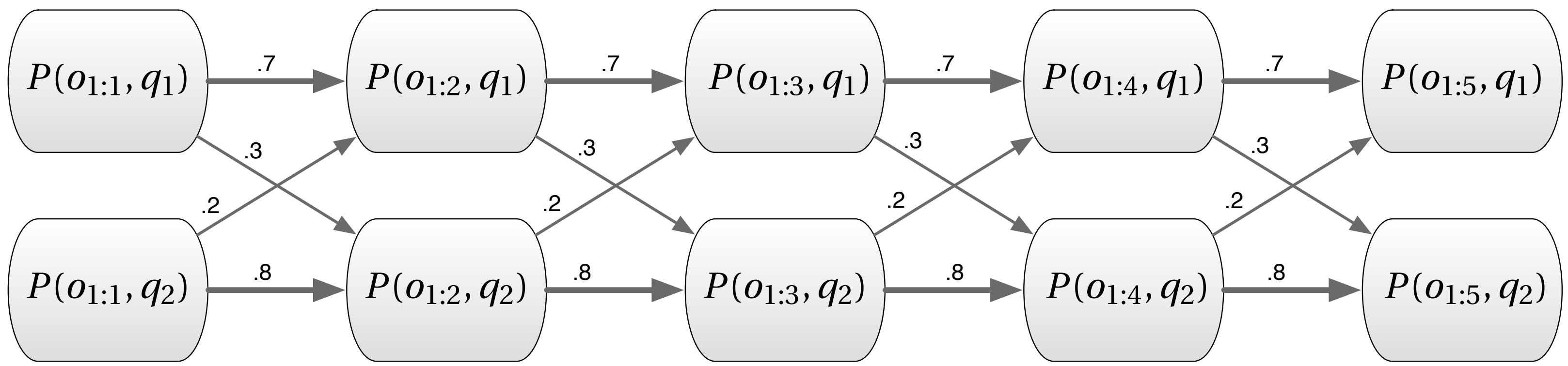


Interesting



So-so

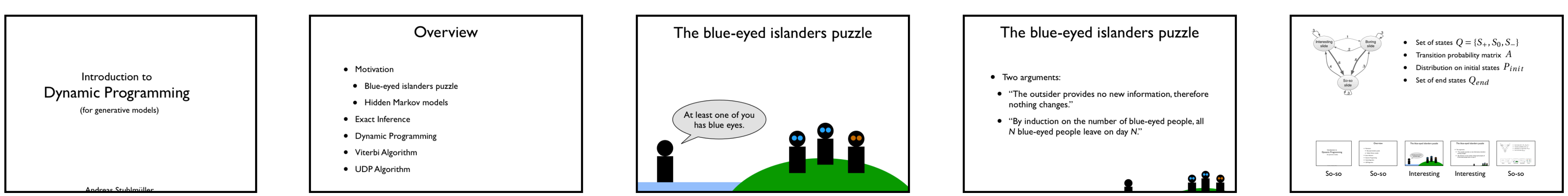
Forward Algorithm



$O(N|Q|^2)$

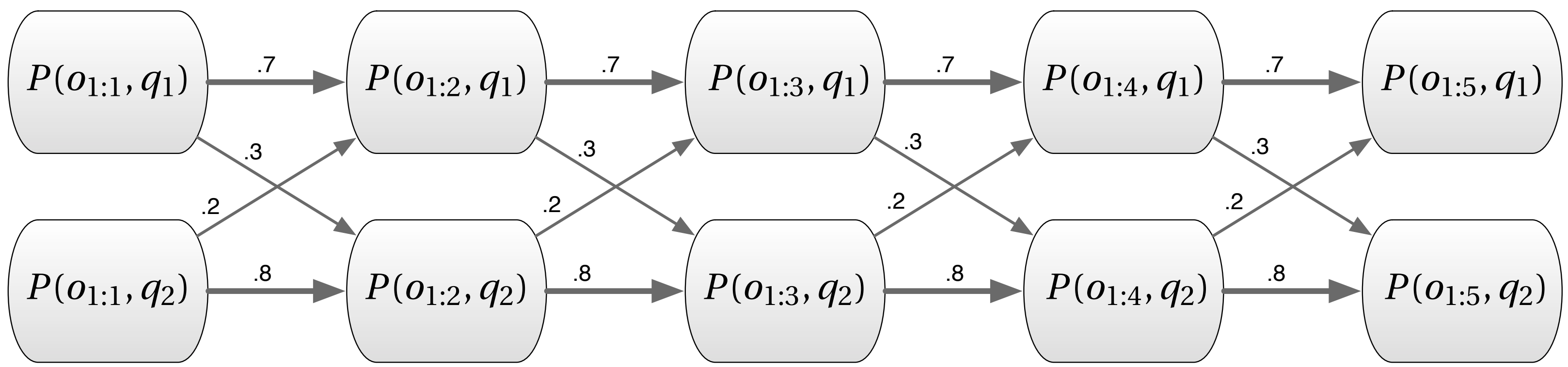
$$P(o_{1:i}, q) = \begin{cases} P_{init}(q)P(o_1|q) & \text{if } i = 1 \\ \sum_{r \in Q} P(o_{1:(i-1)}, r)P(q|r)P(o_i|q) & \text{otherwise} \end{cases}$$

$$P(o_{1:i}) = \sum_{q \in Q} P(o_{1:i}, q)$$



So-so So-so Interesting Interesting So-so

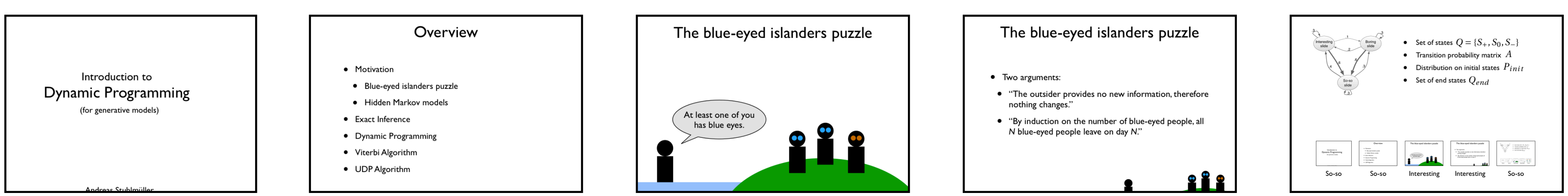
Viterbi Algorithm



$O(N|Q|^2)$

$$P_{max}(o_{1:i}, q) = \begin{cases} P_{init}(q)P(o_1|q) & \text{if } i = 1 \\ \max_{r \in Q} P_{max}(o_{1:(i-1)}, r)P(q|r)P(o_i|q) & \text{otherwise} \end{cases}$$

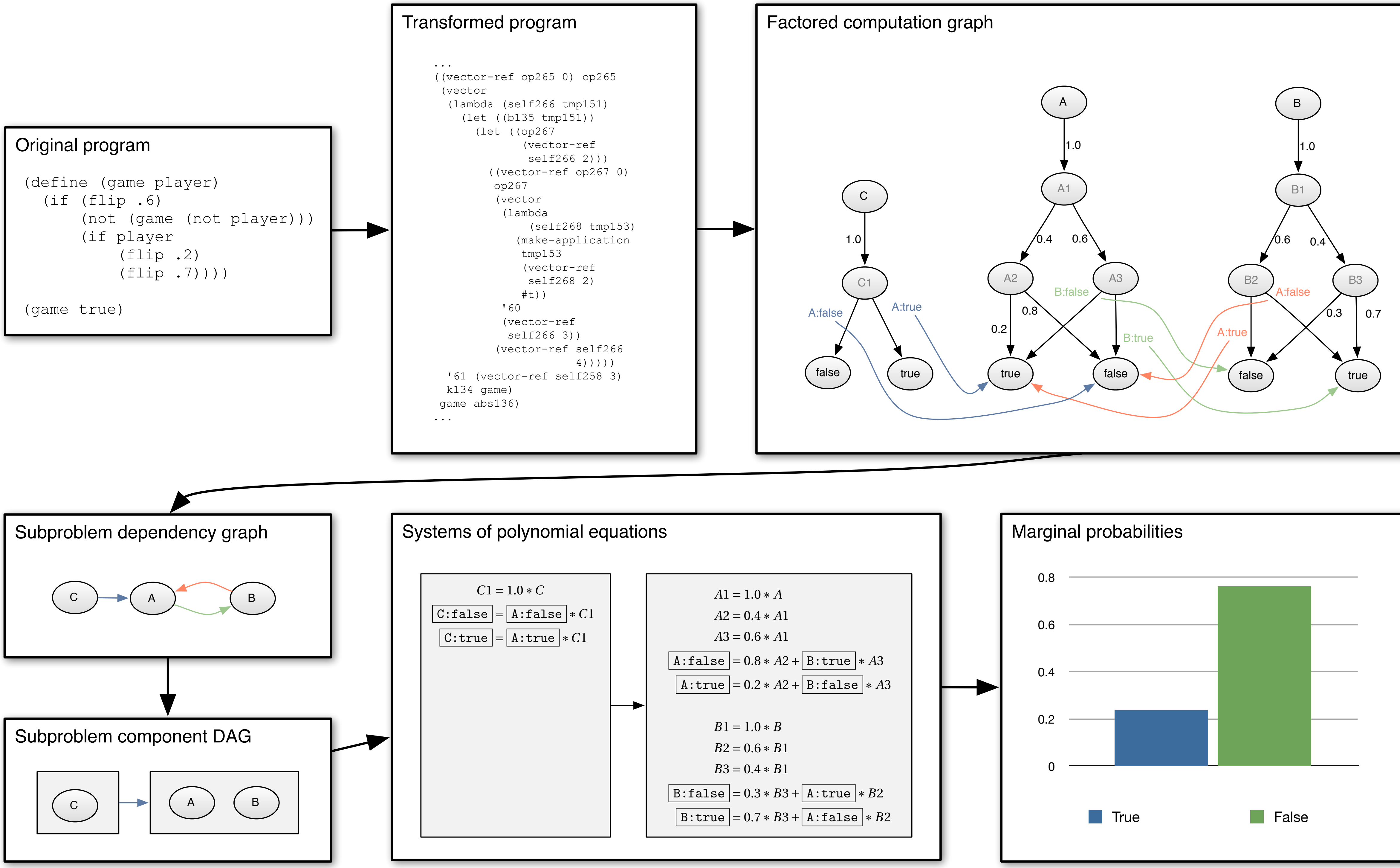
$$P_{max}(o_{1:i}) = \max_{q \in Q} P_{max}(o_{1:i}, q)$$



So-so | So-so | Interesting | Interesting | So-so

Dynamic Programming Algorithms

- Many DP algorithms exist ...
  - HMMs: Forward, Viterbi, Forward-Backward
  - PCFGs: Earley, CYK
  - MDPs: Bellman backups
- ... but probabilistic programming allows us to write many more models without known solutions
- Goal: Universal DP to make inference in some of these models tractable

