

Side Channel Analysis Using a Model Counting Constraint Solver and Symbolic Execution

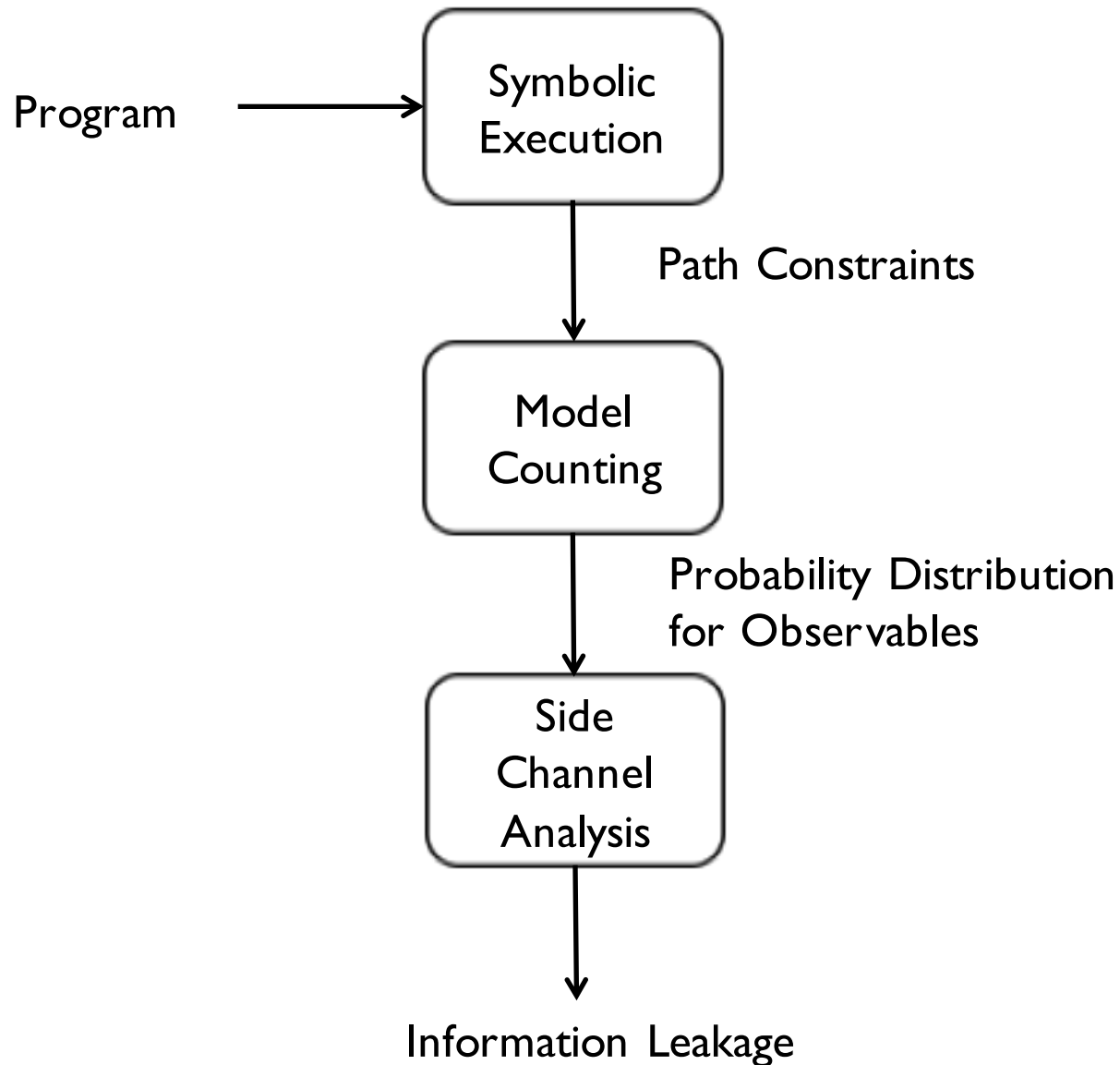
Tevfik Bultan

Computer Science Department
University of California, Santa Barbara

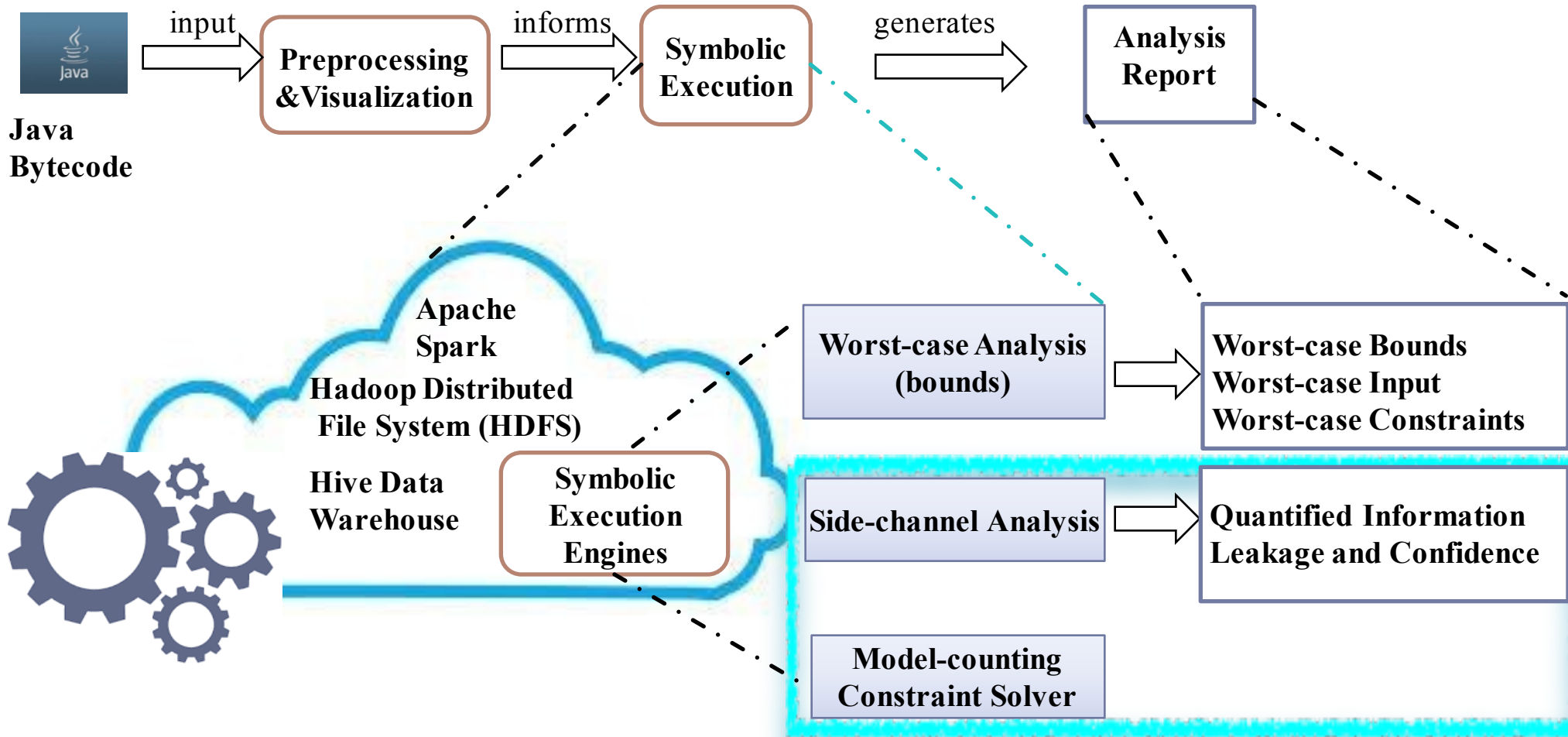
Joint work with:

Abdulbaki Aydin, Lucas Bang, UCSB
Corina Pasareanu, Quoc-Sang Phan, CMU, NASA

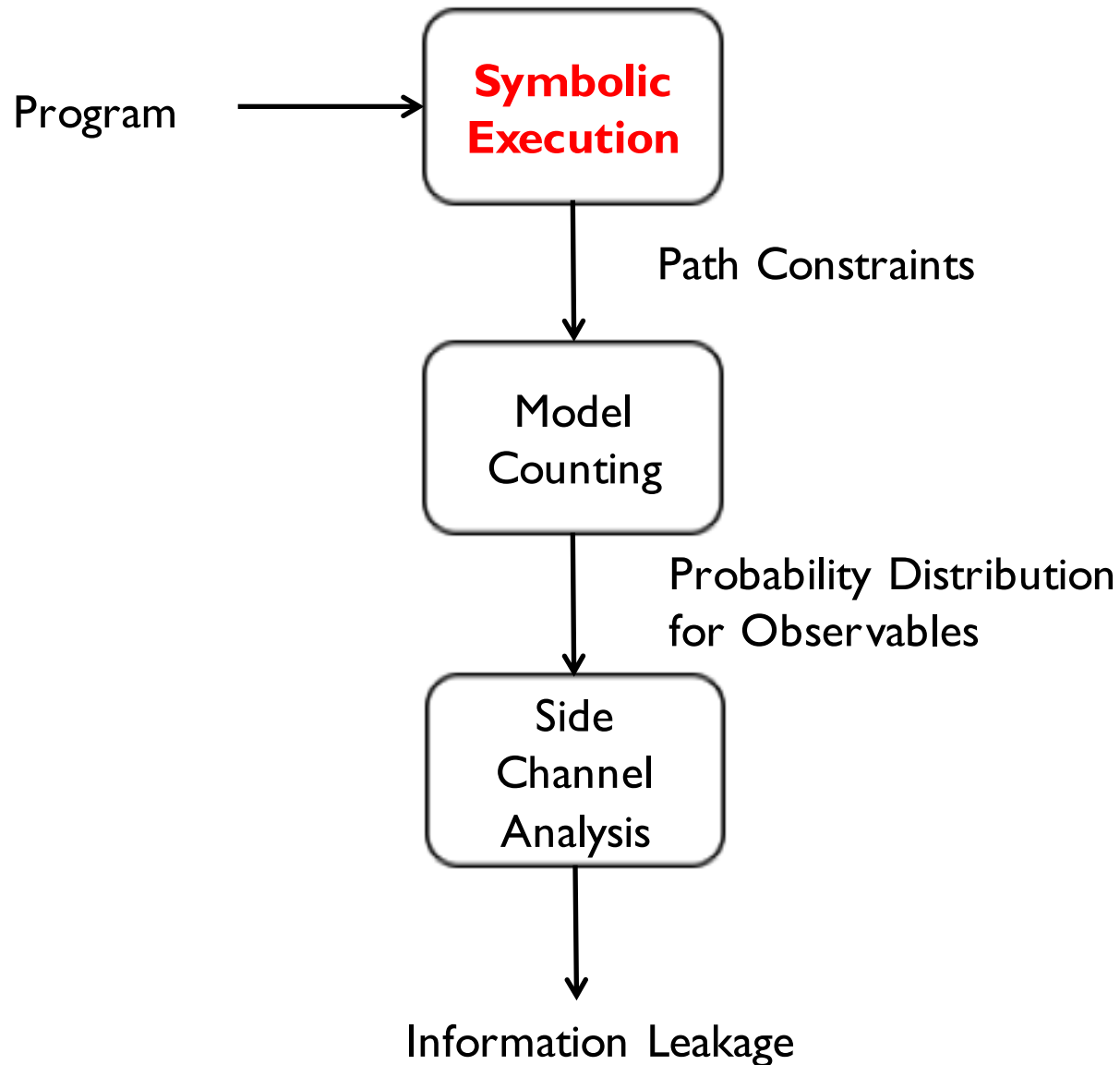
Overview



ISSTAC Project: Vanderbilt, UCSB, CMU



Overview



JPF and SPF

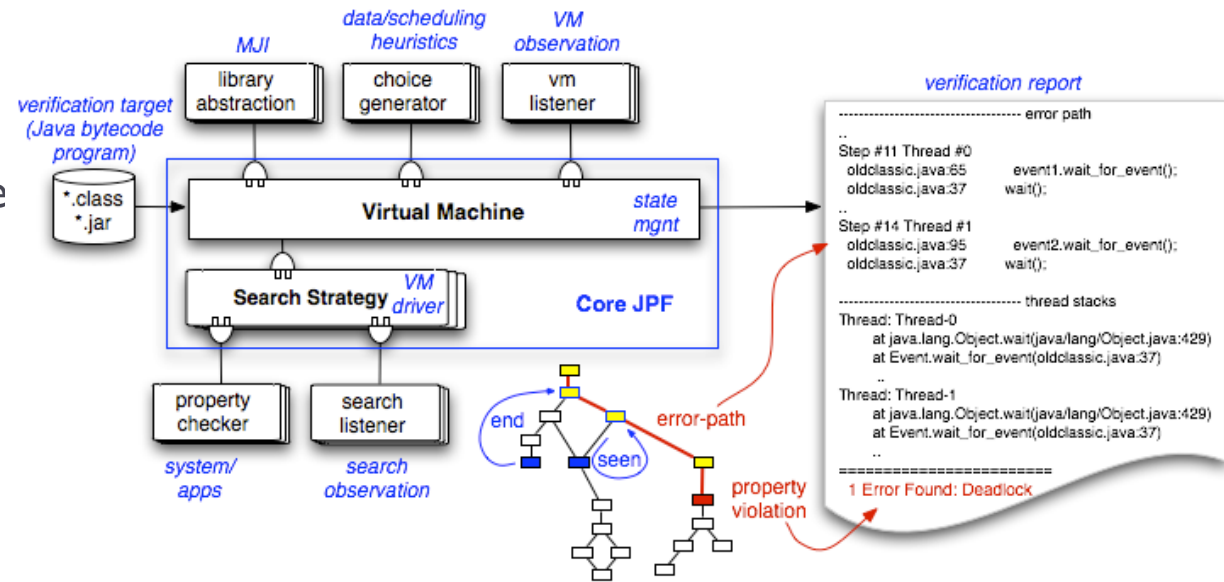
Java PathFinder

Extensible tool for Java bytecode verification

Uses specialized JVM

Developed at NASA Ames since 1999

Open-sourced



Symbolic PathFinder (SPF)

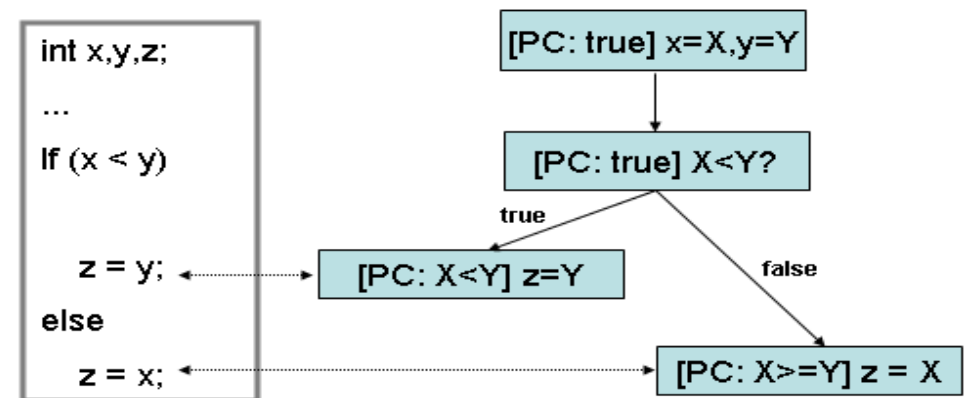
Symbolic execution tool for Java bytecode; open-sourced