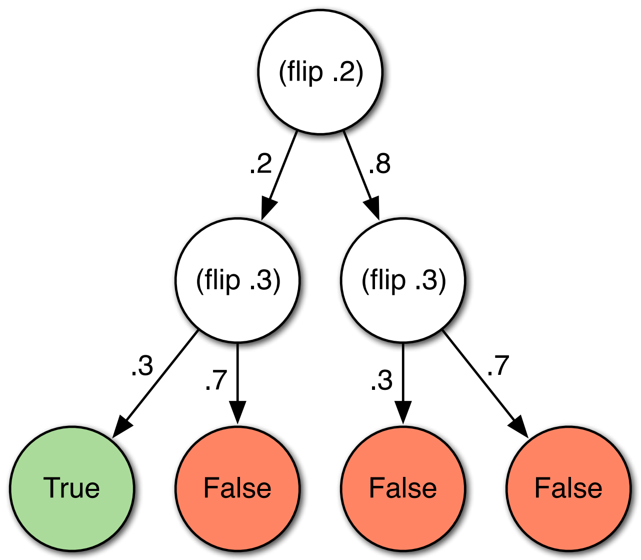


(Stuhlmüller, Roy, O'Donnell, and Goodman, under review)

DP for probabilistic programs

- Enumeration

```
(define (foo)
  (and (flip .2)
       (flip .3)))

(foo)
```

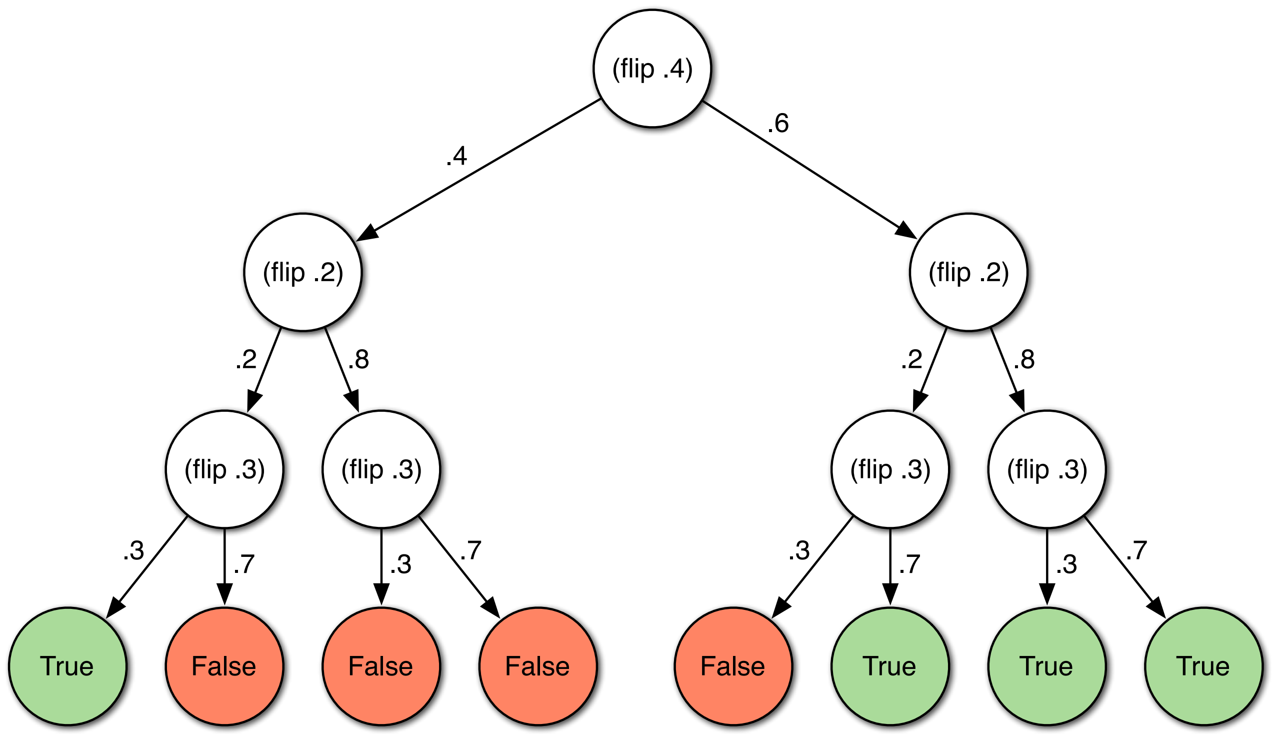


- Factorization
- State merging
- Challenge: Recursive non-tail dependencies

DP for probabilistic programs

- Enumeration
- Factorization

```
(define (outer)
  (if (flip .4)
      (inner)
      (not (inner))))

(define (inner)
  (and (flip .2)
       (flip .3)))

(outer)
```

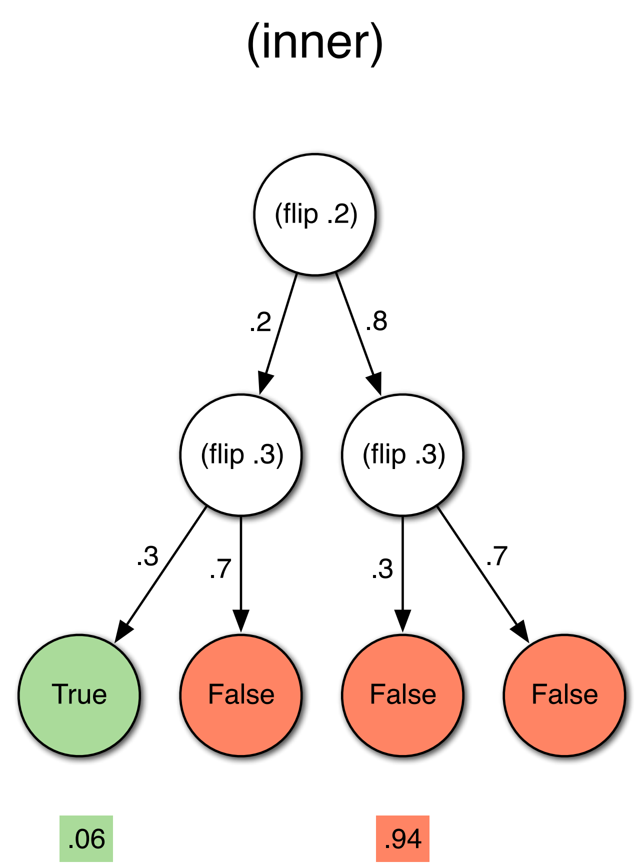


- State merging
- Challenge: Recursive non-tail dependencies

DP for probabilistic programs

- Enumeration
- Factorization

```
(define (outer)
  (if (flip .4)
      (inner)
      (not (inner))))

(define (inner)
  (and (flip .2)
       (flip .3)))

(outer)
```

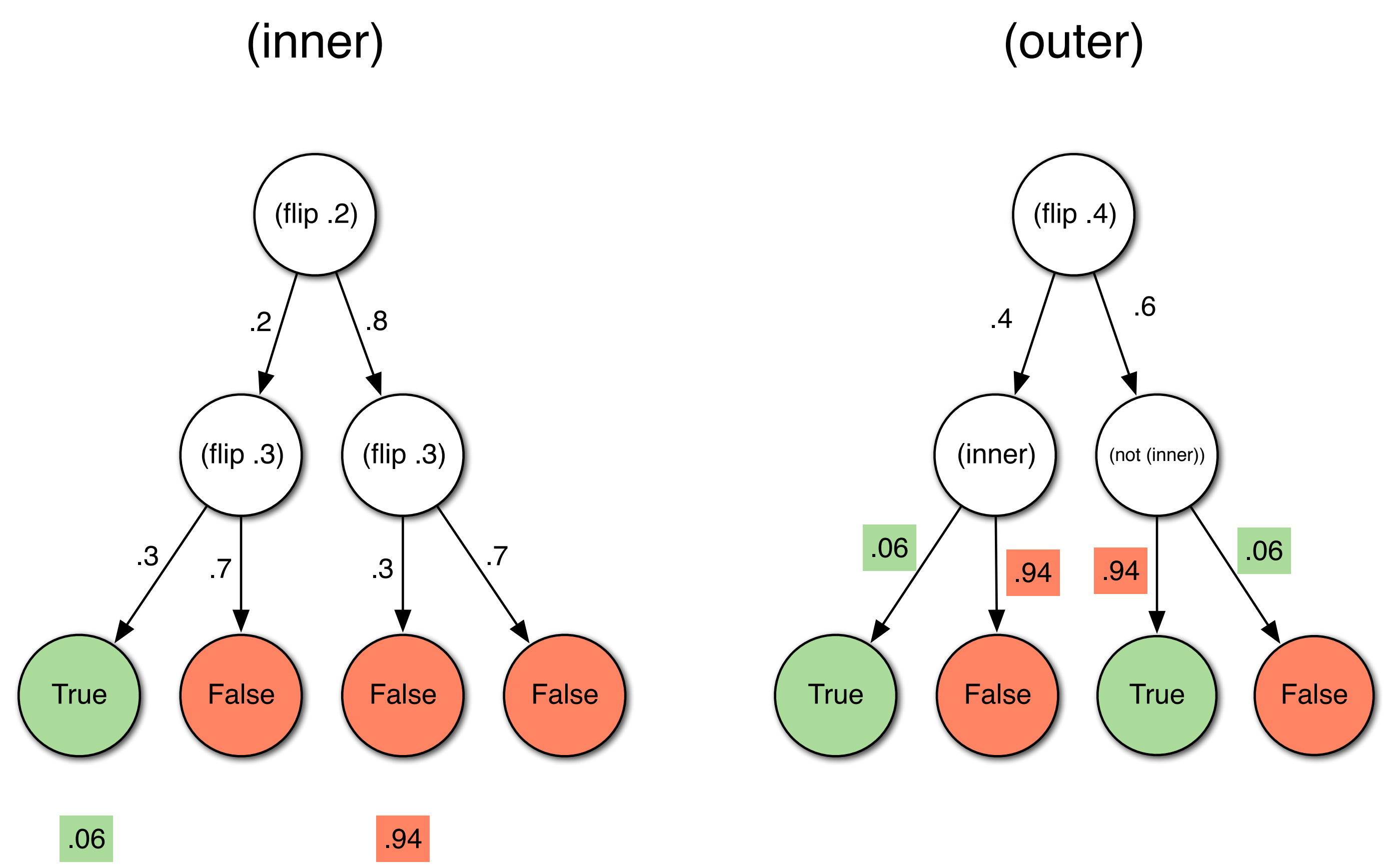


- State merging
- Challenge: Recursive non-tail dependencies

DP for probabilistic programs

- Enumeration
- Factorization

```
(define (outer)
  (if (flip .4)
      (inner)
      (not (inner))))

(define (inner)
  (and (flip .2)
       (flip .3)))

(outer)
```

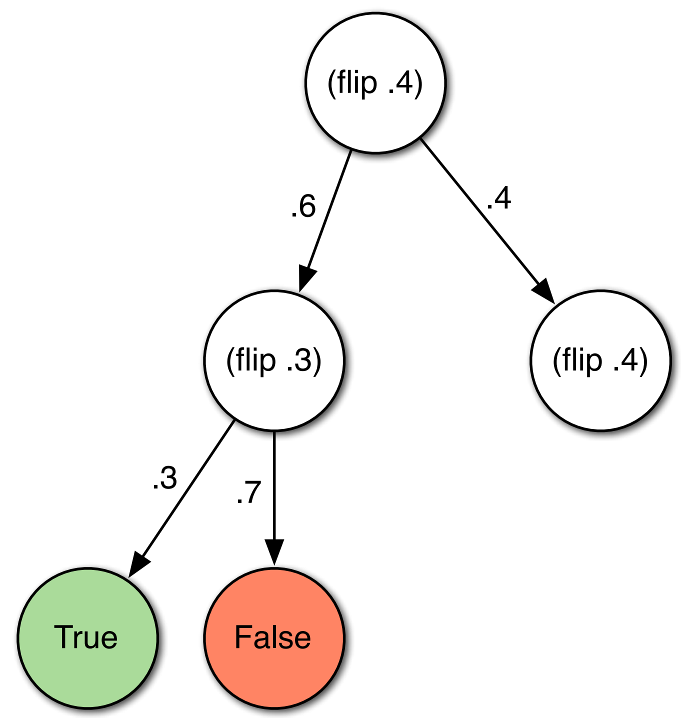


- State merging
- Challenge: Recursive non-tail dependencies

DP for probabilistic programs

- Enumeration
- Factorization
- State merging

```
(define (tail)
  (if (flip .4)
      (tail)
      (flip .3)))

(tail)
```

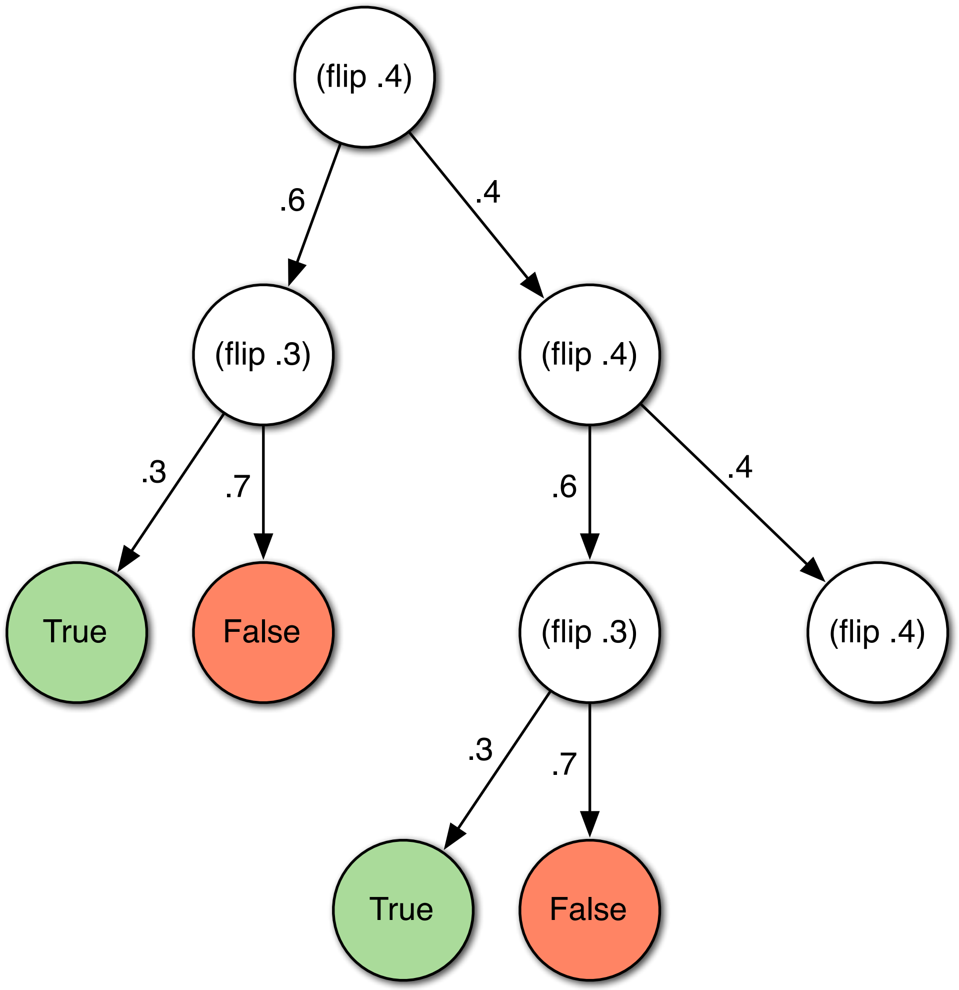


- Challenge: Recursive non-tail dependencies

DP for probabilistic programs

- Enumeration
- Factorization
- State merging

```
(define (tail)
  (if (flip .4)
      (tail)
      (flip .3)))

(tail)
```

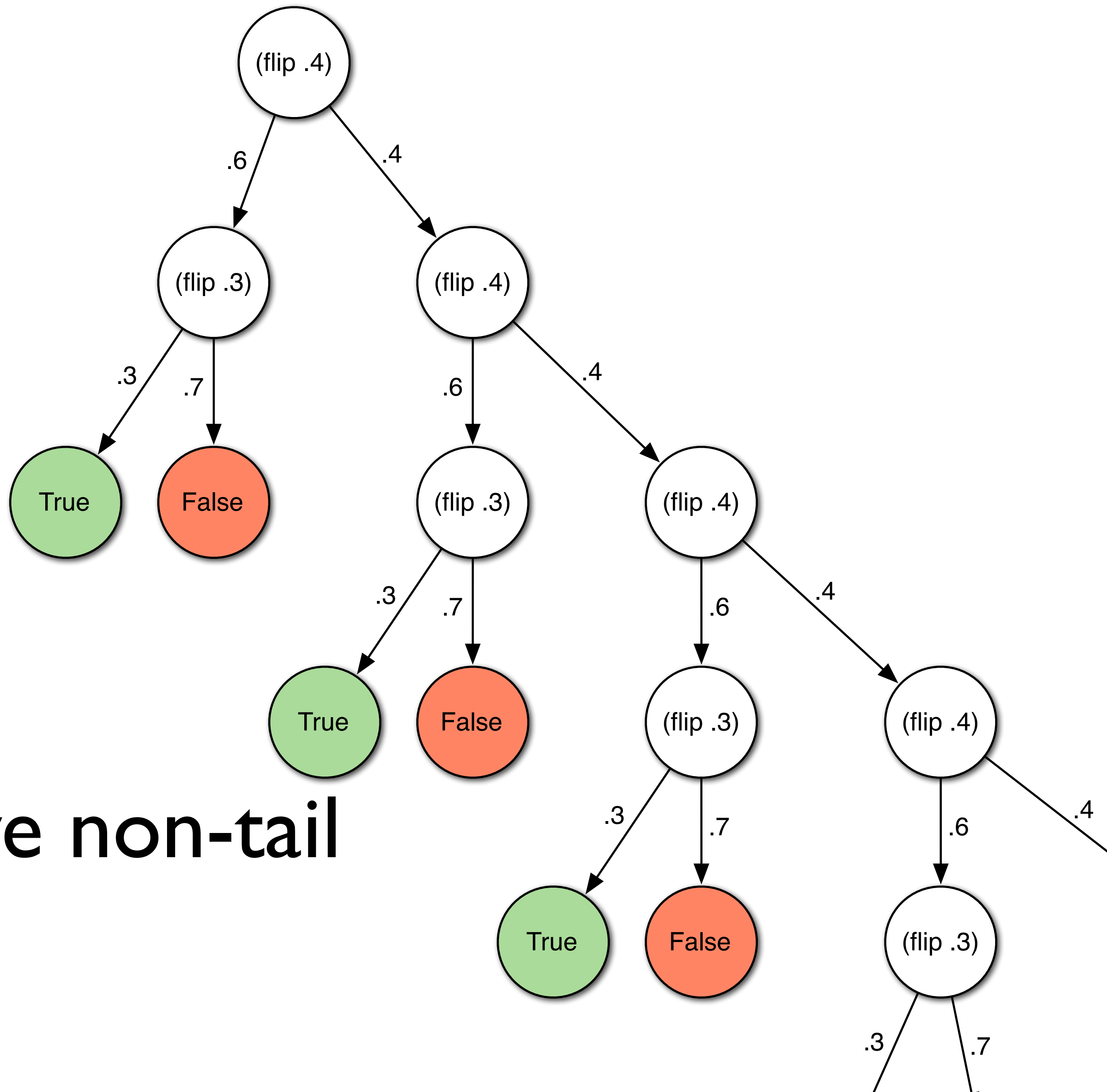


- Challenge: Recursive non-tail dependencies

DP for probabilistic programs

- Enumeration
- Factorization
- State merging

```
(define (tail)
  (if (flip .4)
      (tail)
      (flip .3)))

(tail)
```

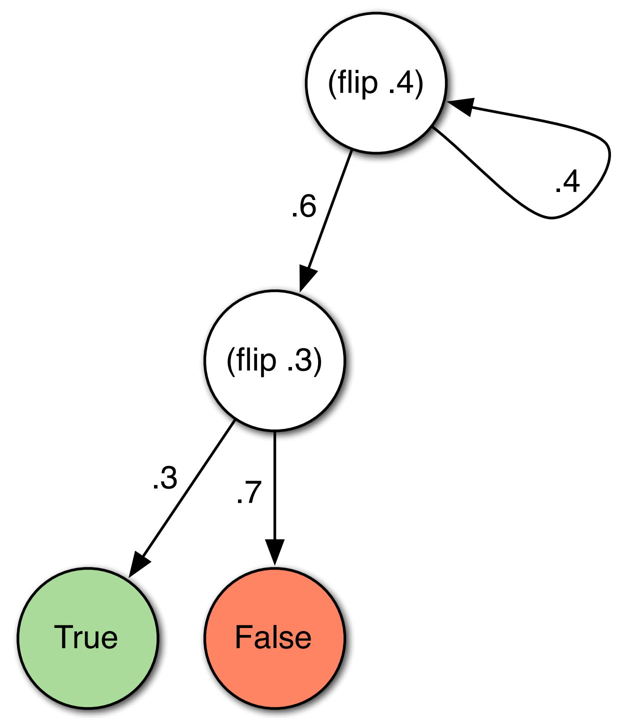


- Challenge: Recursive non-tail dependencies

DP for probabilistic programs

- Enumeration
- Factorization
- State merging

```
(define (tail)
  (if (flip .4)
      (tail)
      (flip .3)))

(tail)
```