Uses lazy initializations to handle complex data structures and arrays as inputs

Handles multi-threading

Provides support for symbolic string operations

Supports quantitative reasoning



A PIN Checker

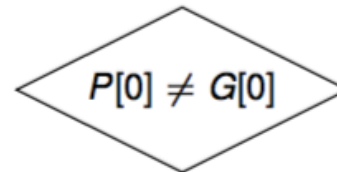
--

```
bool checkPIN(guess[])
for(i = 0; i < 4; i++)
    if(guess[i] != PIN[i])
        return false
return true
```

P: PIN, *G*: guess



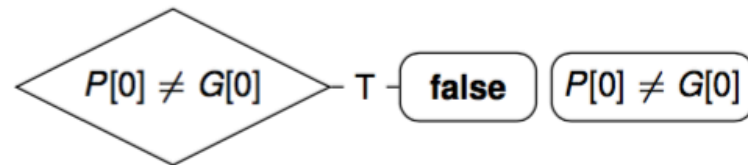
Symbolic Execution of PIN Checker



```
bool checkPIN(guess[])  
for(i = 0; i < 4; i++)  
    if(guess[i] != PIN[i])  
        return false  
return true
```

P: PIN, *G*: guess

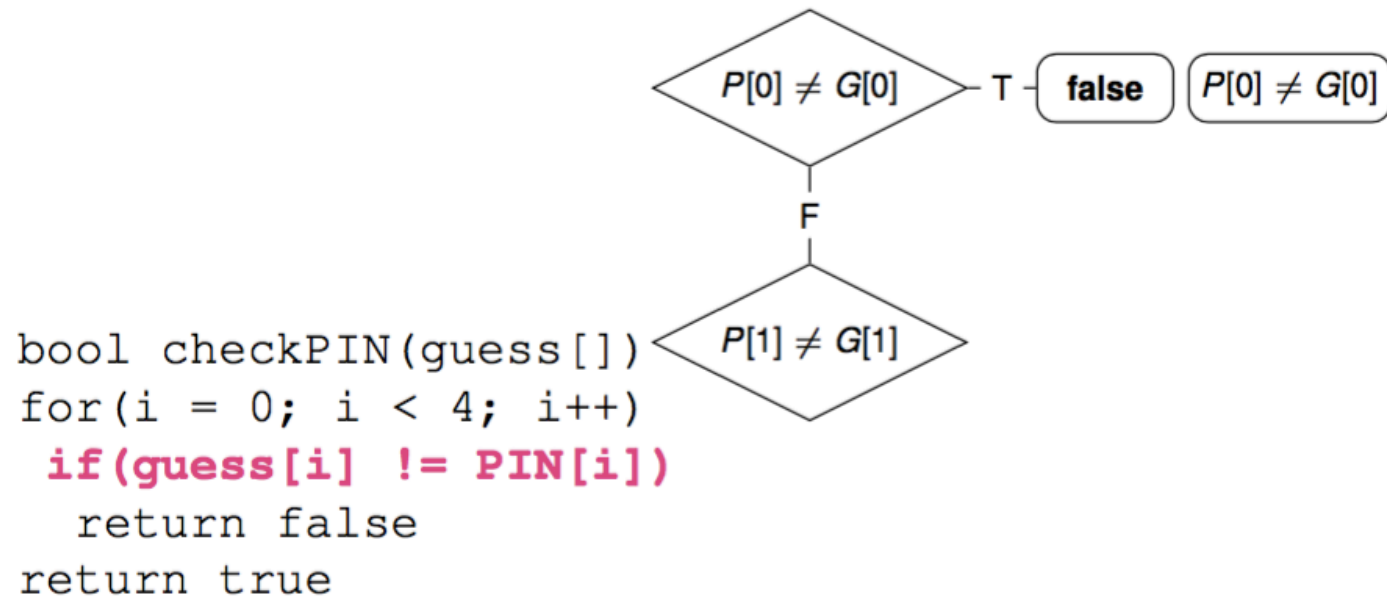
Symbolic Execution of PIN Checker



```
bool checkPIN(guess[])  
for(i = 0; i < 4; i++)  
    if(guess[i] != PIN[i])  
        return false  
return true
```

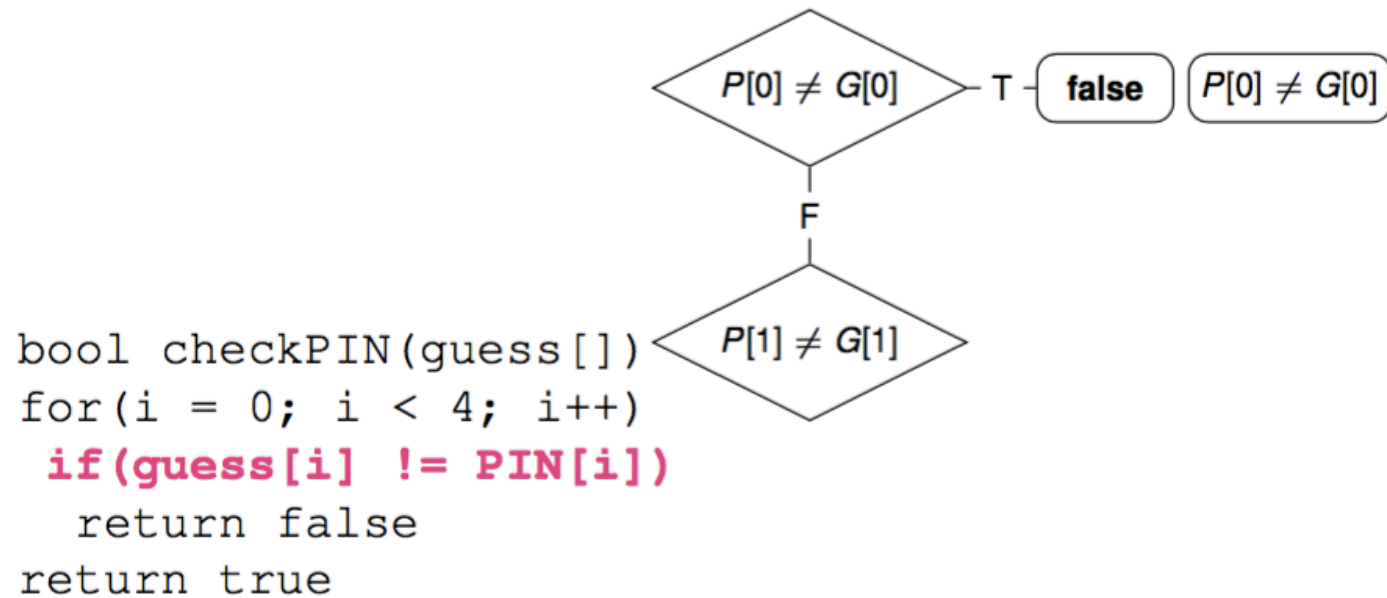
P: PIN, *G*: guess

Symbolic Execution of PIN Checker



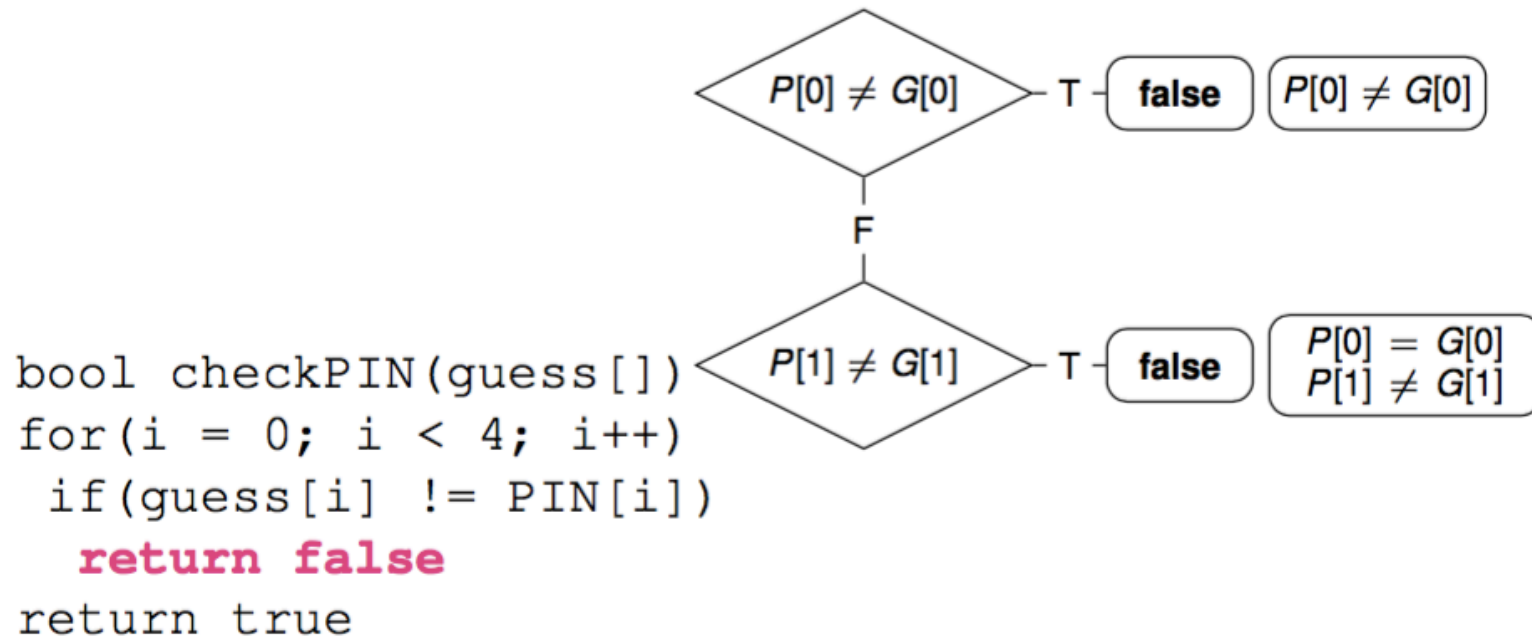
P : PIN, G : guess

Symbolic Execution of PIN Checker



P : PIN, G : guess

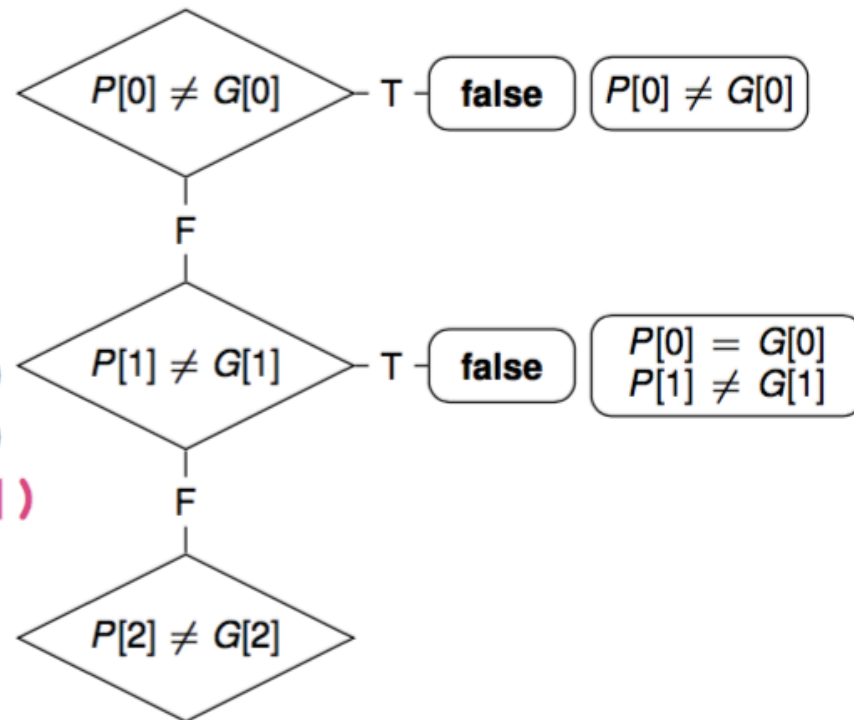
Symbolic Execution of PIN Checker



P : PIN, G : guess

Symbolic Execution of PIN Checker

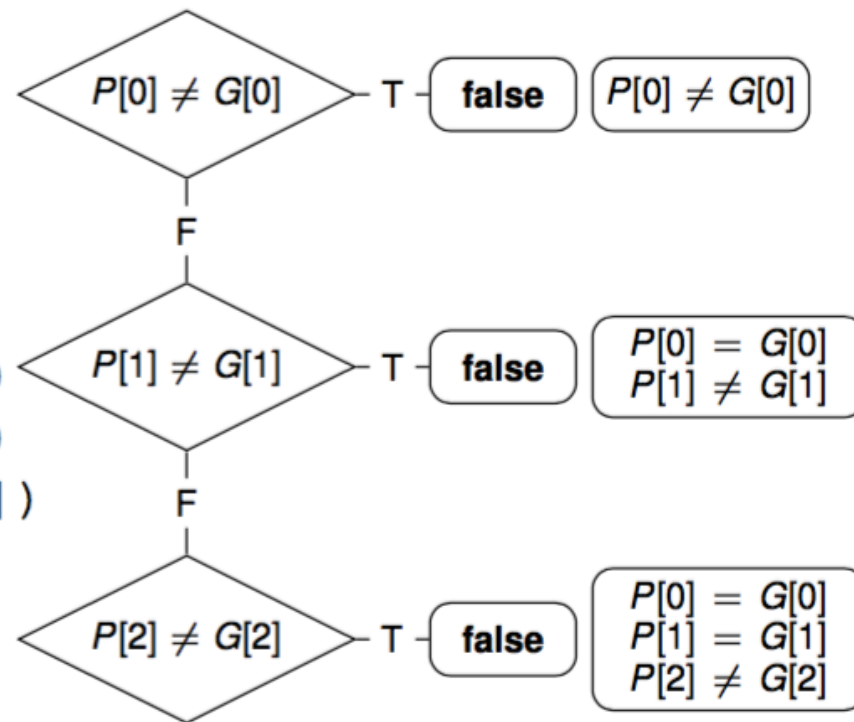
```
bool checkPIN(guess[])  
for(i = 0; i < 4; i++)  
    if(guess[i] != PIN[i])  
        return false  
return true
```



P : PIN, G : guess

Symbolic Execution of PIN Checker

```
bool checkPIN(guess[])  
for(i = 0; i < 4; i++)  
  if(guess[i] != PIN[i])  
    return false  
return true
```

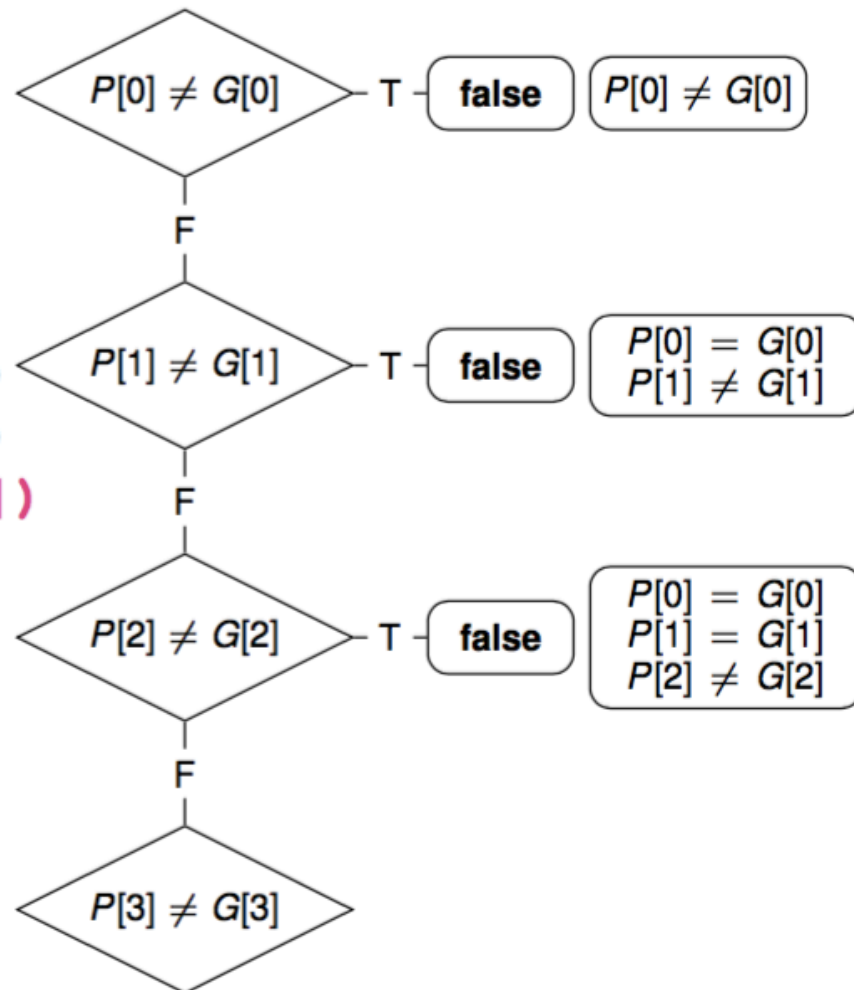


P: PIN, *G*: guess

Symbolic Execution of PIN Checker

```
bool checkPIN(guess[])  
for(i = 0; i < 4; i++)  
    if(guess[i] != PIN[i])  
        return false  
return true
```

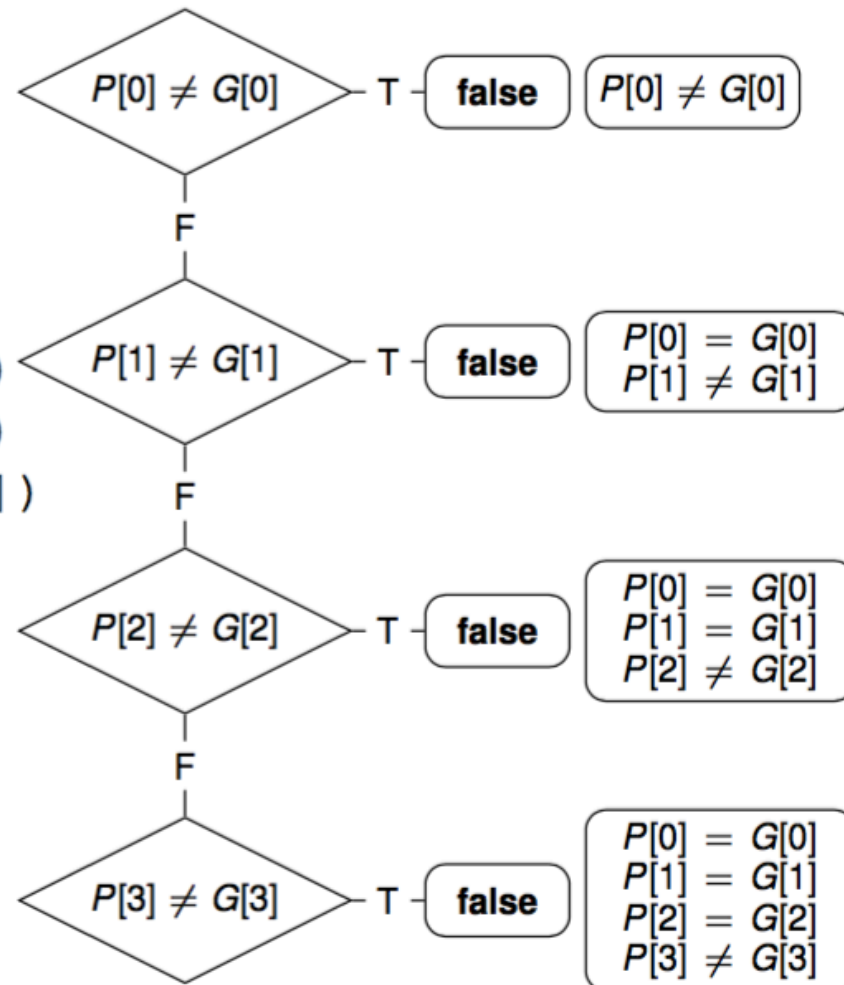
P : PIN, G : guess



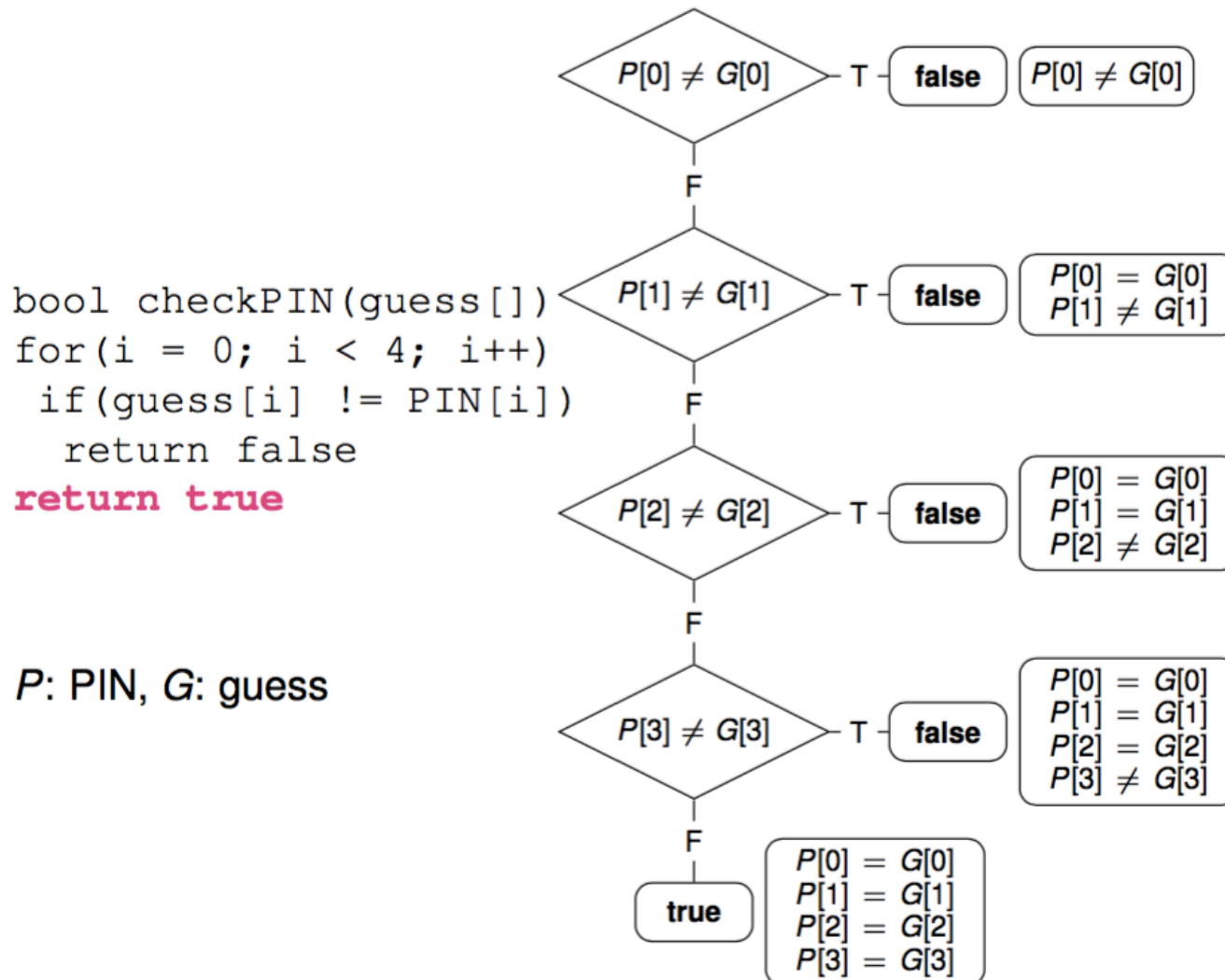
Symbolic Execution of PIN Checker

```
bool checkPIN(guess[])  
for(i = 0; i < 4; i++)  
  if(guess[i] != PIN[i])  
    return false  
return true
```

P: PIN, *G*: guess



Symbolic Execution of PIN Checker



Probabilistic Symbolic Execution

- ▶ Can we determine the probability of executing a particular program path?
- ▶ Let PC_i denote the path constraint of an execution path
- ▶ Let $|PC_i|$ denote the number of possible solutions for PC_i
- ▶ Let $|D|$ denote the size of the input domain
- ▶ Assume uniform distribution over the input domain

$$p(PC_i) = \frac{|PC_i|}{|D|}$$



Probabilistic Symbolic Execution of PIN Checker

- ▶ Assume binary 4 digit PIN. P and G each have 4 bits.
- ▶ $|D| = 2^8 = 256$.

i	0	1	2	3	4
PC_i	$P[0] \neq G[0]$	$P[0] = G[0]$ $P[1] \neq G[1]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] \neq G[2]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] \neq G[3]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] = G[3]$
$ PC_i $					
p_i					

- ▶ $p(PC_i) = \frac{|PC_i|}{|D|}$

Probabilistic Symbolic Execution of PIN Checker

- ▶ Assume binary 4 digit PIN. P and G each have 4 bits.
- ▶ $|D| = 2^8 = 256$.

i	0	1	2	3	4
PC_i	$P[0] \neq G[0]$	$P[0] = G[0]$ $P[1] \neq G[1]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] \neq G[2]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] \neq G[3]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] = G[3]$
$ PC_i $	128				
p_i	1/2				

- ▶ $p(PC_i) = \frac{|PC_i|}{|D|}$

Probabilistic Symbolic Execution of PIN Checker

- ▶ Assume binary 4 digit PIN. P and G each have 4 bits.
- ▶ $|D| = 2^8 = 256$.

i	0	1	2	3	4
PC_i	$P[0] \neq G[0]$	$P[0] = G[0]$ $P[1] \neq G[1]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] \neq G[2]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] \neq G[3]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] = G[3]$
$ PC_i $	128	64			
p_i	1/2	1/4			

- ▶ $p(PC_i) = \frac{|PC_i|}{|D|}$

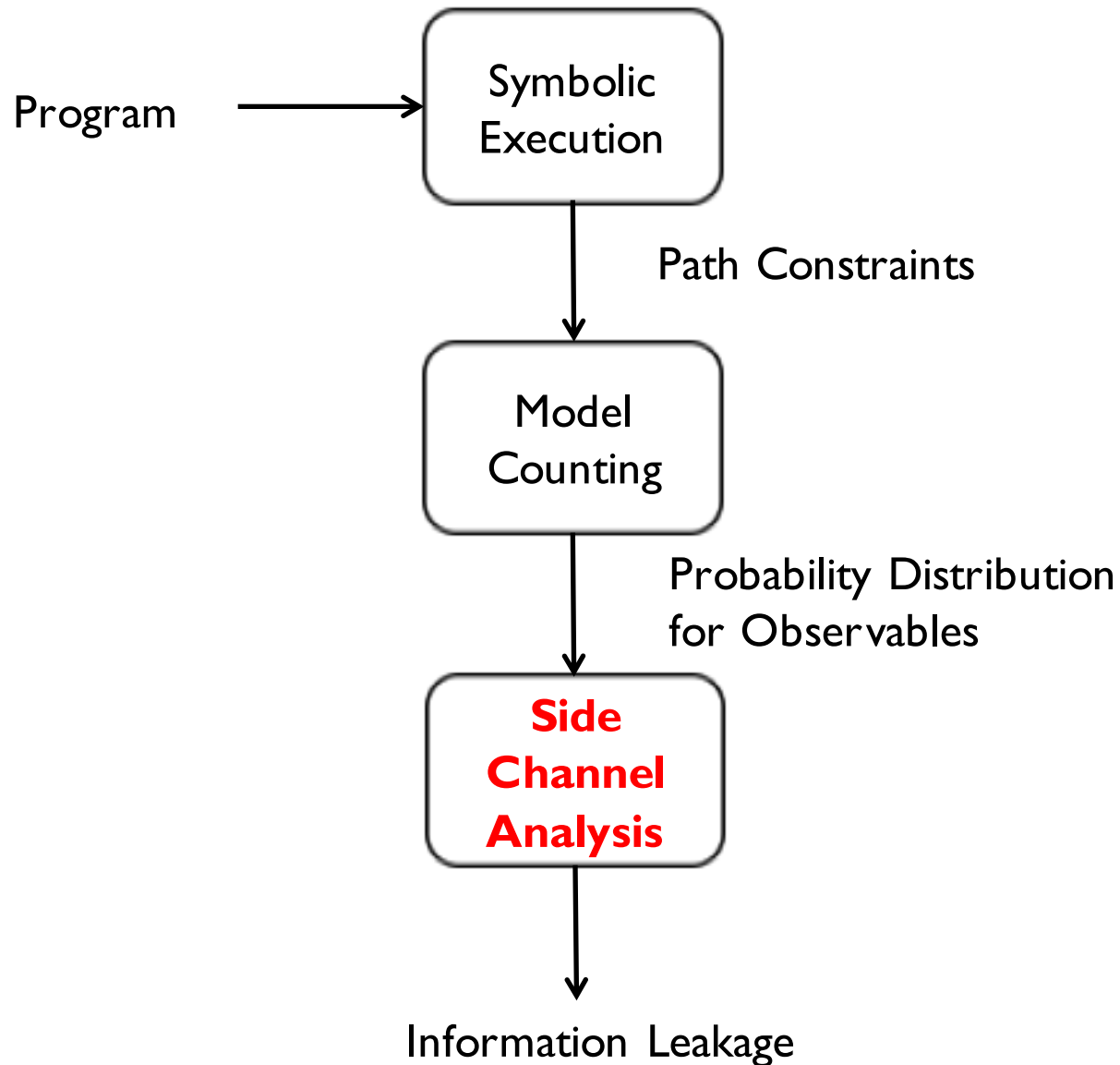
Probabilistic Symbolic Execution of PIN Checker

- ▶ Assume binary 4 digit PIN. P and G each have 4 bits.
- ▶ $|D| = 2^8 = 256$.

i	0	1	2	3	4
PC_i	$P[0] \neq G[0]$	$P[0] = G[0]$ $P[1] \neq G[1]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] \neq G[2]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] \neq G[3]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] = G[3]$
$ PC_i $	128	64	32	16	16
p_i	1/2	1/4	1/8	1/16	1/16

- ▶ Probability that an adversary can guess a prefix of length i in one guess is given by p_i

Overview



Information leakage

- ▶ Note that the PIN checker leaks information about the secret (secret is the pin value P).
- ▶ When an adversary tries a guess G there are two scenarios:
 - If G matches P then adversary learns the PIN
 - If G does not match P , then the adversary learns that the PIN values is not G
- ▶ This is due to the public output of the PIN checker
 - This is called the main channel
- ▶ However, there may be other observations one can make about the PIN checker

Information leakage

- ▶ An adversary may observe more than just the public output of a program.
- ▶ An adversary may observe:
 - execution time
 - memory usage
 - file size
 - network package size
- ▶ There may be information leakage about the secret from these observable values. These are called side channels.