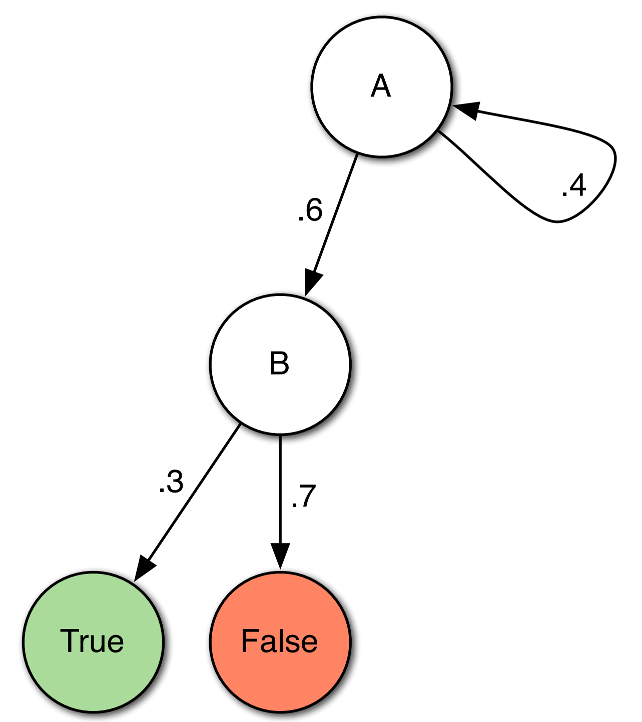


- Challenge: Recursive non-tail dependencies

DP for probabilistic programs

- Enumeration
- Factorization
- State merging

```
(define (tail)
  (if (flip .4)
      (tail)
      (flip .3)))

(tail)
```

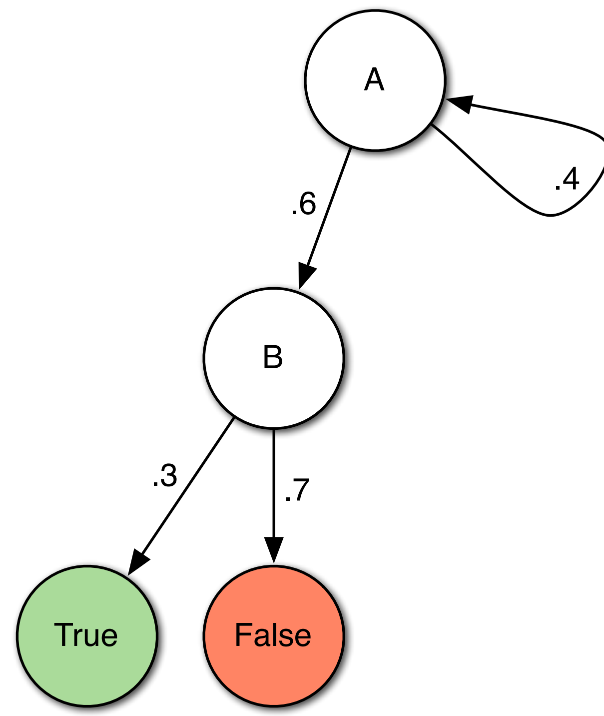


- Challenge: Recursive non-tail dependencies

DP for probabilistic programs

- Enumeration
- Factorization
- State merging

```
(define (tail)
  (if (flip .4)
      (tail)
      (flip .3)))

(tail)
```

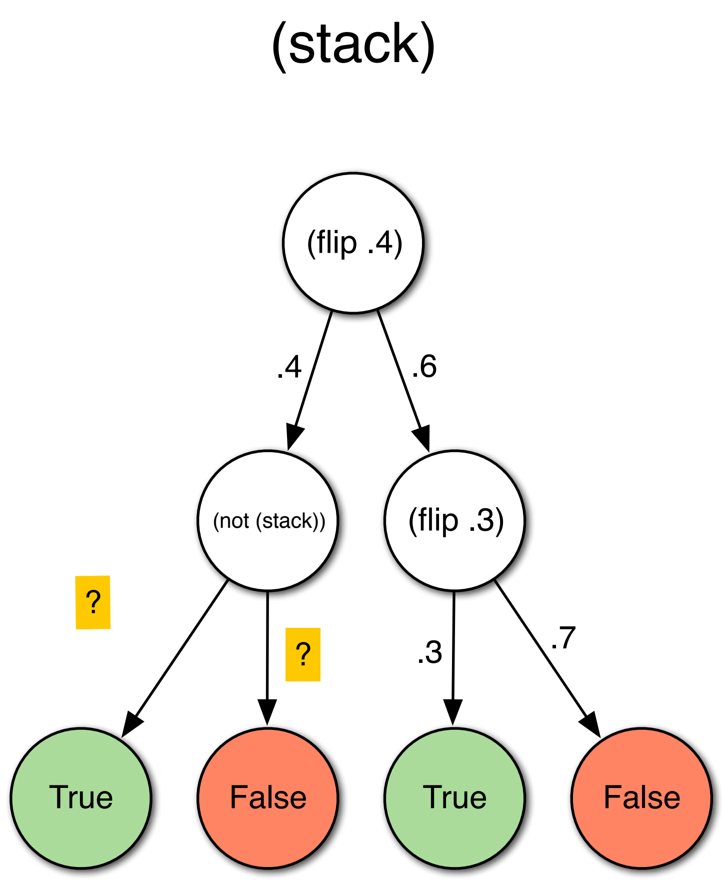


$$A = 1 + .4 * A$$
$$B = .6 * A$$
$$\text{True} = .3 * B$$
$$\text{False} = .7 * B$$

- Challenge: Recursive non-tail dependencies

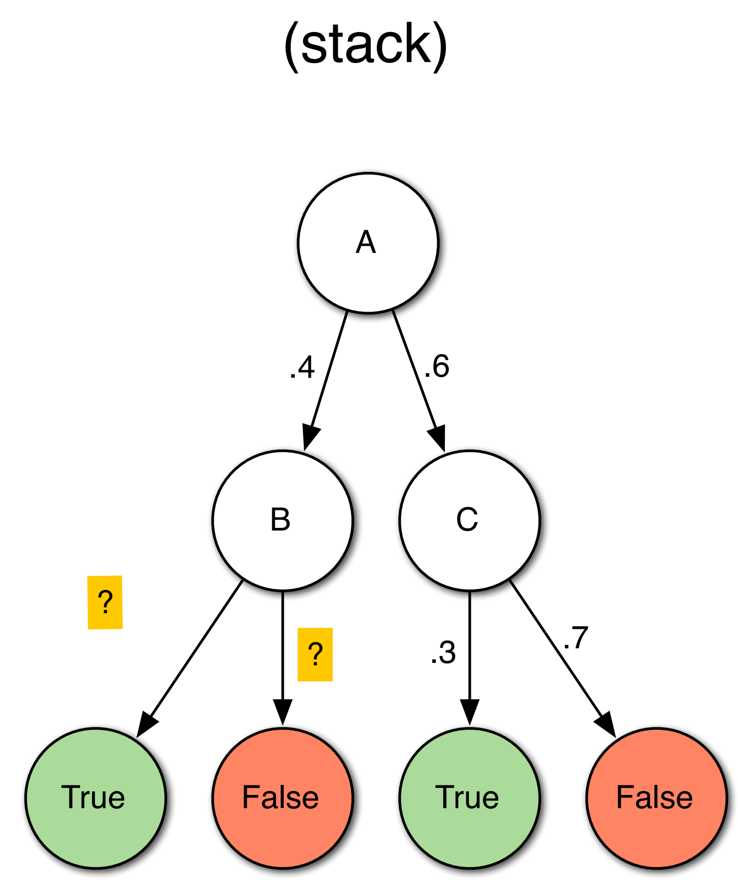
Challenge: Recursive dependencies

```
(define (stack)
  (if (flip .4)
      (not (stack))
      (flip .3)))

(stack)
```



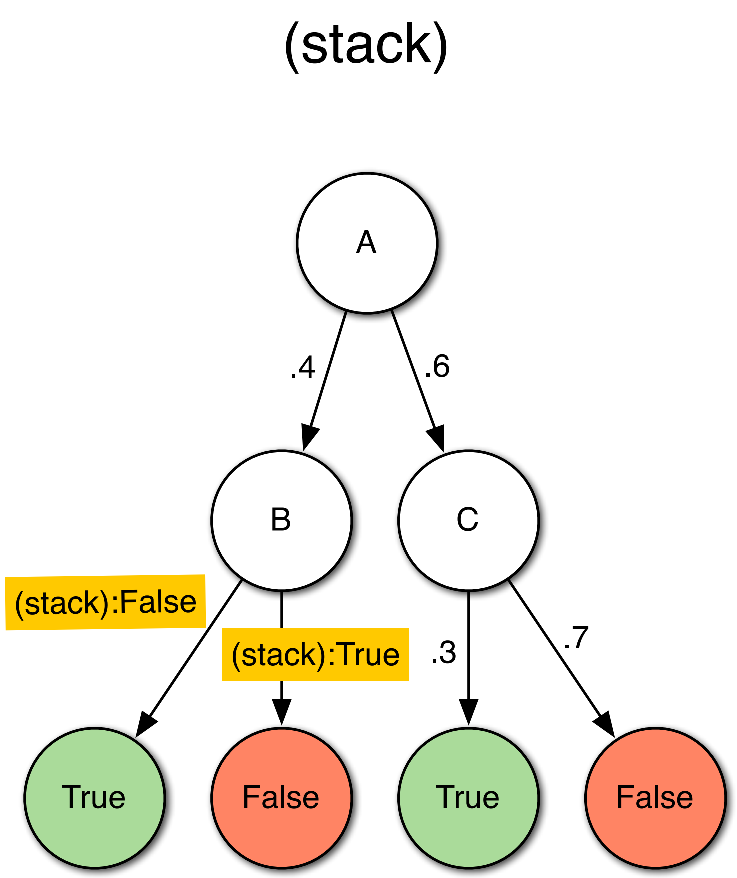
Challenge: Recursive dependencies

```
(define (stack)
  (if (flip .4)
      (not (stack))
      (flip .3)))

(stack)
```



Challenge: Recursive dependencies

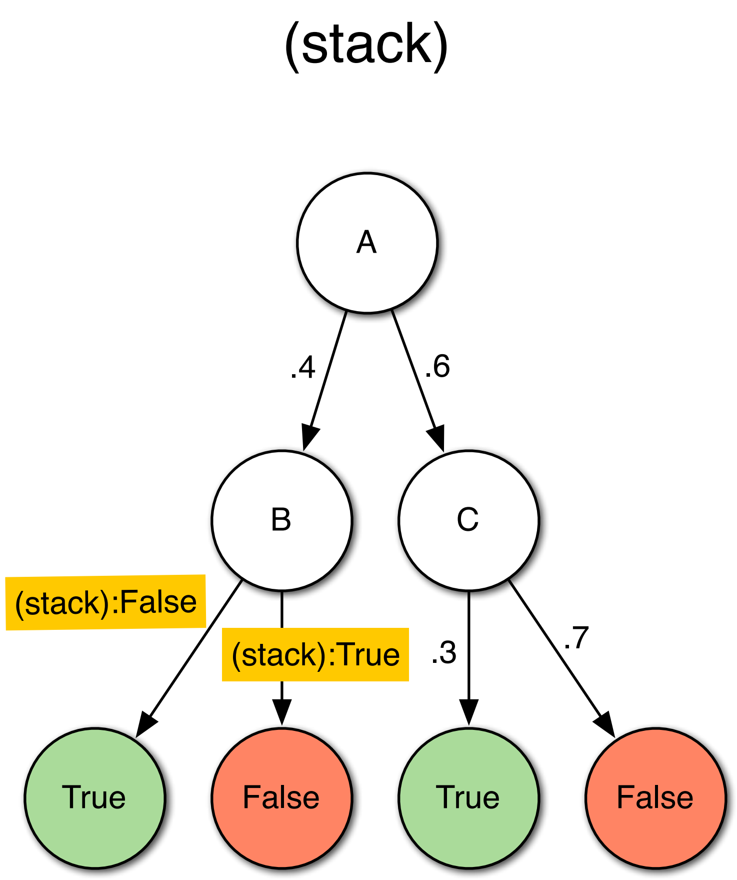
```
(define (stack)
  (if (flip .4)
      (not (stack))
      (flip .3)))

(stack)
```

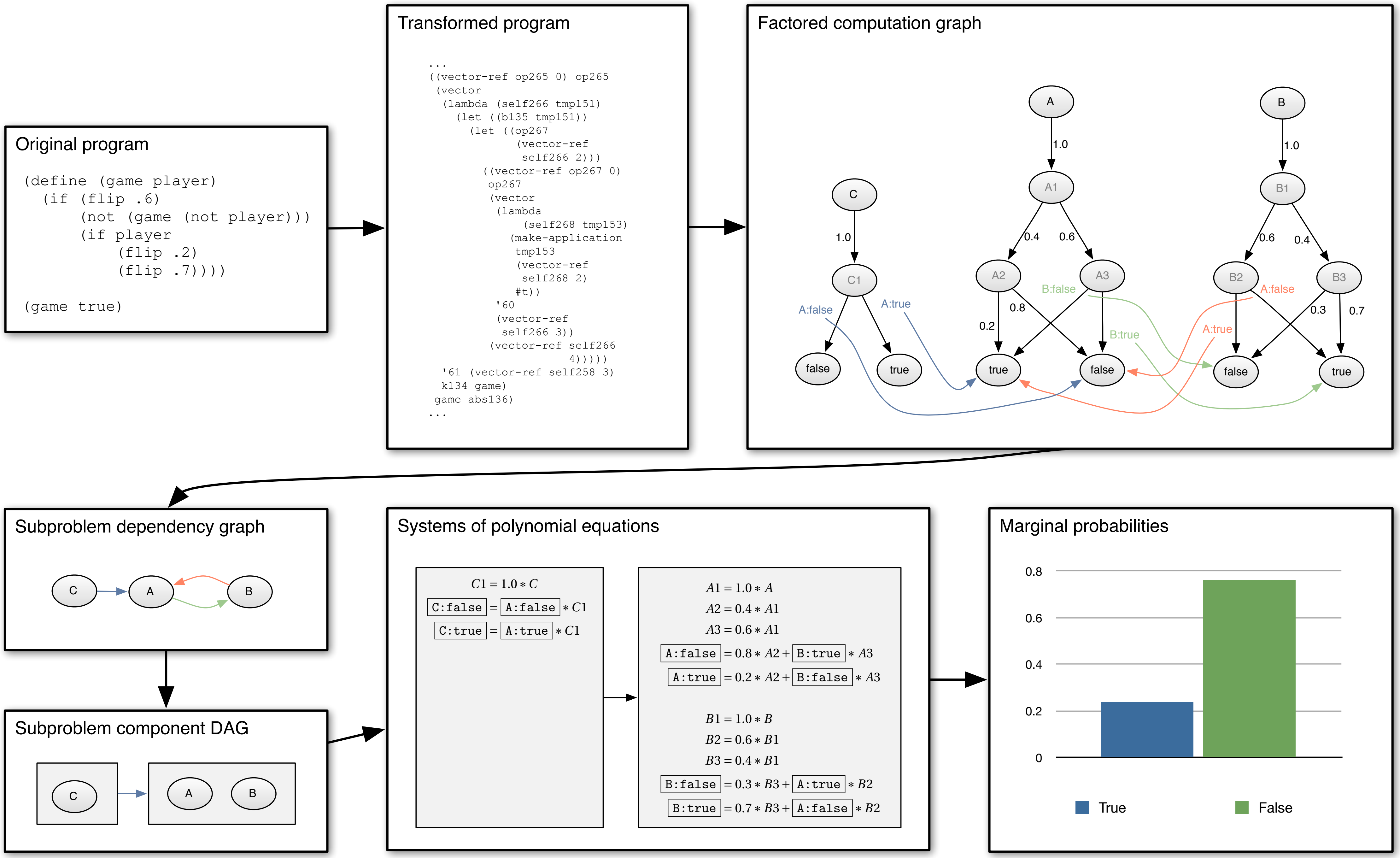


Challenge: Recursive dependencies

```
(define (stack)
  (if (flip .4)
      (not (stack))
      (flip .3)))

(stack)
```

$$A = 1$$
$$B = .4 * A$$
$$C = .6 * A$$
$$(\text{stack}) : \text{True} = (\text{stack}) : \text{False} * B + .3 * C$$
$$(\text{stack}) : \text{False} = (\text{stack}) : \text{True} * B + .7 * C$$




Original program
(define (game player)
(if (flip .6)
(not (game (not player)))
(if player
(flip .2)
(flip .7))))
(game true)
Transformed program
...
((vector-ref op265 0) op265
(vector
(lambda (self266 tmp151)
(let ((b135 tmp151))
(let ((op267
(vector-ref
self266 2)))
((vector-ref op267 0)
op267
(vector
(lambda
(self268 tmp153)
(make-application
tmp153
(vector-ref
self268 2)
#t))
'60
(vector-ref
self266 3))
(vector-ref self266
4)))))
'61 (vector-ref self258 3)
k134 game)
game abs136)
...
Factored computation graph
A
B
C
1.0
A1
B1
C1
0.4
0.6
A2
A3
B2
B3
A:false
A:true
B:false
B:true
0.2
0.8
0.3
0.7
false
true
Subproblem dependency graph
Subproblem component DAG
Systems of polynomial equations
C1 = 1.0 * C
C:false = A:false * C1
C:true = A:true * C1
A1 = 1.0 * A
A2 = 0.4 * A1
A3 = 0.6 * A1
A:false = 0.8 * A2 + B:true * A3
A:true = 0.2 * A2 + B:false * A3
B1 = 1.0 * B
B2 = 0.6 * B1
B3 = 0.4 * B1
B:false = 0.3 * B3 + A:true * B2
B:true = 0.7 * B3 + A:false * B2
Marginal probabilities
0.8
0.6
0.4
0.2
0
True
False

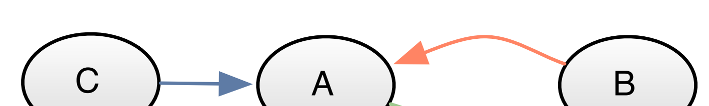
Original program

```
(define (game player)
  (if (flip .6)
      (not (game (not player)))
      (if player
          (flip .2)
          (flip .7)))))

(game true)
```

Transformed program

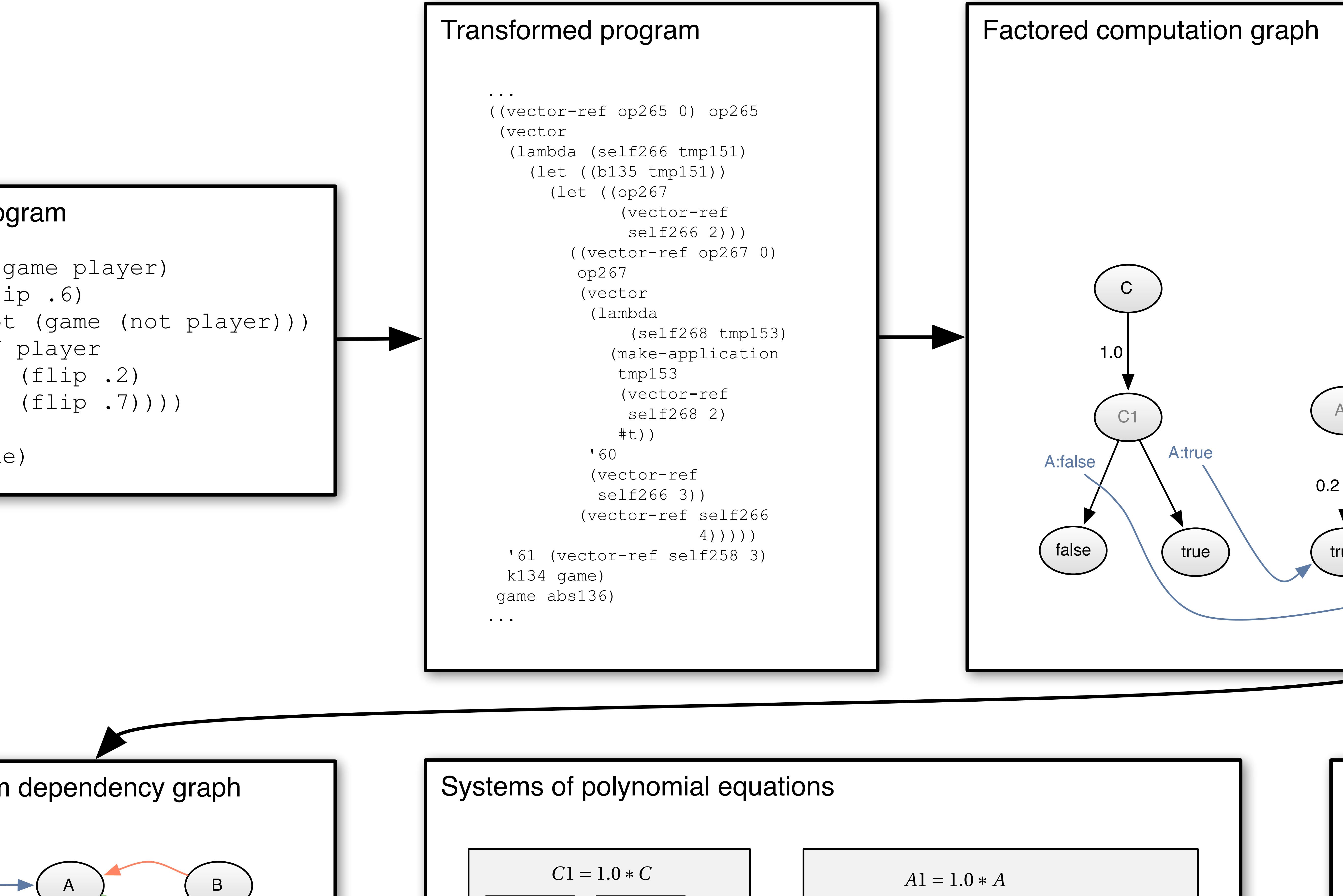
```
...
((vector-ref op265 0) op265
 (vector
  (lambda (self266 tmp151)
    (let ((b135 tmp151))
      (let ((op267
              (vector-ref
               self266 2)))
        ((vector-ref op267 0)
         op267
         (vector
          (lambda
              (self268 tmp153)
            (make-application
             tmp153
             (vector-ref
              self268 2)
             #t))
          '60
          (vector-ref
           self266 3))
         (vector-ref self266
                     4)))))
  '61 (vector-ref self258 3)
  k134 game)
 game abs136)
...
```