Information Leakage

- ▶ How can we quantify leakage from a side channel (or main channel)?

- ▶ Shannon Entropy

$$H = \sum p_i \log \frac{1}{p_i} = E \left[\log \frac{1}{p_i} \right]$$

- ▶ Intuition:
- ▶ The **expected** amount of **information gain** (i.e., the expected amount of surprise) expressed in terms of **bits**

Information Leakage

- ▶ Example:
- ▶ Seattle weather, always raining:
 - ▶ $p_{\text{rain}} = 1, p_{\text{sun}} = 0$
 - ▶ Entropy: $H = 0$
- ▶ Costa Rica weather, coin flip:
 - ▶ $p_{\text{rain}} = 1/2, p_{\text{sun}} = 1/2$
 - ▶ Entropy: $H = 1$
- ▶ Santa Barbara weather, almost always beautiful:
 - ▶ $p_{\text{rain}} = 1/10, p_{\text{sun}} = 9/10$
 - ▶ Entropy: $H = 0.496$

Information Leakage via Side Channels

- ▶ Side channels produce a set of observables that partition the secret:

$$\mathcal{O} = \{o_1, o_2, \dots, o_m\}$$

- ▶ By computing the probability of observable values we can compute the entropy:

$$\mathcal{H}(P) = - \sum_{i=1, m} p(o_i) \log_2(p(o_i))$$

- ▶ We can compute the probability of observable values using model counting:

the probability of observing o_i is:

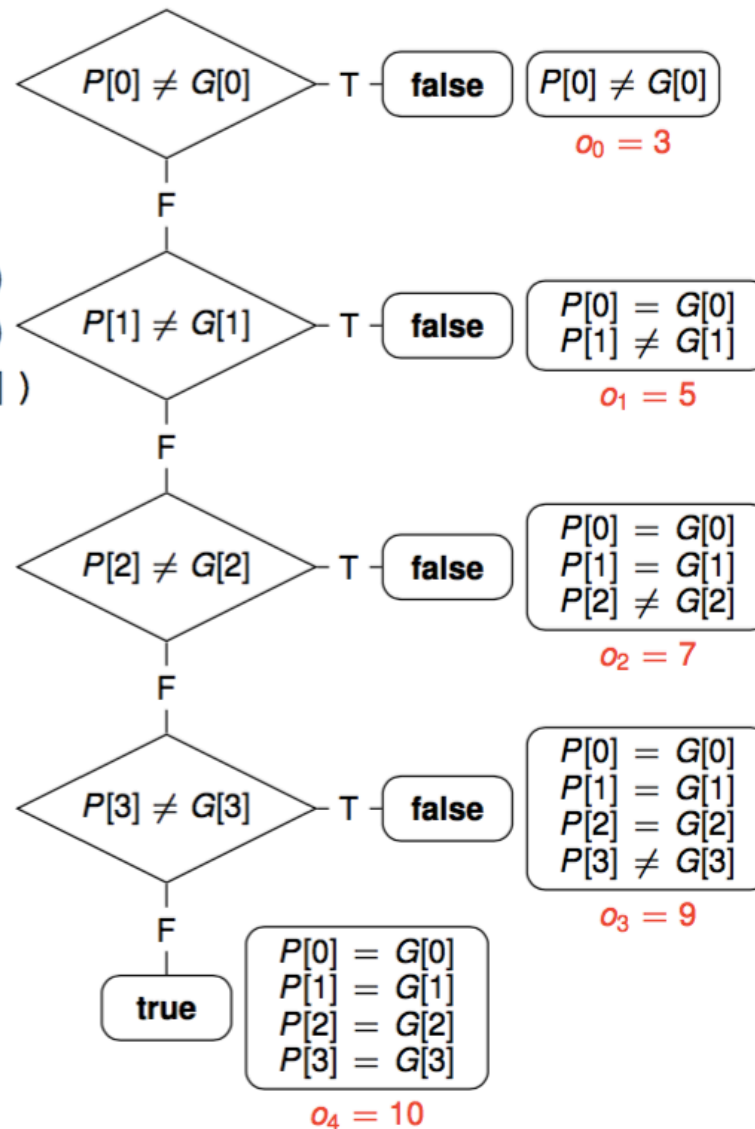
$$p(o_i) = \frac{\sum_{\text{cost}(\pi_j)=o_i} \#(PC_j(h, l))}{\#D}$$

Symbolic Execution of PIN Checker

```
bool checkPIN(guess[])
for(i = 0; i < 4; i++)
  if(guess[i] != PIN[i])
    return false
return true
```

P : PIN, G : guess

o_i = lines of code



Probabilistic Symbolic Execution of PIN Checker

- ▶ Assume binary 4 digit PIN. P and G each have 4 bits.
- ▶ $|D| = 2^8 = 256$.

i	0	1	2	3	4
PC_i	$P[0] \neq G[0]$	$P[0] = G[0]$ $P[1] \neq G[1]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] \neq G[2]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] \neq G[3]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] = G[3]$
return	false	false	false	false	true
$ PC_i $	128	64	32	16	16
p_i	1/2	1/4	1/8	1/16	1/16
o_i	3	5	7	9	10

Information Leakage

i	0	1	2	3	4
PC_i	$P[0] \neq G[0]$	$P[0] = G[0]$ $P[1] \neq G[1]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] \neq G[2]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] \neq G[3]$	$P[0] = G[0]$ $P[1] = G[1]$ $P[2] = G[2]$ $P[3] = G[3]$
return	false	false	false	false	true
$ PC_i $	128	64	32	16	16
p_i	1/2	1/4	1/8	1/16	1/16
o_i	3	5	7	9	10

$$H = \sum p_i \log \frac{1}{p_i} = 1.8750$$

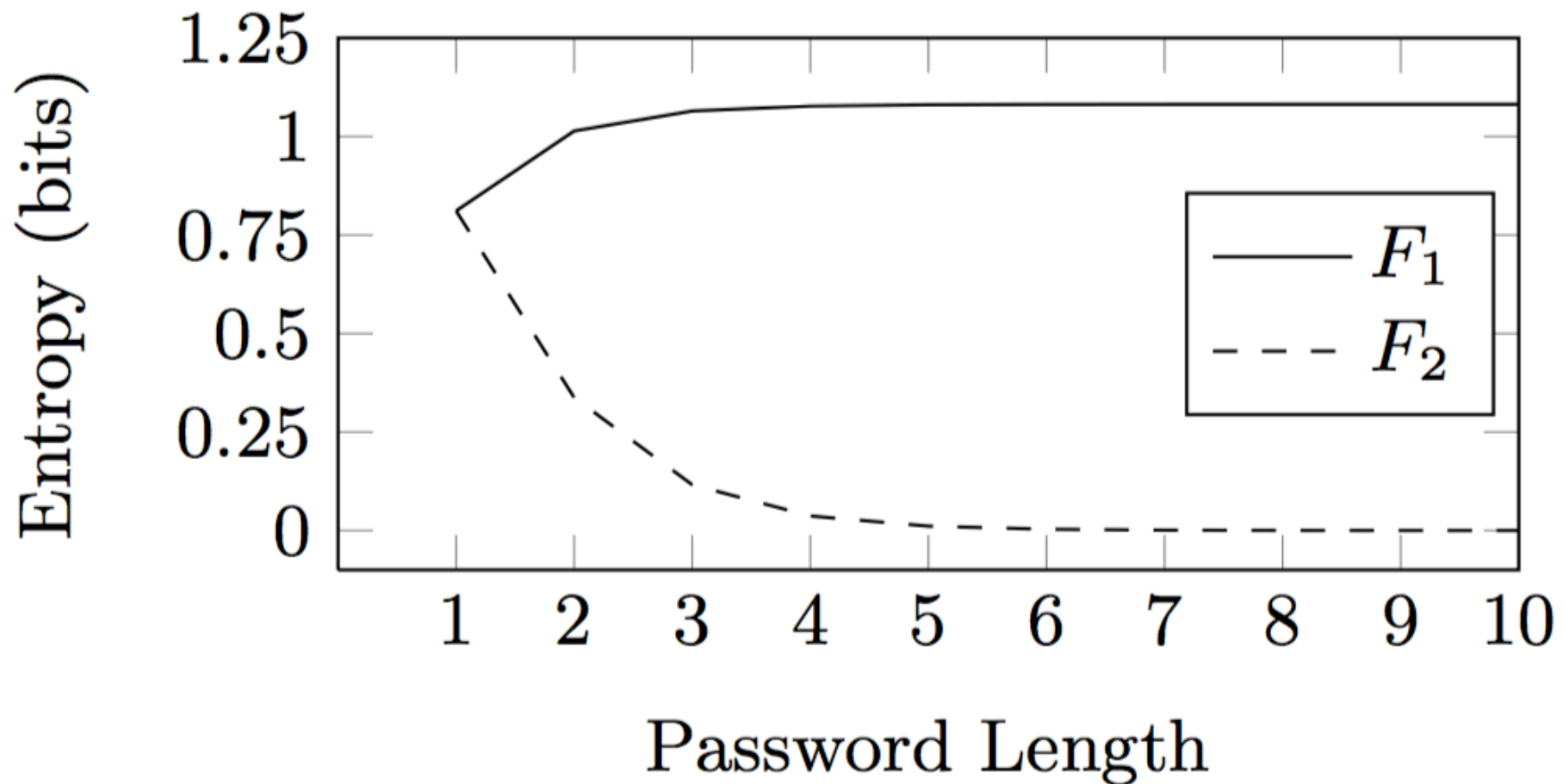
- ▶ H: The expected amount of information gain by the adversary

A secure PIN checker

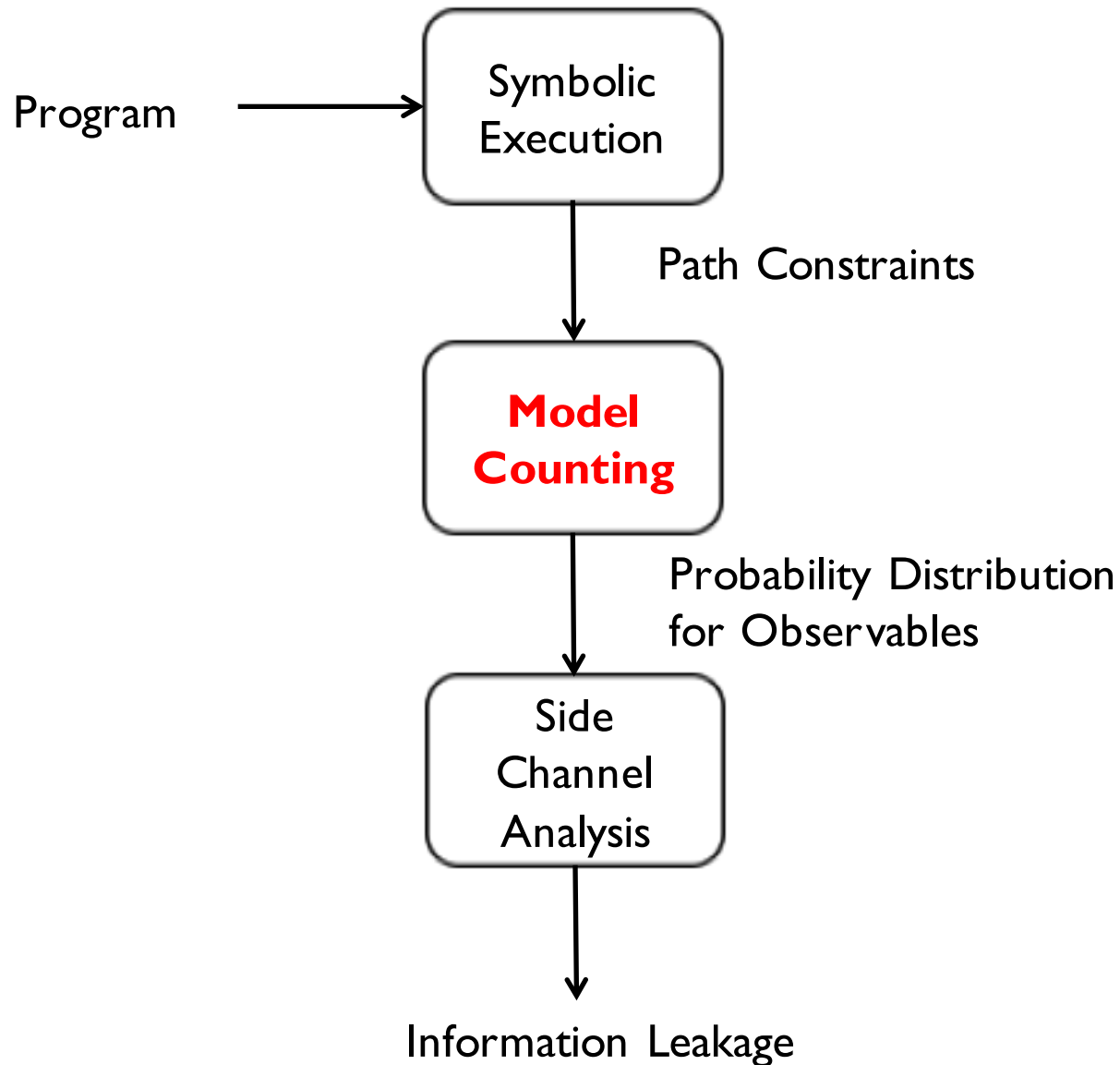
```
public verifyPassword (guess[])
  matched = true
  for (int i = 0; i < 4; i++)
    if (guess[i] != PIN[i])
      matched = false
    else
      matched = matched
  return matched
```

- ▶ Only two observables: o_0 : does not match, o_1 : full match
- ▶ $p(o_0) = 15/16$, $p(o_1) = 1/16$
- ▶ $H_{\text{secure}} = 0.33729 < H_{\text{sidechannel}} = 1.8750$

Secure vs. insecure PIN checker



Overview



Model Counting String Constraint Solver

INPUT

string
constraint:

C

**Automata-Based
model Counting
string constraint
solver
(ABC)**

OUTPUT

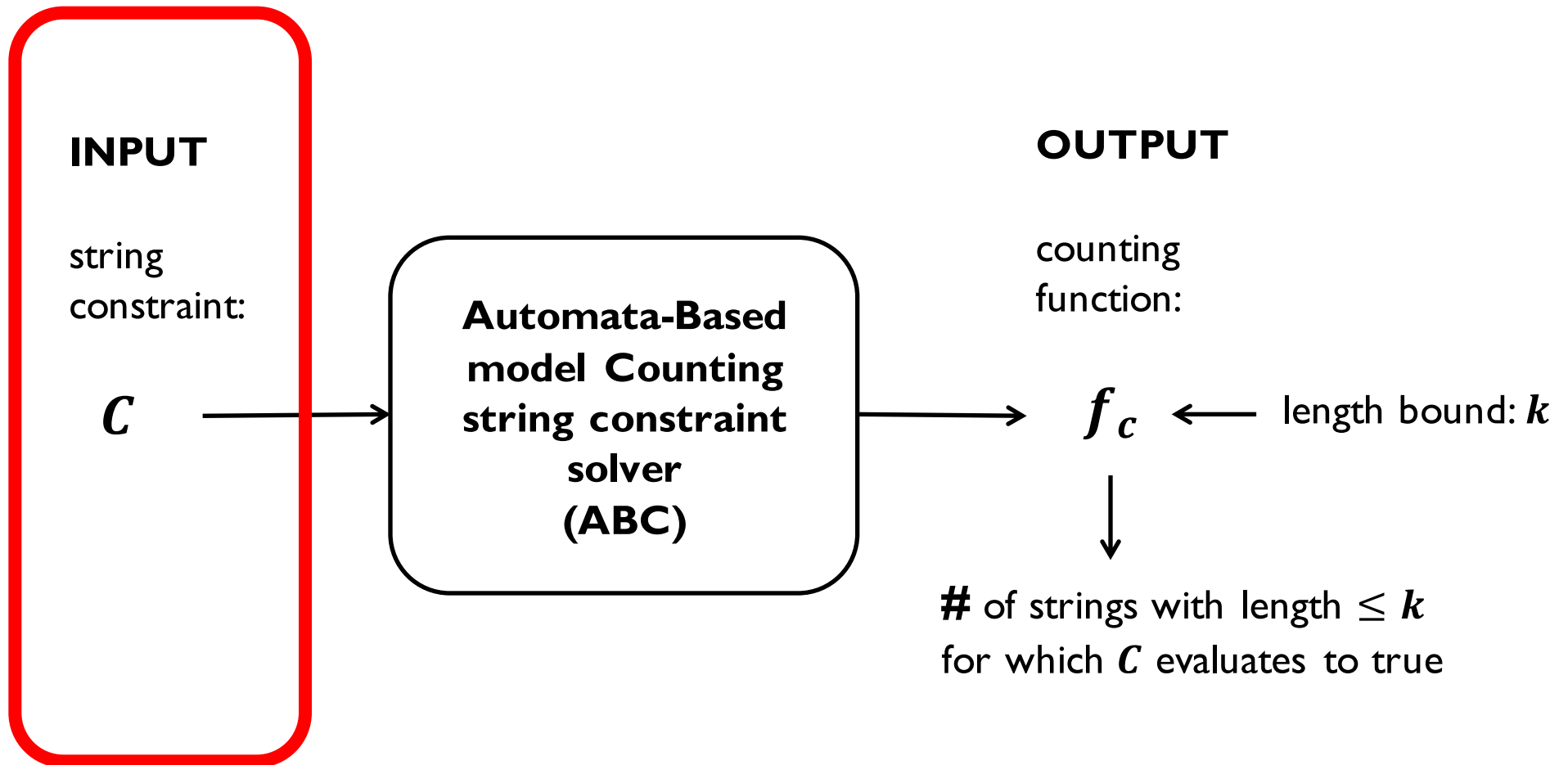
counting
function:

f_c

← length bound: k

↓
of strings with length $\leq k$
for which C evaluates to true

Model Counting String Constraint Solver



String Constraint Language

$C \rightarrow bterm$

$bterm \rightarrow v \mid true \mid false$
| $\neg bterm \mid bterm \wedge bterm \mid bterm \vee bterm \mid (bterm)$
| $sterm = sterm$
| $match(sterm, sterm)$
| $contains(sterm, sterm)$
| $begins(sterm, sterm)$
| $ends(sterm, sterm)$
| $iterm = iterm \mid iterm < iterm \mid iterm > iterm$

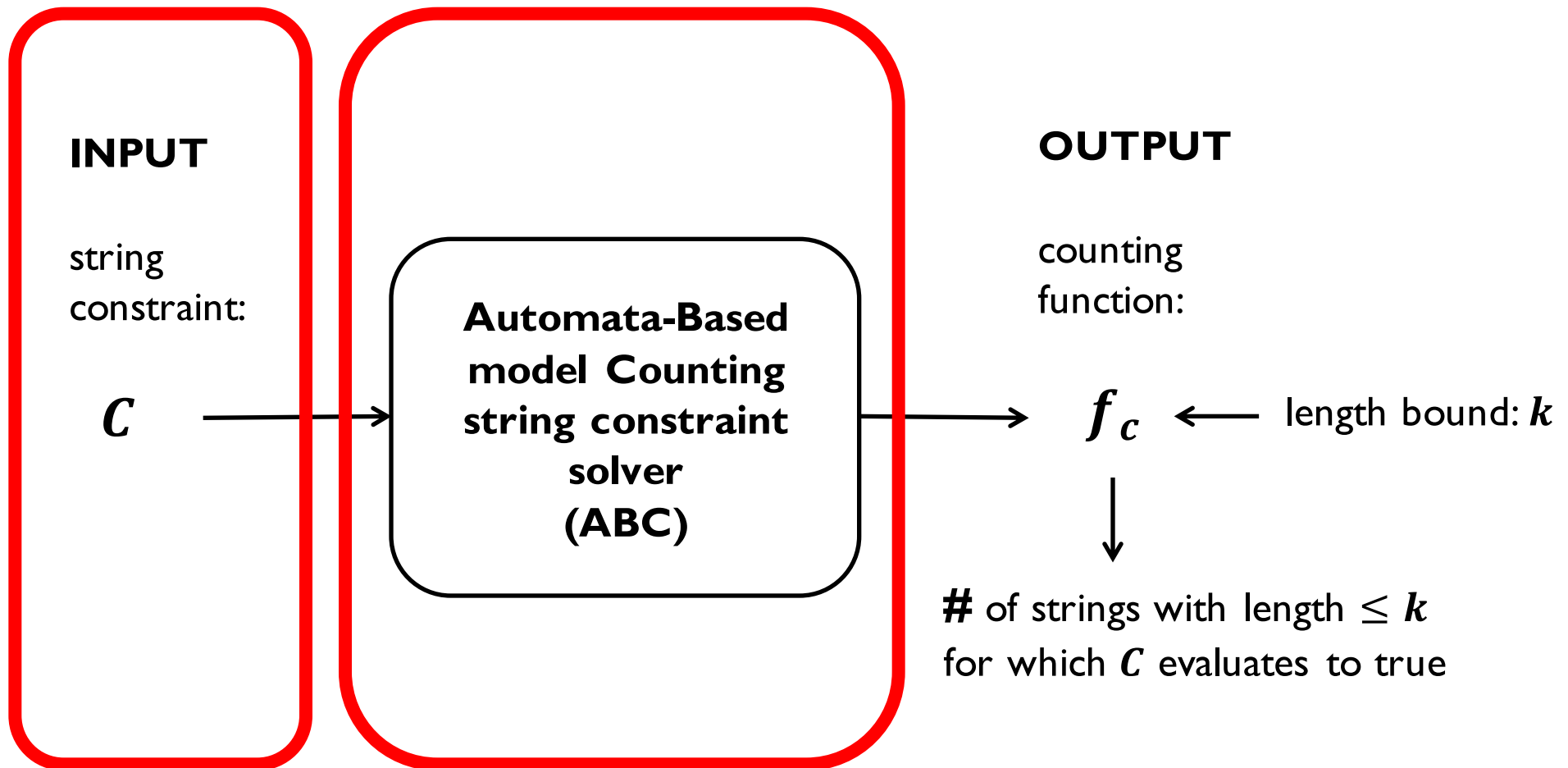
$iterm \rightarrow v \mid n$
| $iterm + iterm \mid iterm - iterm \mid iterm \times n \mid (iterm)$
| $length(sterm) \mid toint(sterm)$
| $indexof(sterm, sterm)$
| $lastindexof(sterm, sterm)$

$sterm \rightarrow v \mid \epsilon \mid s$
| $sterm.sterm \mid sterm|sterm \mid sterm^* \mid (sterm)$
| $charat(sterm, iterm) \mid tostring(iterm)$
| $toupper(sterm) \mid tolower(sterm)$
| $substring(sterm, iterm, iterm)$
| $replacefirst(sterm, sterm, sterm)$
| $replacelast(sterm, sterm, sterm)$
| $replaceall(sterm, sterm, sterm)$

Example String Expressions

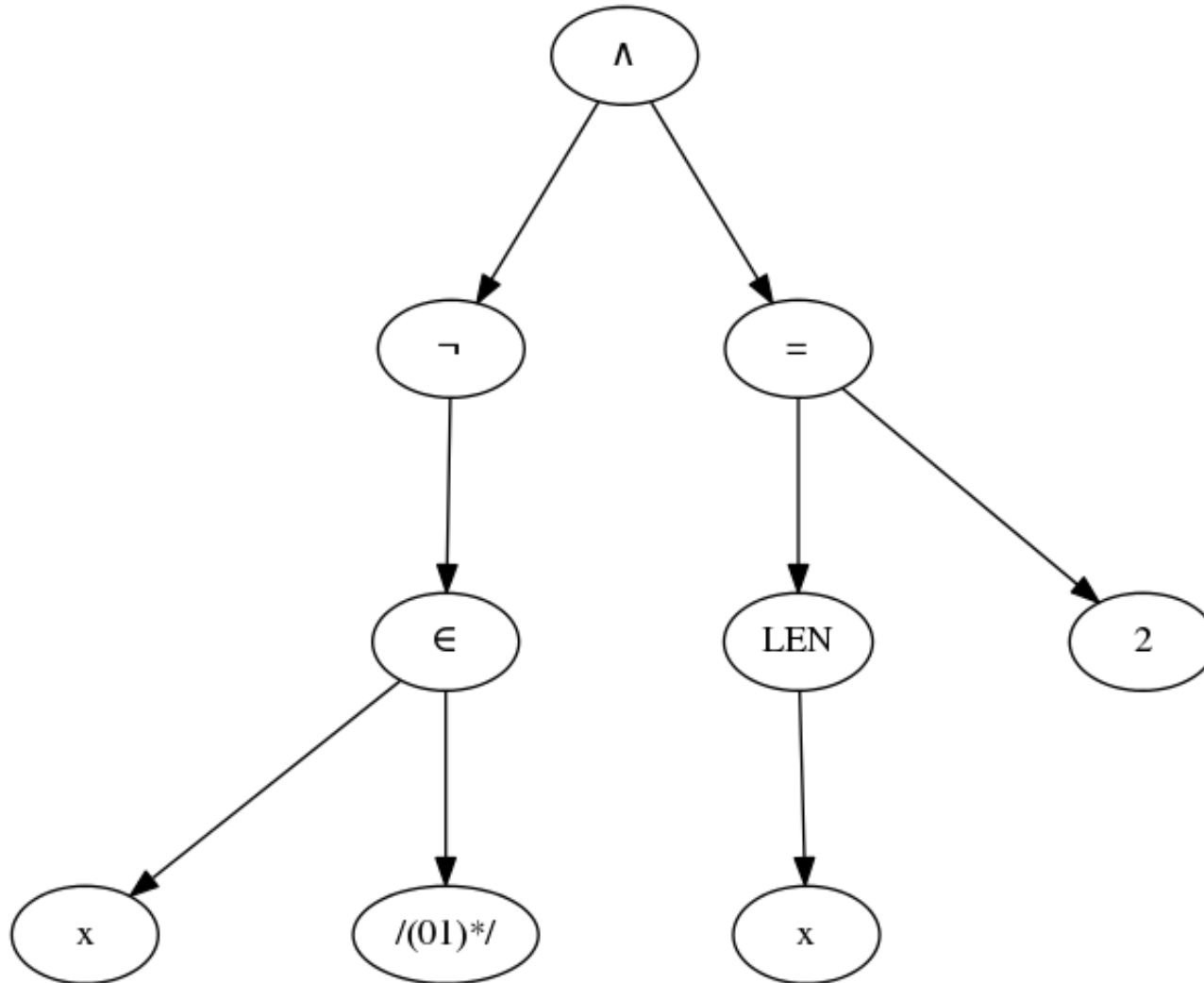
	String Expression	Constraint Language
Java	<code>s.length()</code>	<code>length(s)</code>
	<code>s.isEmpty()</code>	<code>length(s) == 0</code>
	<code>s.startsWith(t,n)</code>	$0 \leq n \wedge n \leq s \wedge$ <code>begins(substring(s,n, s),t)</code>
	<code>s.indexOf(t,n)</code>	<code>indexof(substring(s,n, s),t)</code>
	<code>s.replaceAll(p,r)</code>	<code>replaceall(s,p,r)</code>
PHP	<code>strrpos(s, t)</code>	<code>lastindexof(s,t)</code>
	<code>substr_replace(s, t,i,j)</code>	<code>substring(s,0,i).t.substring(s,j, s)</code>
	<code>strip_tags(s)</code>	<code>replaceall(s,("<a>" "<p>" ...), "")</code>
	<code>mysql_real_escape_string(s)</code>	<code>...replaceall(s, replaceall(s, "\\ ", "\\ \\ ") , "' ", "\\' ") ...</code>

Model Counting String Constraint Solver



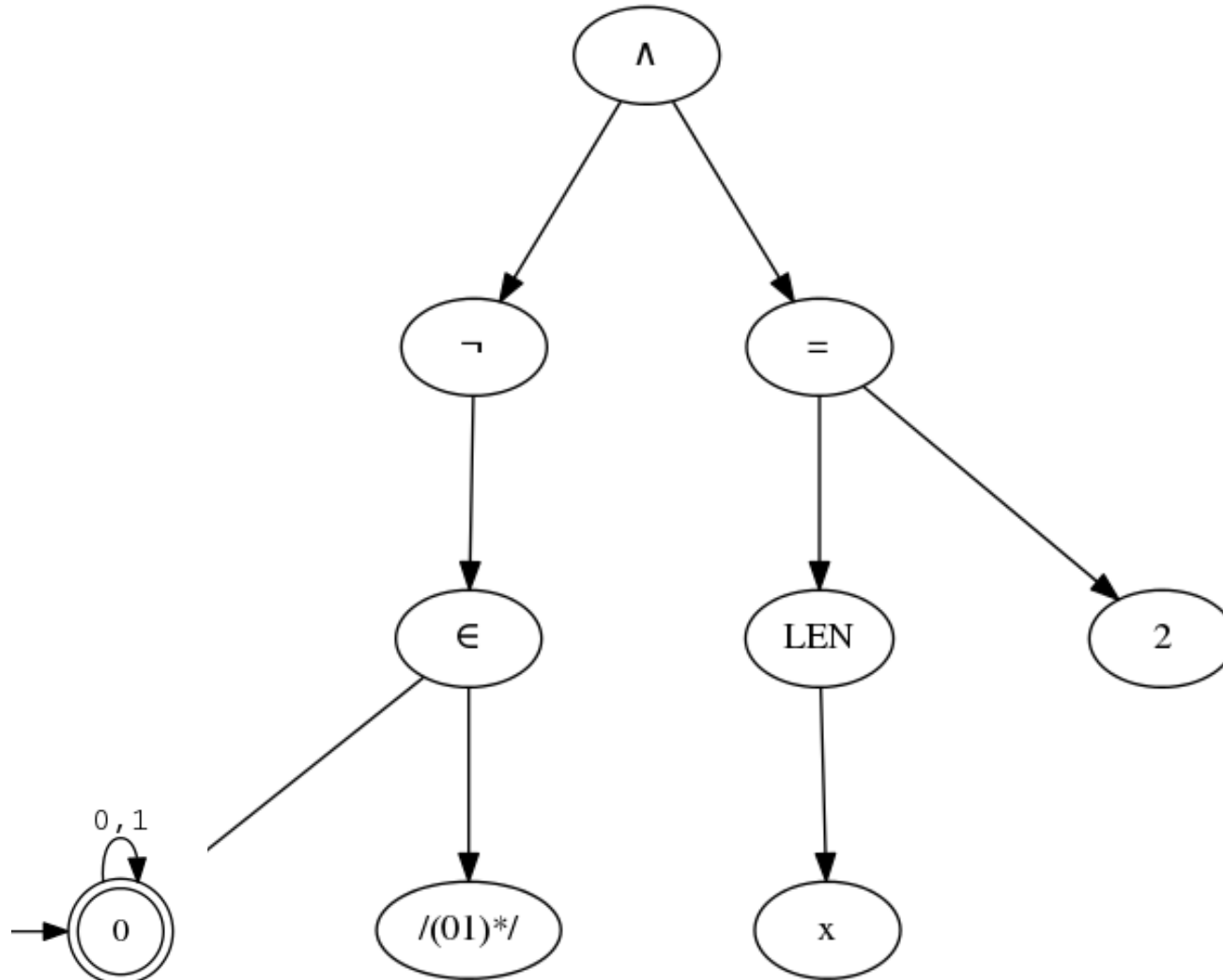
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