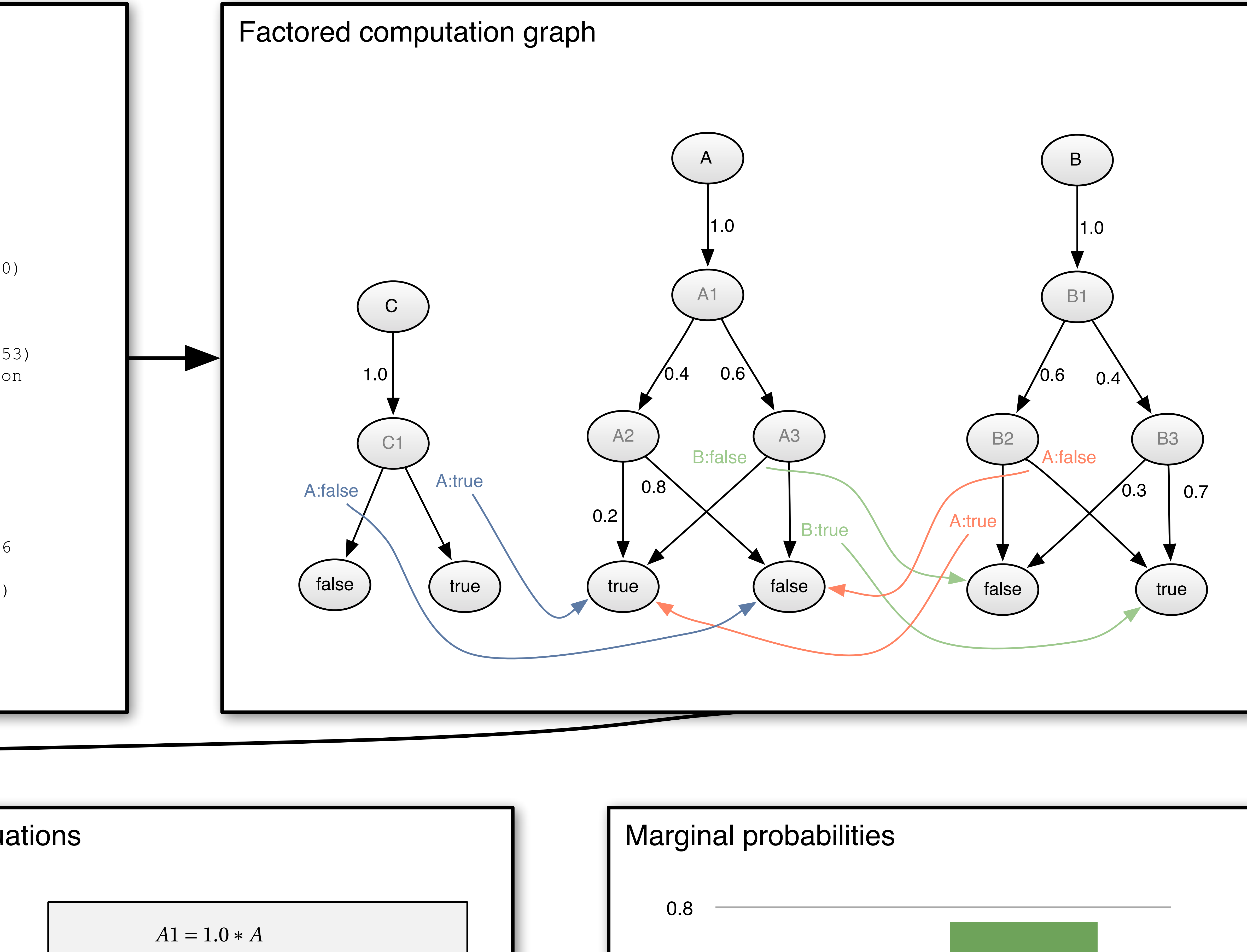
Subproblem dependency graph

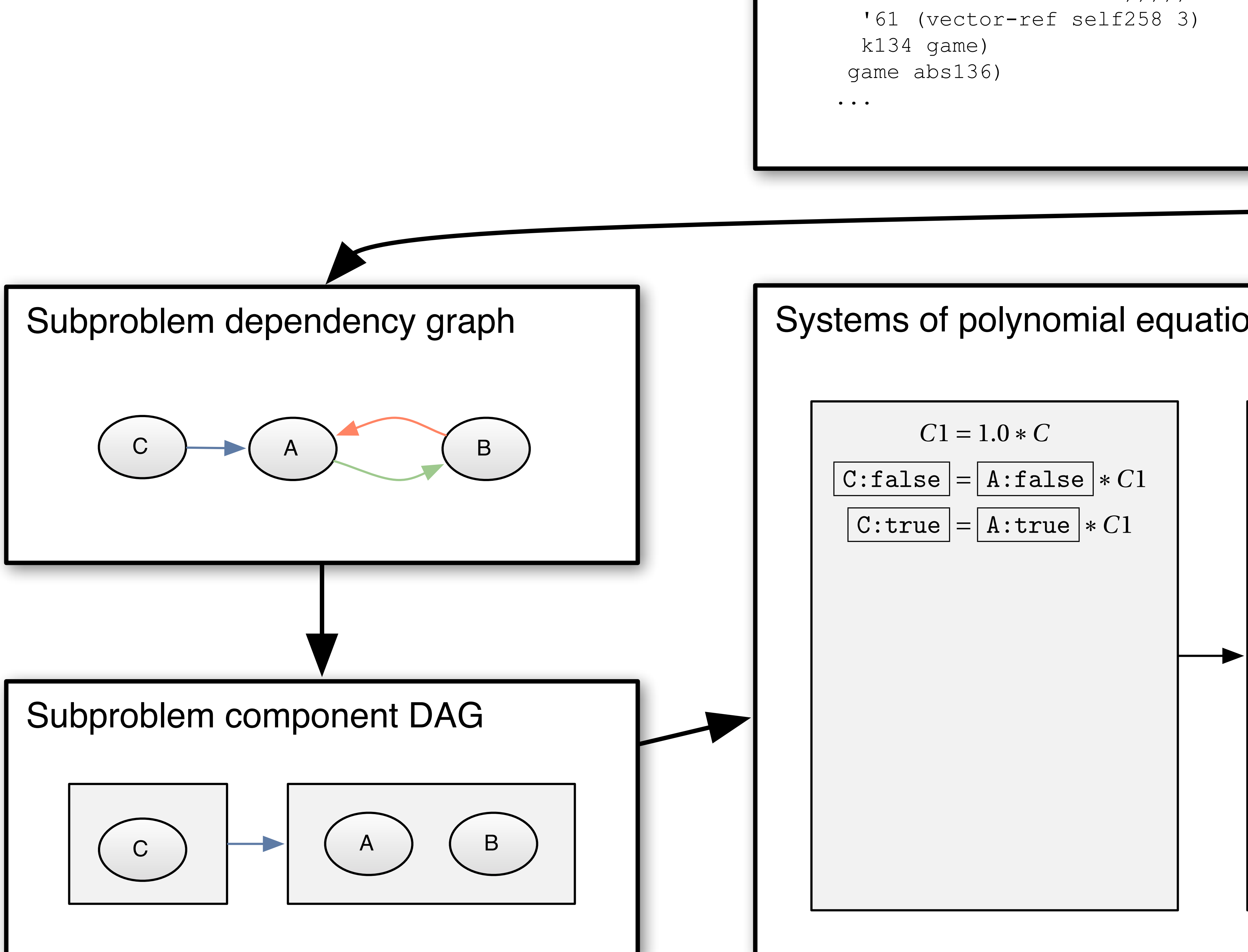


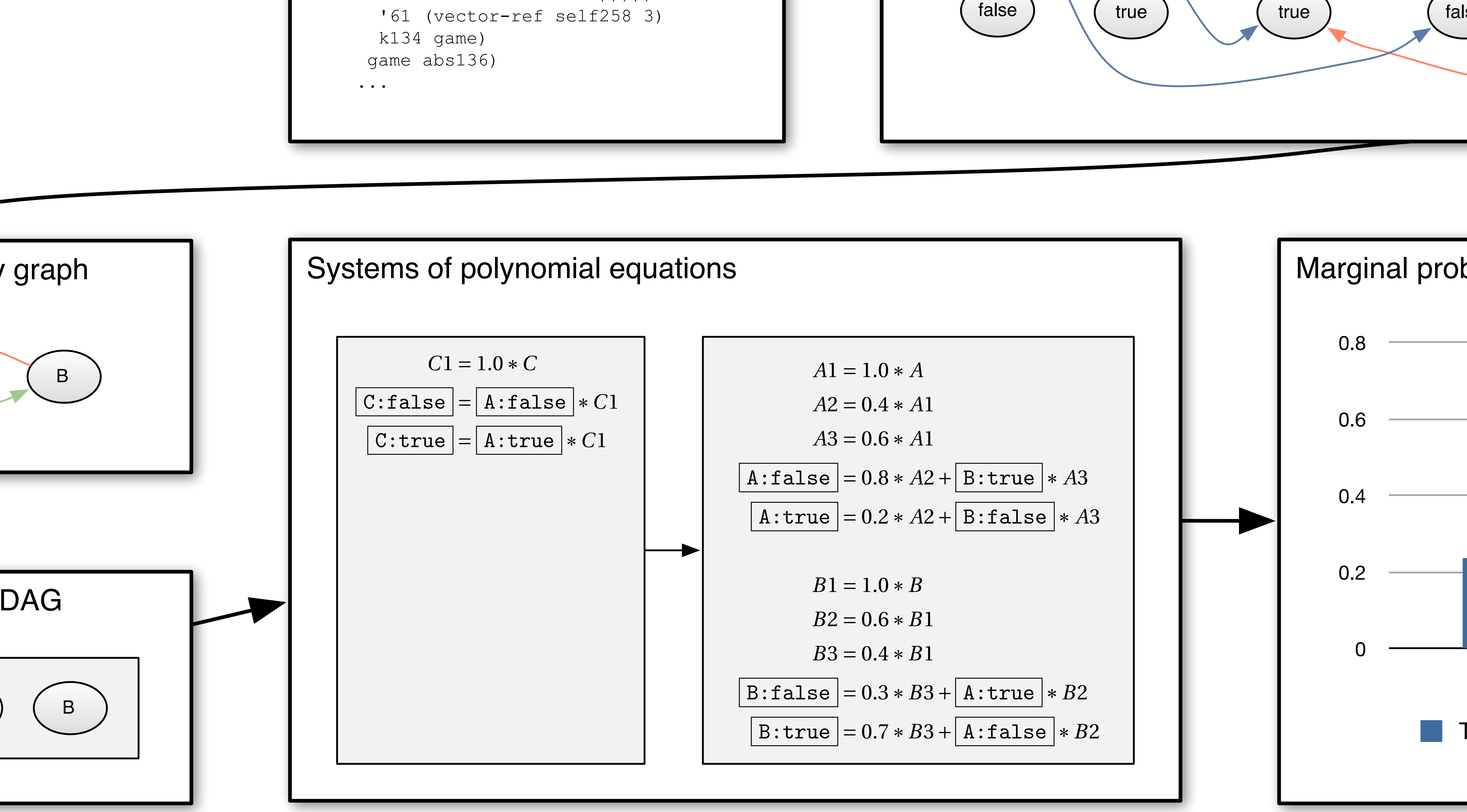
Systems of polynomial equations

$$C1 = 1.0 * C$$

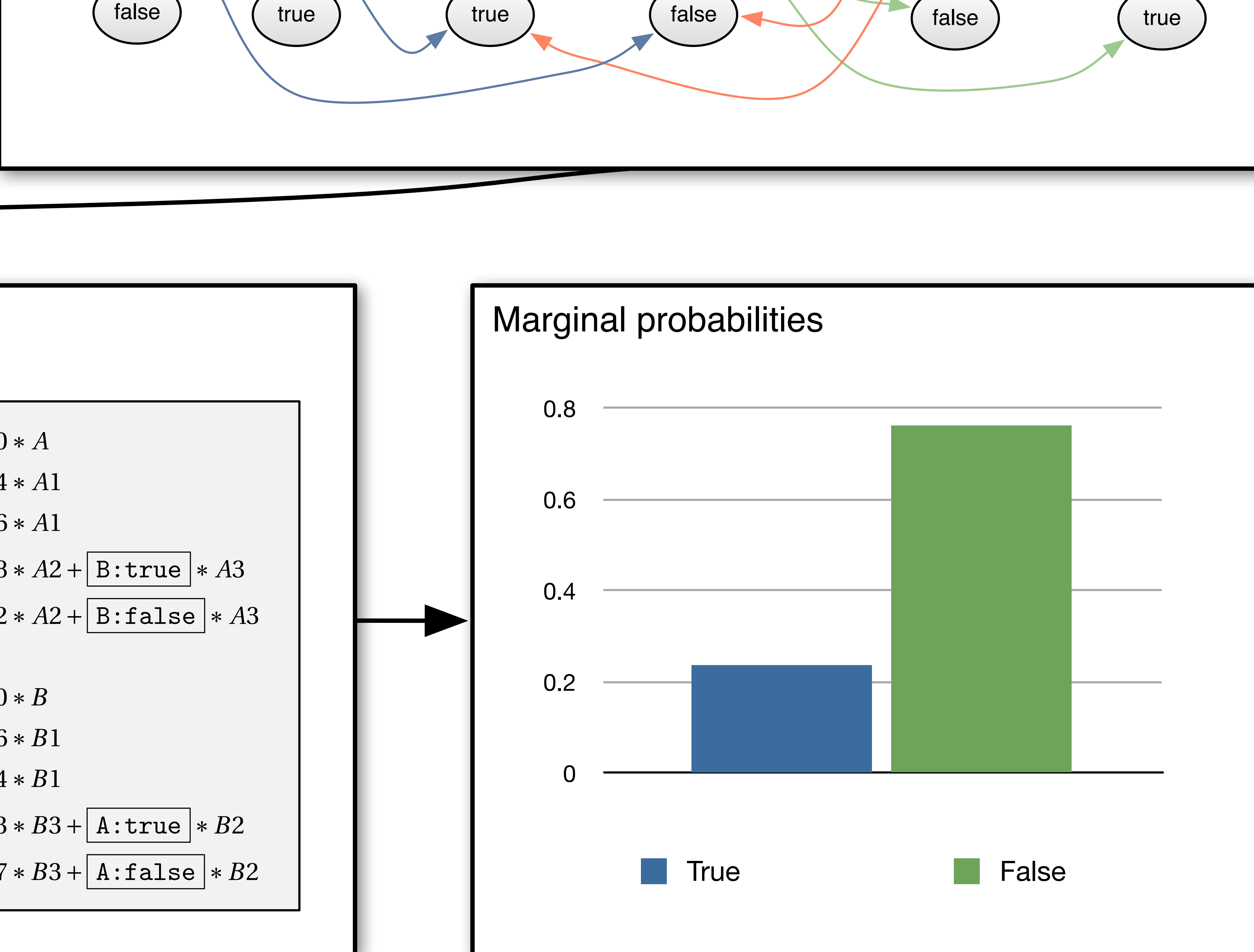

ogram
game player)
ip .6)
t (game (not player)))
player
(flip .2)
(flip .7))))
e)
Transformed program
...
((vector-ref op265 0) op265
(vector
(lambda (self266 tmp151)
(let ((b135 tmp151))
(let ((op267
(vector-ref
self266 2)))
((vector-ref op267 0)
op267
(vector
(lambda
(self268 tmp153)
(make-application
tmp153
(vector-ref
self268 2)
#t))
'60
(vector-ref
self266 3))
(vector-ref self266
4)))))
'61 (vector-ref self258 3)
k134 game)
game abs136)
...
Factored computation graph
C
1.0
C1
A:false
A:true
0.2
false
true
tr
n dependency graph
A
B
Systems of polynomial equations
C1 = 1.0 * C
A1 = 1.0 * A


Factored computation graph
A
1.0
A1
0.4
0.6
A2
A3
0.2
0.8
B
1.0
B1
0.6
0.4
B2
B3
0.3
0.7
C
1.0
C1
A:false
A:true
false
true
true
false
false
true
B:false
B:true
A:false
A:true
0)
53)
on
6
)
uations
$A1 = 1.0 * A$
Marginal probabilities
0.8

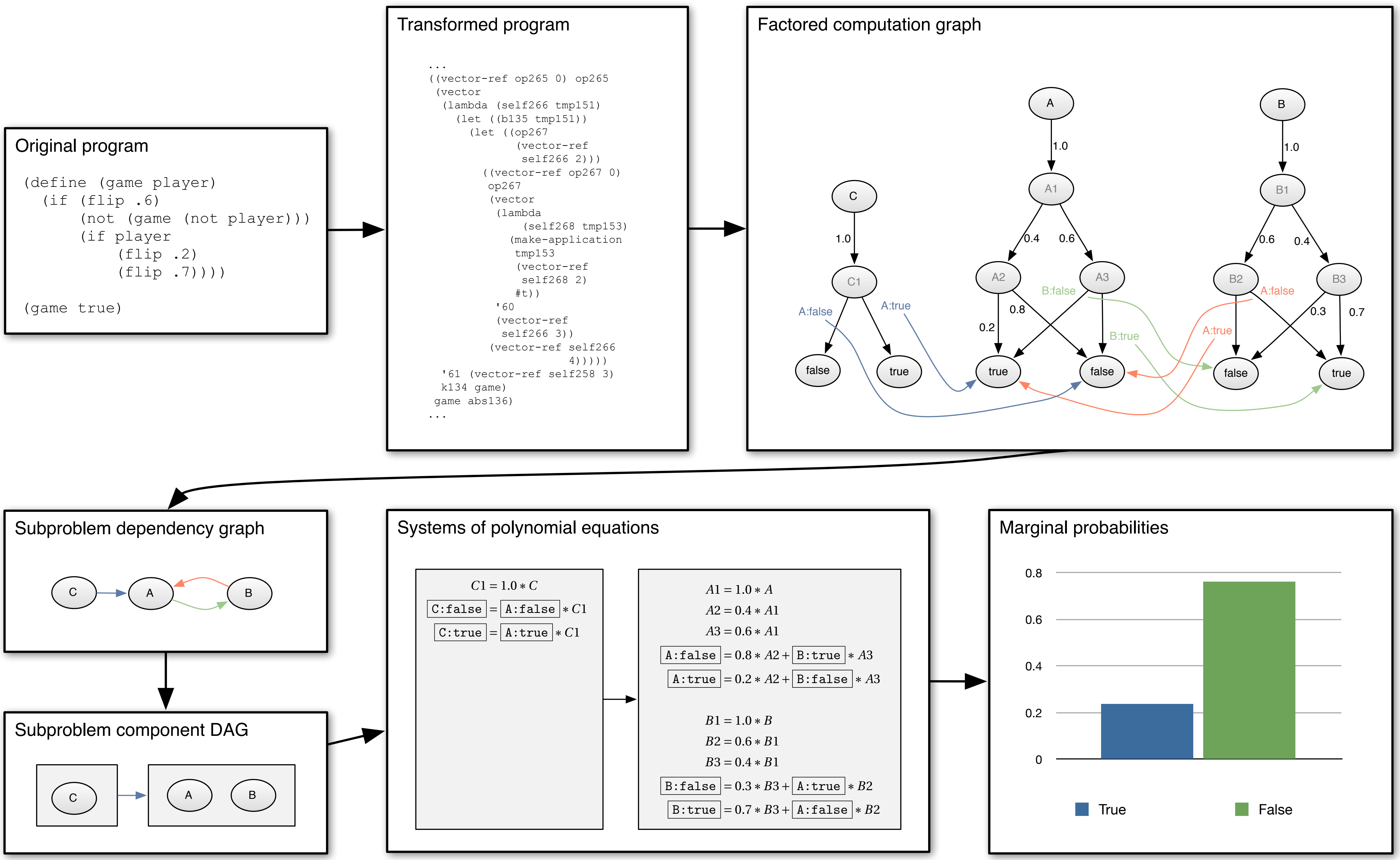

'61 (vector-ref self258 3)
k134 game)
game abs136)
...
Subproblem dependency graph
C
A
B
Systems of polynomial equatio
$C1 = 1.0 * C$
C:false = A:false * $C1$
C:true = A:true * $C1$
Subproblem component DAG
C
A
B