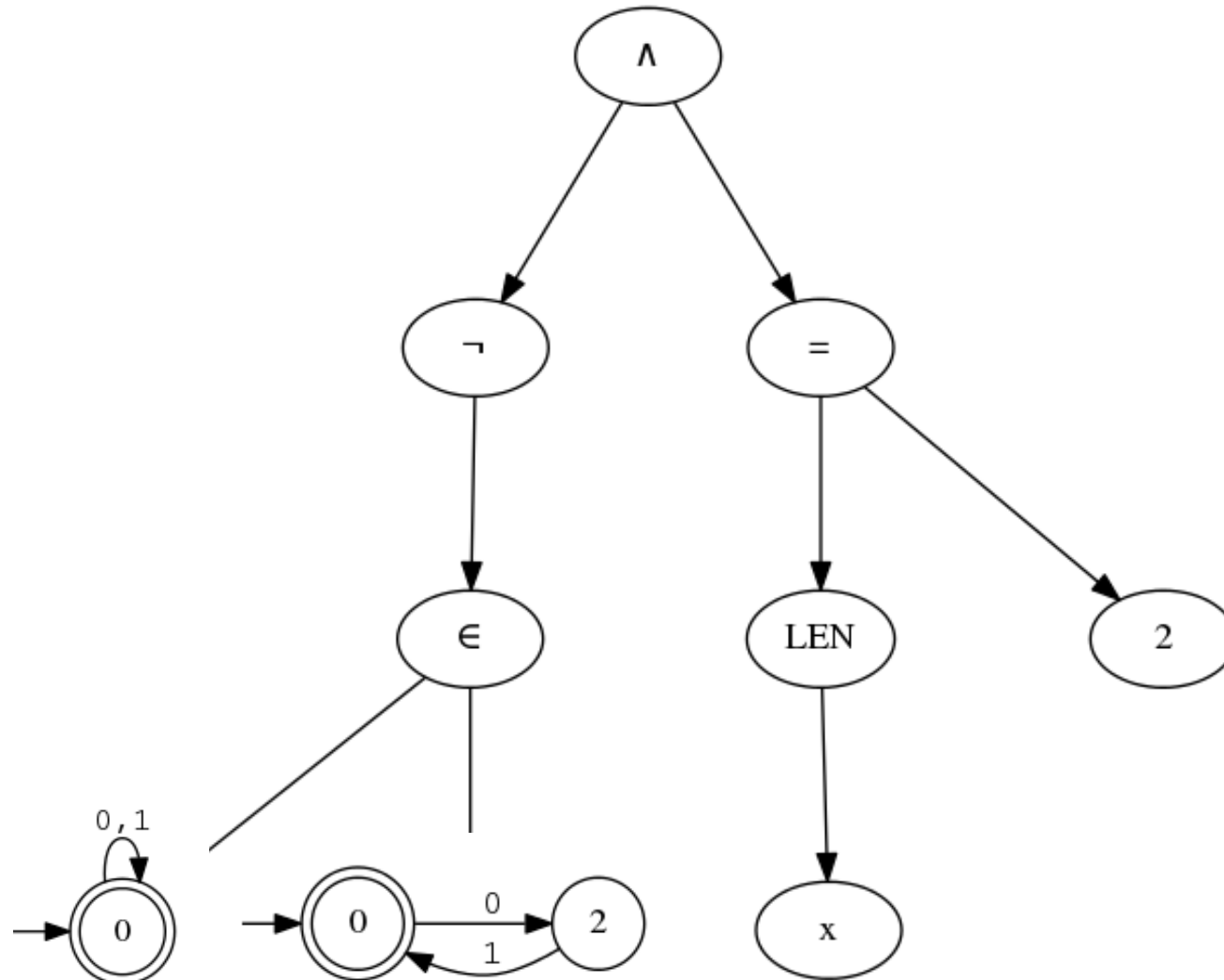
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



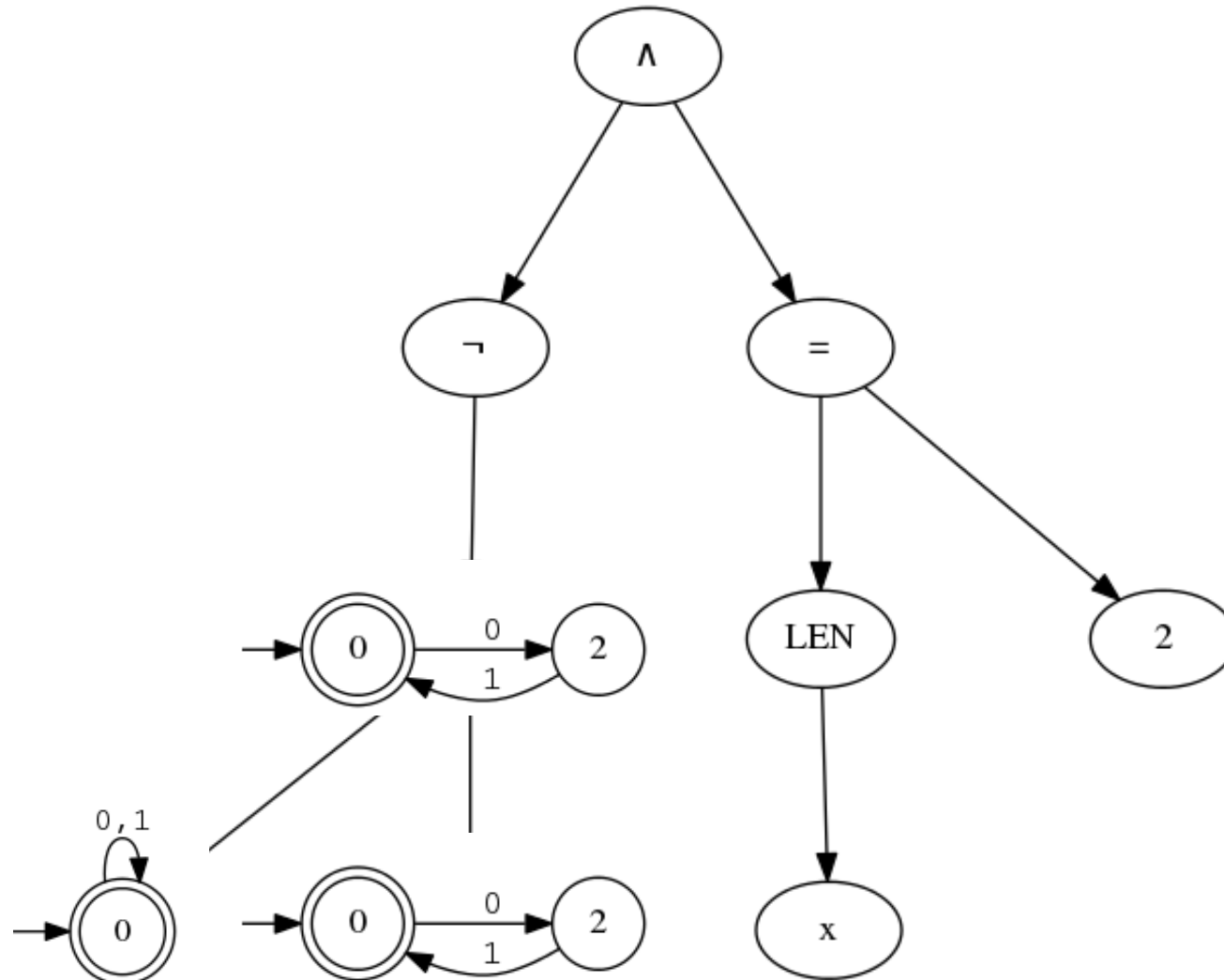
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



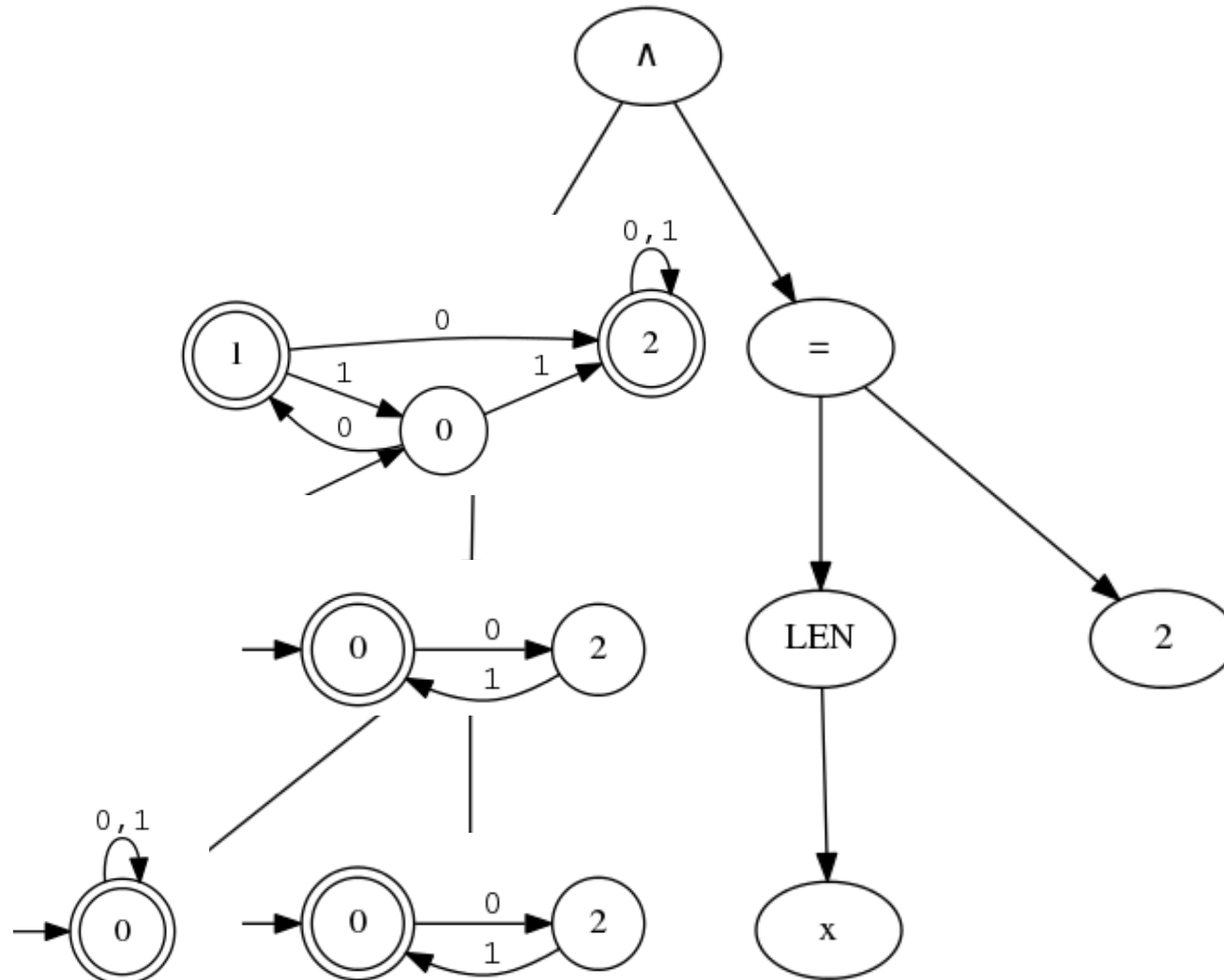
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



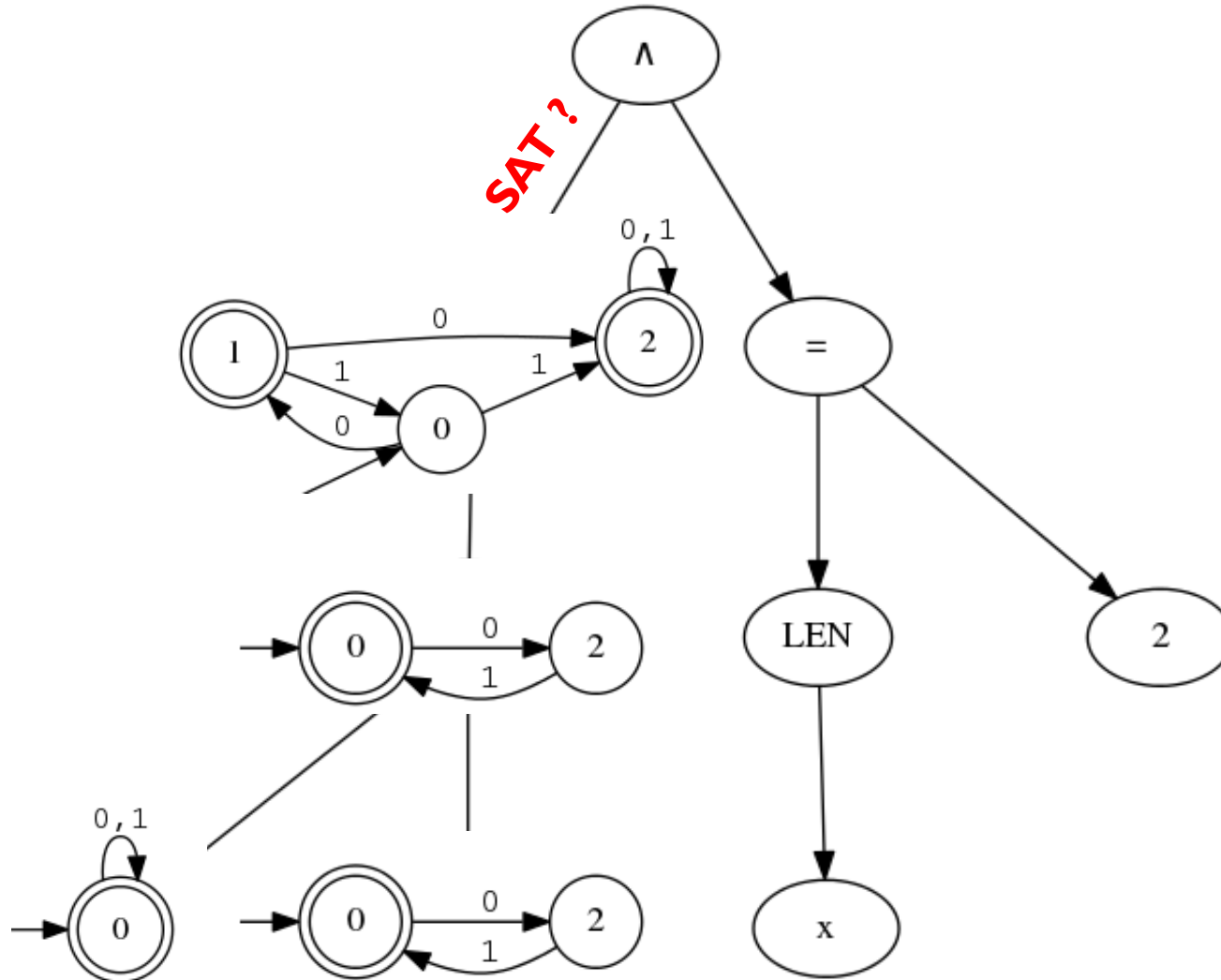
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