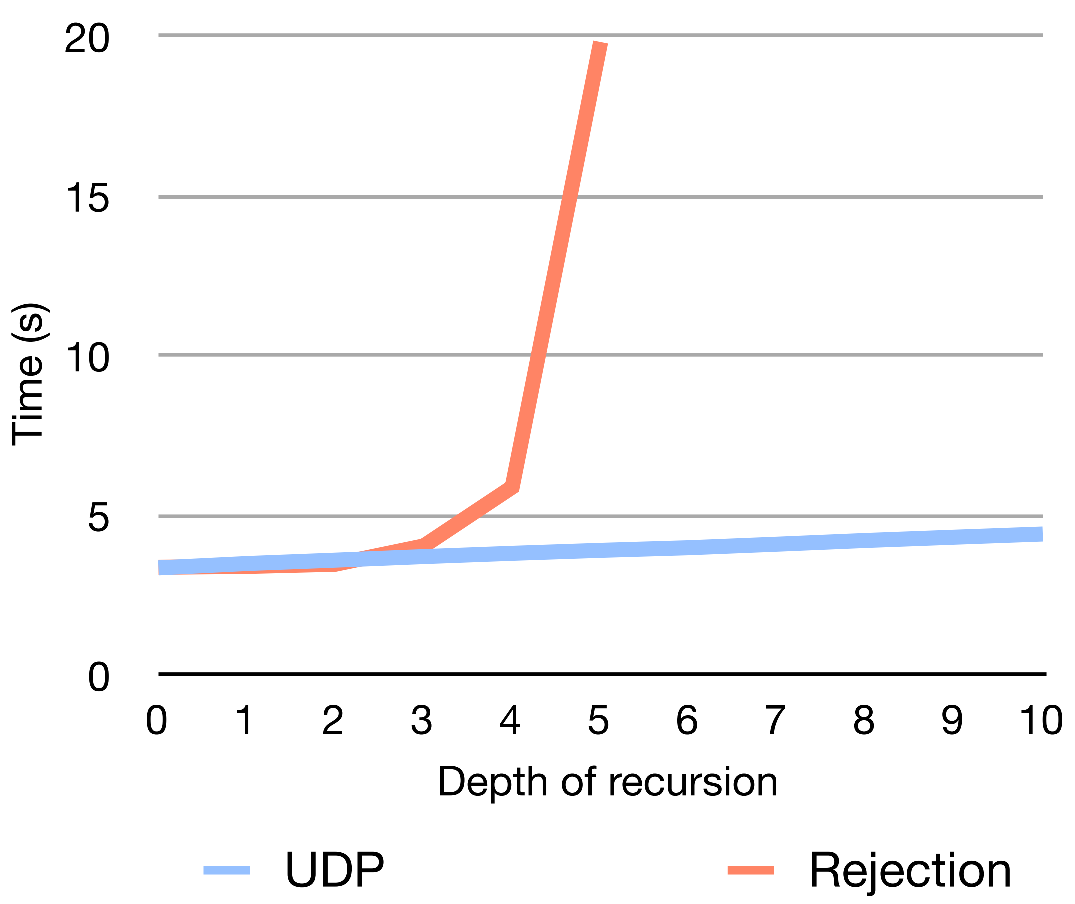
'61 (vector-ref self258 3)
k134 game)
game abs136)
...
false
true
true
graph
B
DAG
B
Systems of polynomial equations
C1 = 1.0 * C
C:false = A:false * C1
C:true = A:true * C1
A1 = 1.0 * A
A2 = 0.4 * A1
A3 = 0.6 * A1
A:false = 0.8 * A2 + B:true * A3
A:true = 0.2 * A2 + B:false * A3
B1 = 1.0 * B
B2 = 0.6 * B1
B3 = 0.4 * B1
B:false = 0.3 * B3 + A:true * B2
B:true = 0.7 * B3 + A:false * B2
Marginal prob
0.8
0.6
0.4
0.2
0



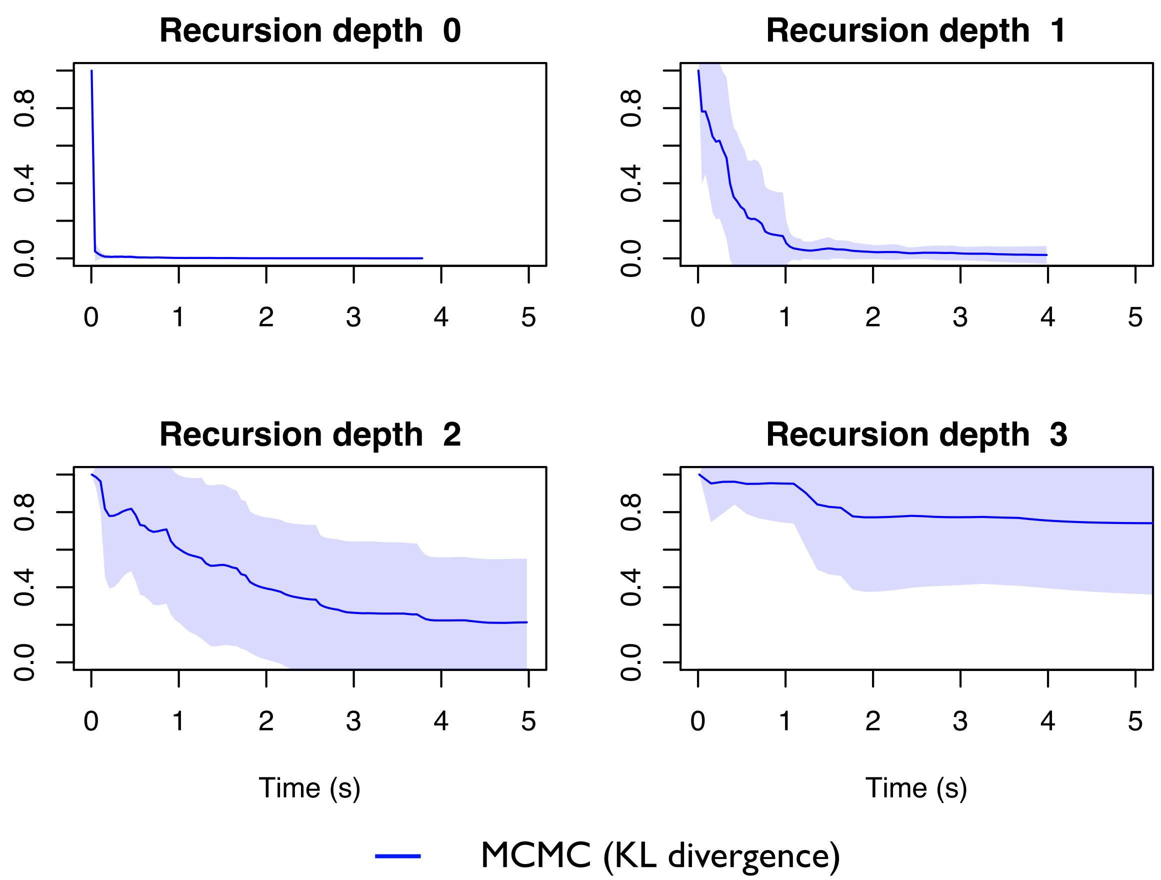
false
true
true
false
false
true
B:true
B:false
A:true
A:false
Marginal probabilities
0.8
0.6
0.4
0.2
0
True
False

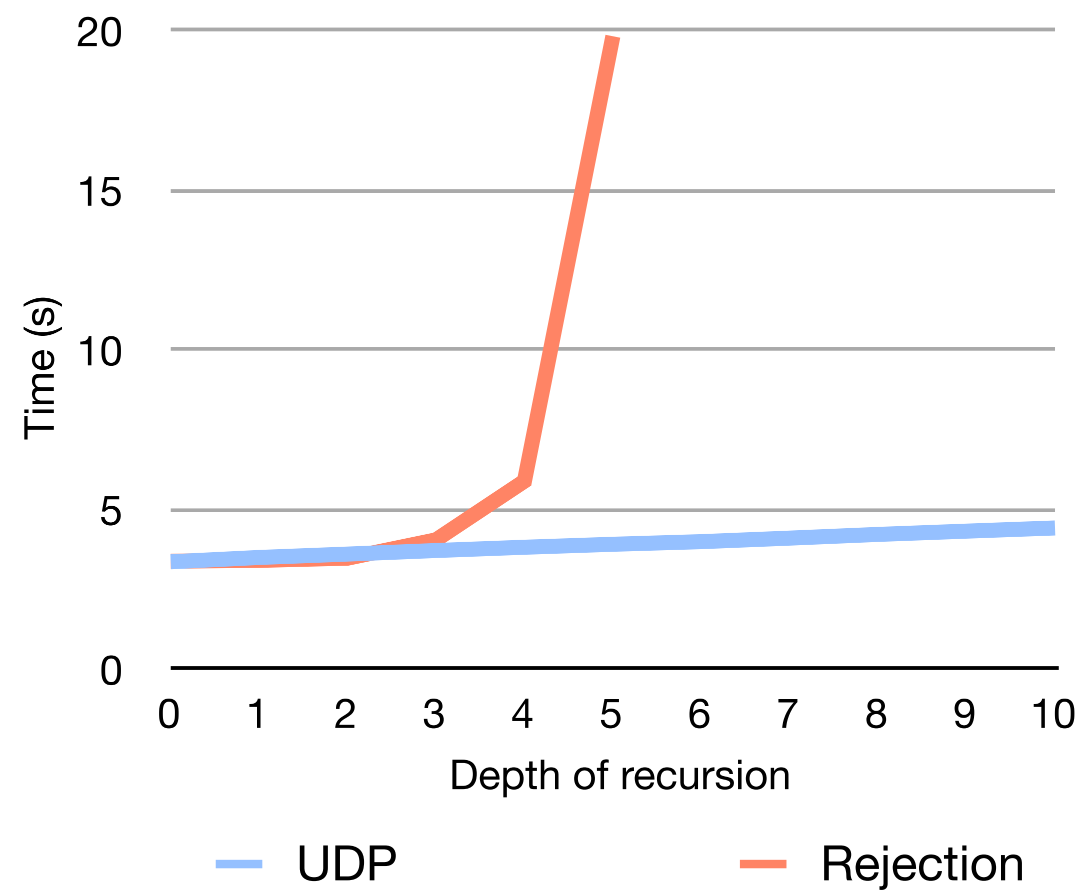

Original program
(define (game player)
(if (flip .6)
(not (game (not player)))
(if player
(flip .2)
(flip .7))))
(game true)
Transformed program
...
((vector-ref op265 0) op265
(vector
(lambda (self266 tmp151)
(let ((b135 tmp151))
(let ((op267
(vector-ref
self266 2)))
((vector-ref op267 0)
op267
(vector
(lambda
(self268 tmp153)
(make-application
tmp153
(vector-ref
self268 2)
#t))
'60
(vector-ref
self266 3))
(vector-ref self266
4)))))
'61 (vector-ref self258 3)
k134 game)
game abs136)
...
Factored computation graph
A
B
C
1.0
A1
B1
C1
0.4
0.6
0.4
A2
A3
B2
B3
A:false
A:true
B:false
B:true
A:false
A:true
0.2
0.8
0.3
0.7
false
true
Subproblem dependency graph
C
A
B
Subproblem component DAG
Systems of polynomial equations
C1 = 1.0 * C
C:false = A:false * C1
C:true = A:true * C1
A1 = 1.0 * A
A2 = 0.4 * A1
A3 = 0.6 * A1
A:false = 0.8 * A2 + B:true * A3
A:true = 0.2 * A2 + B:false * A3
B1 = 1.0 * B
B2 = 0.6 * B1
B3 = 0.4 * B1
B:false = 0.3 * B3 + A:true * B2
B:true = 0.7 * B3 + A:false * B2
Marginal probabilities
0.8
0.6
0.4
0.2
0
True
False

A simple coordination game



A simple coordination game





Cosh is a Church implementation based on the UDP algorithm:
github.com/stuhlmueller/cosh