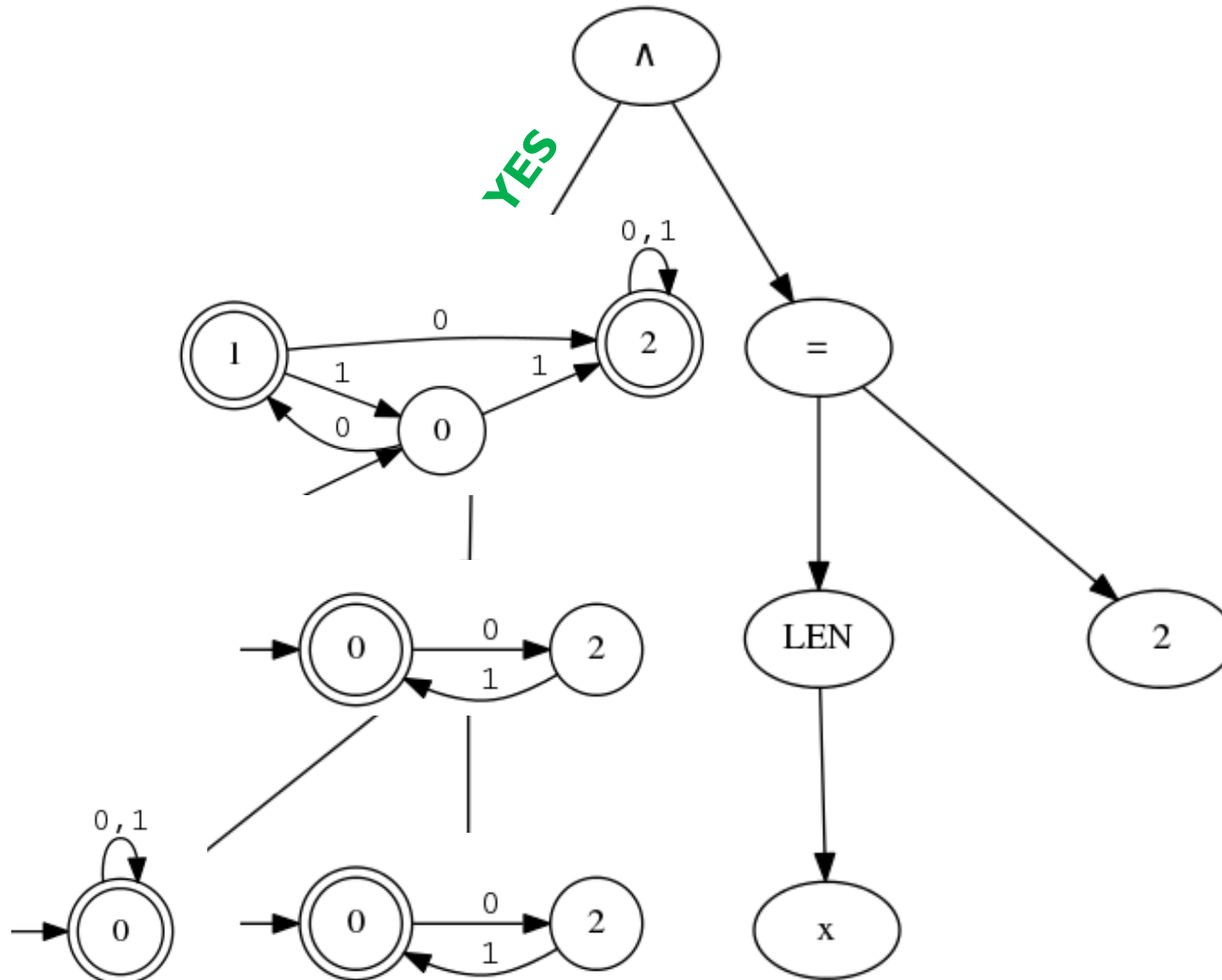
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



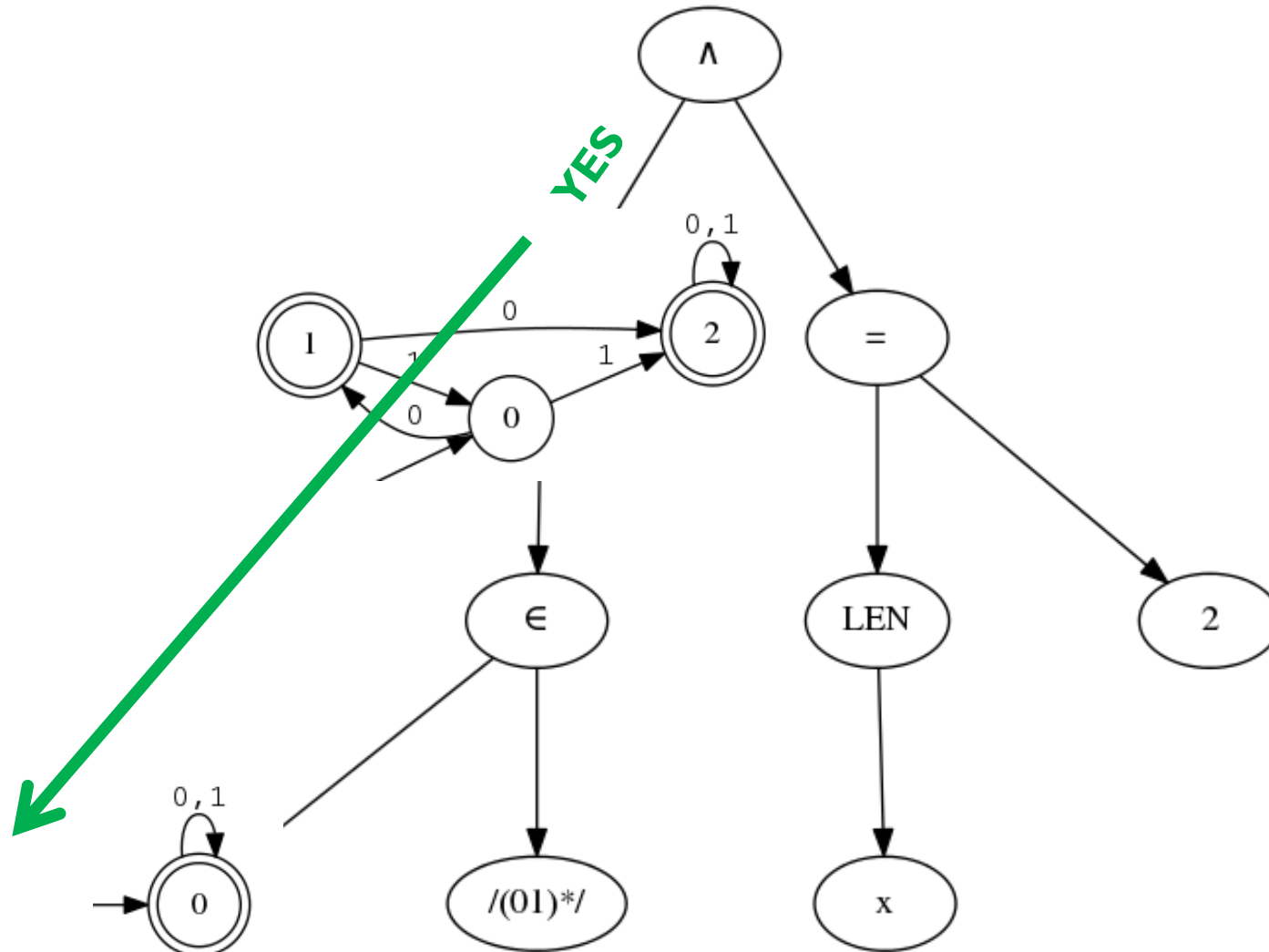
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



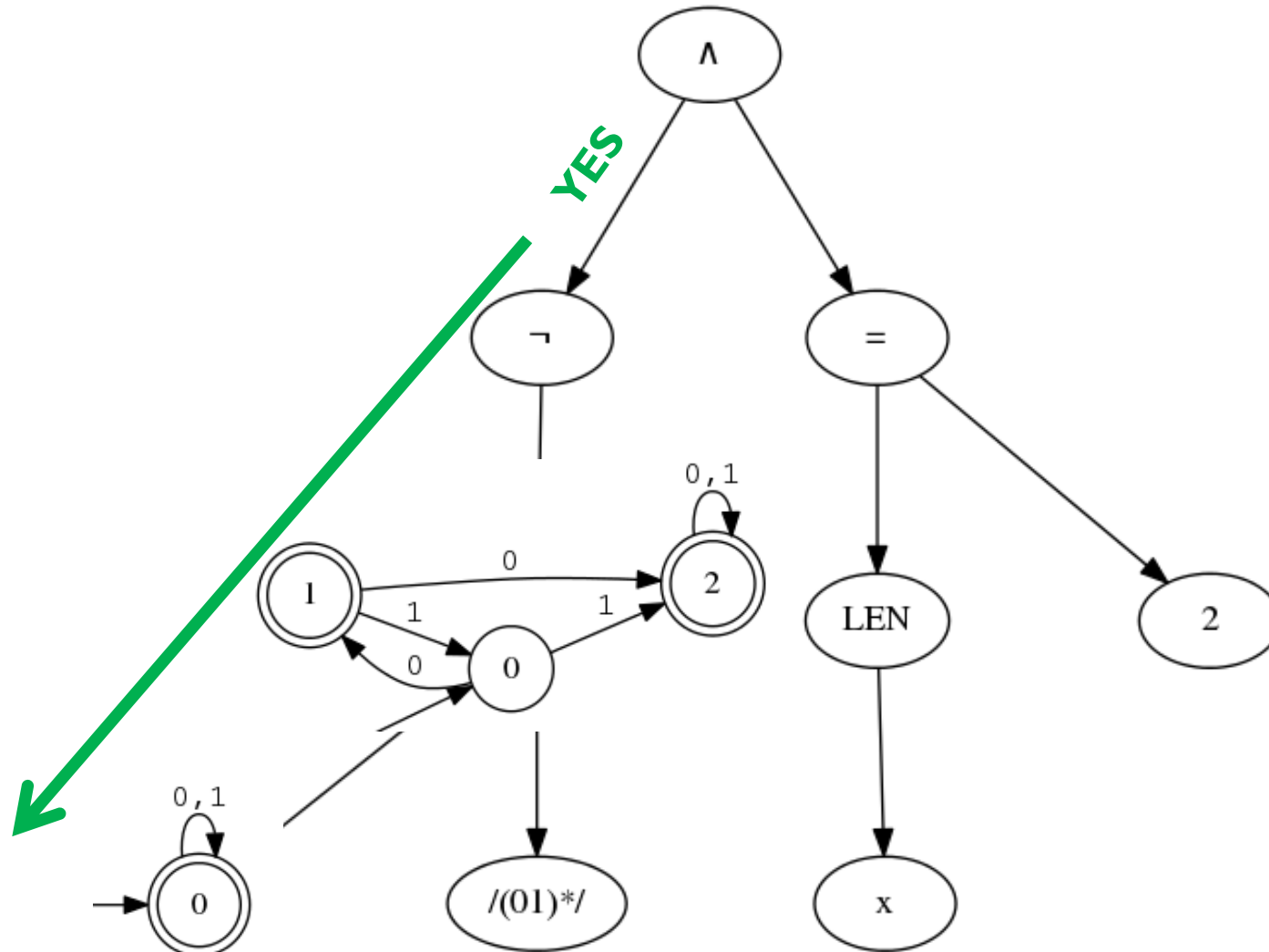
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



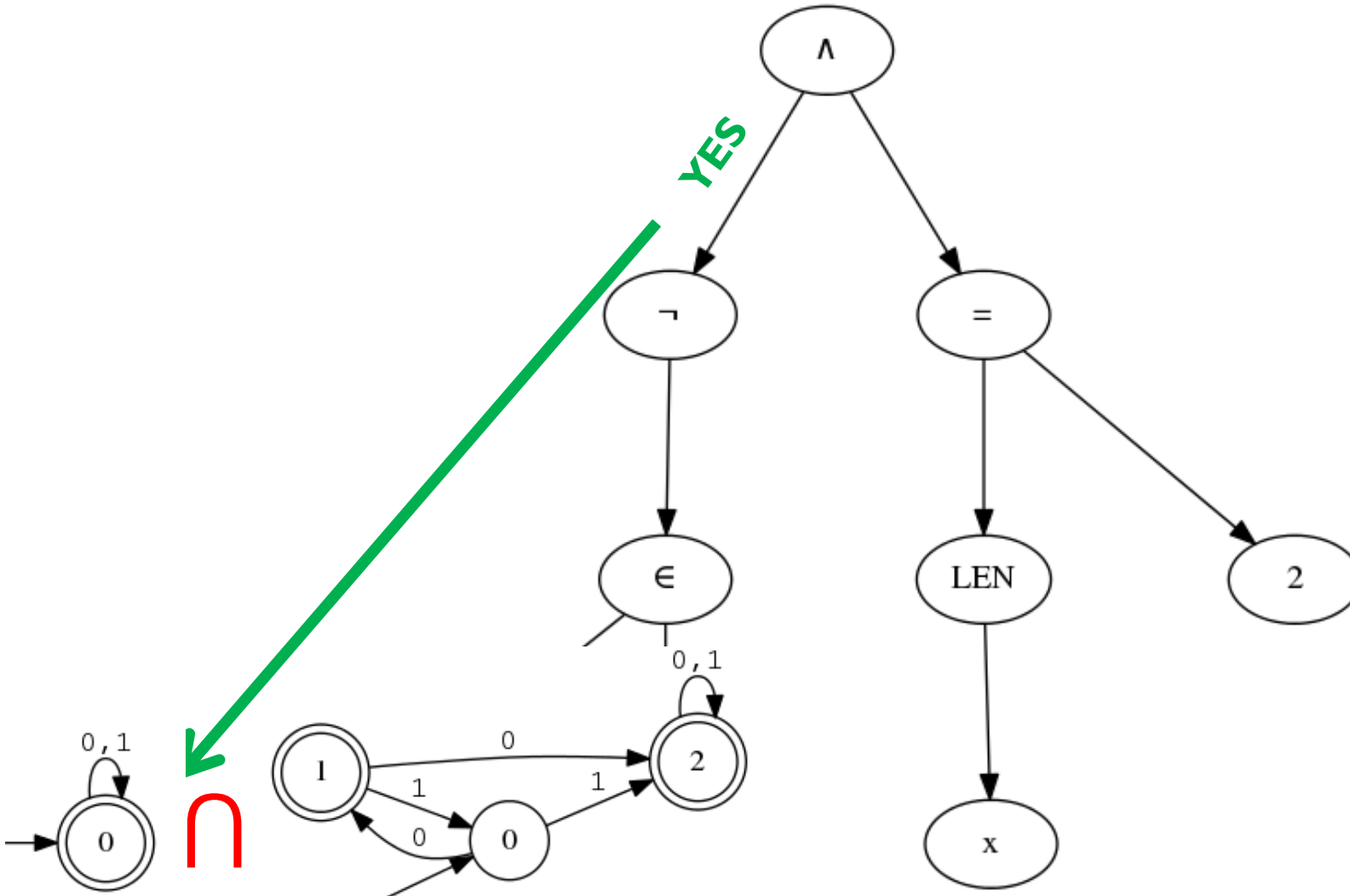
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



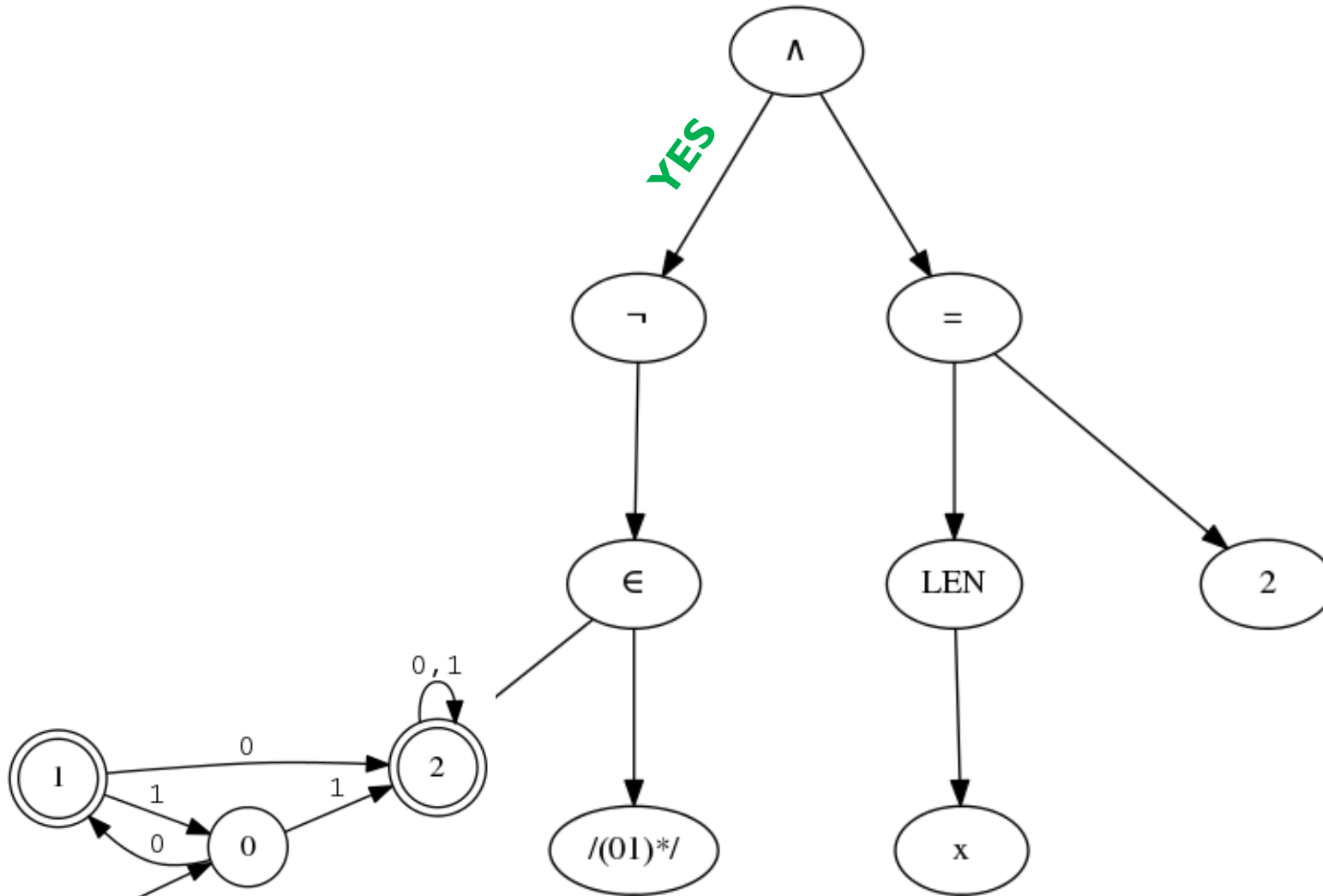
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



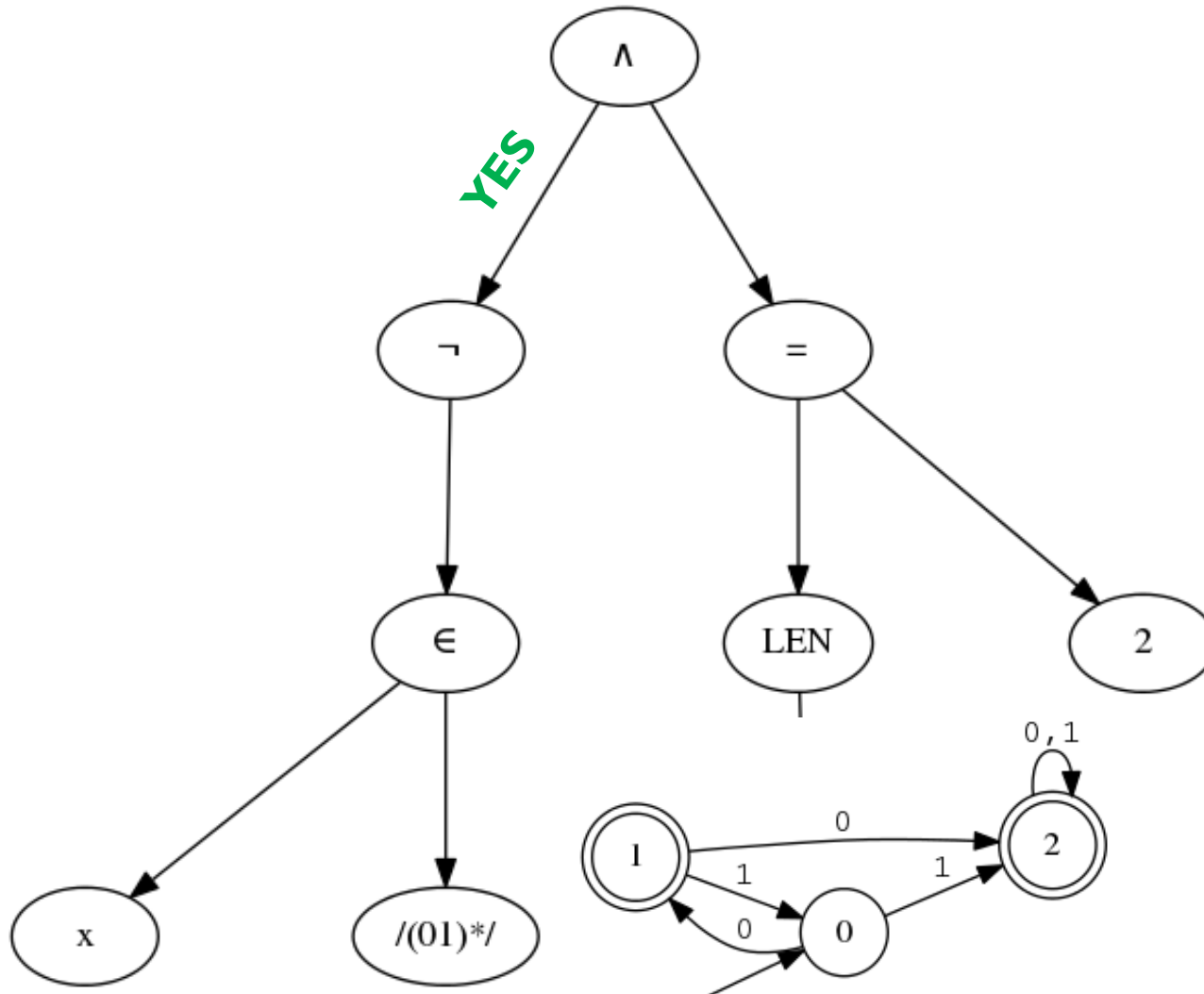
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



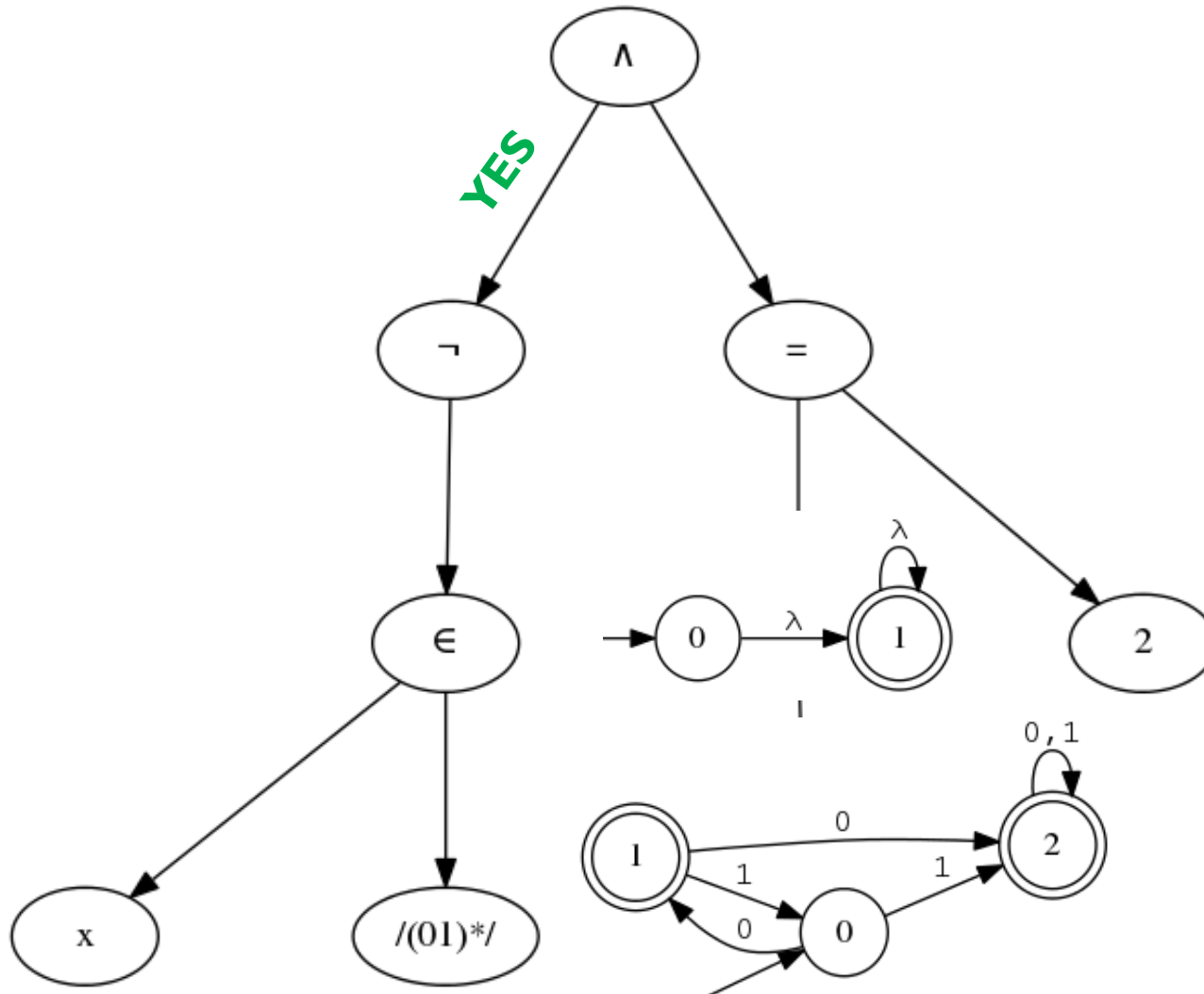
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



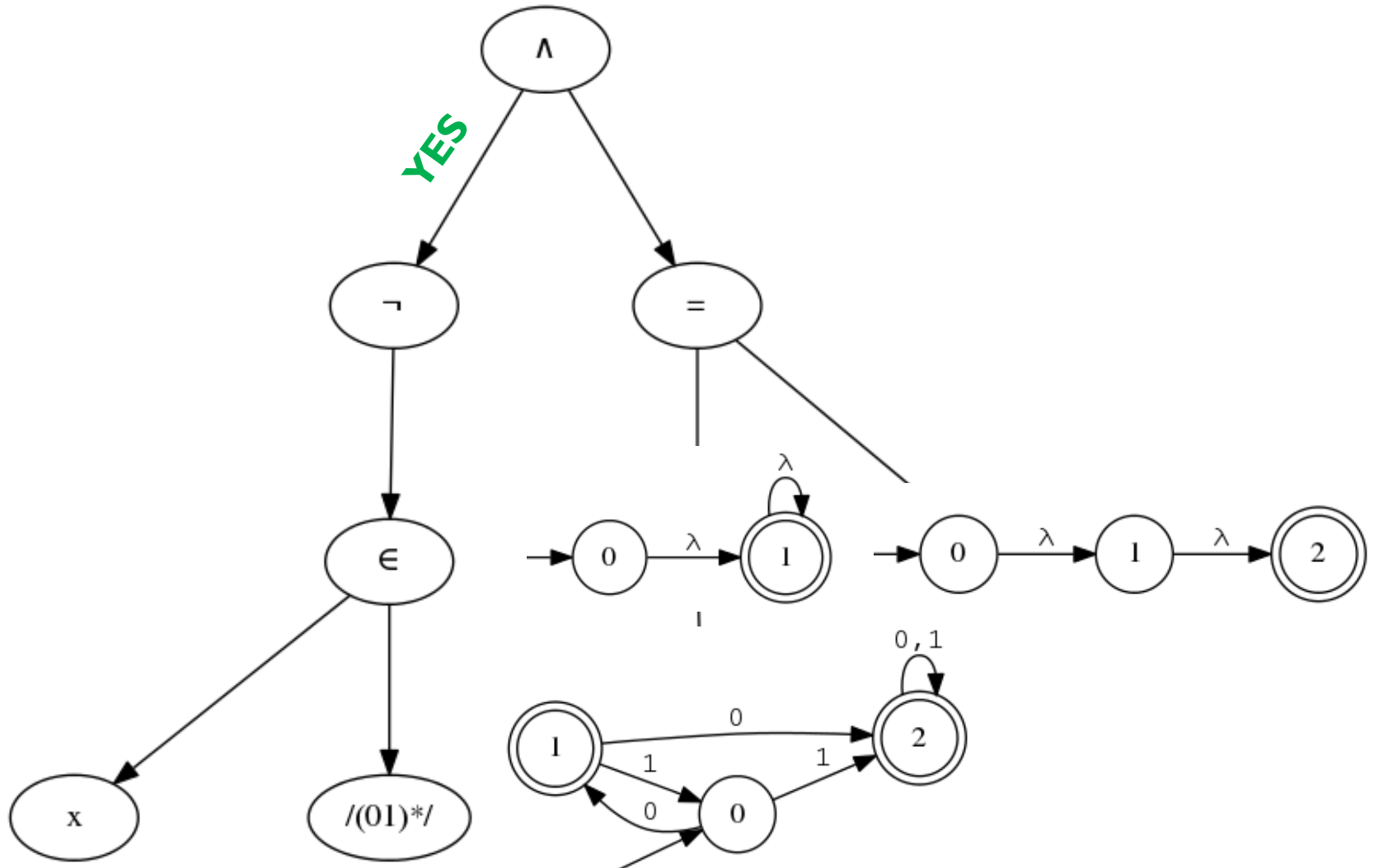
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



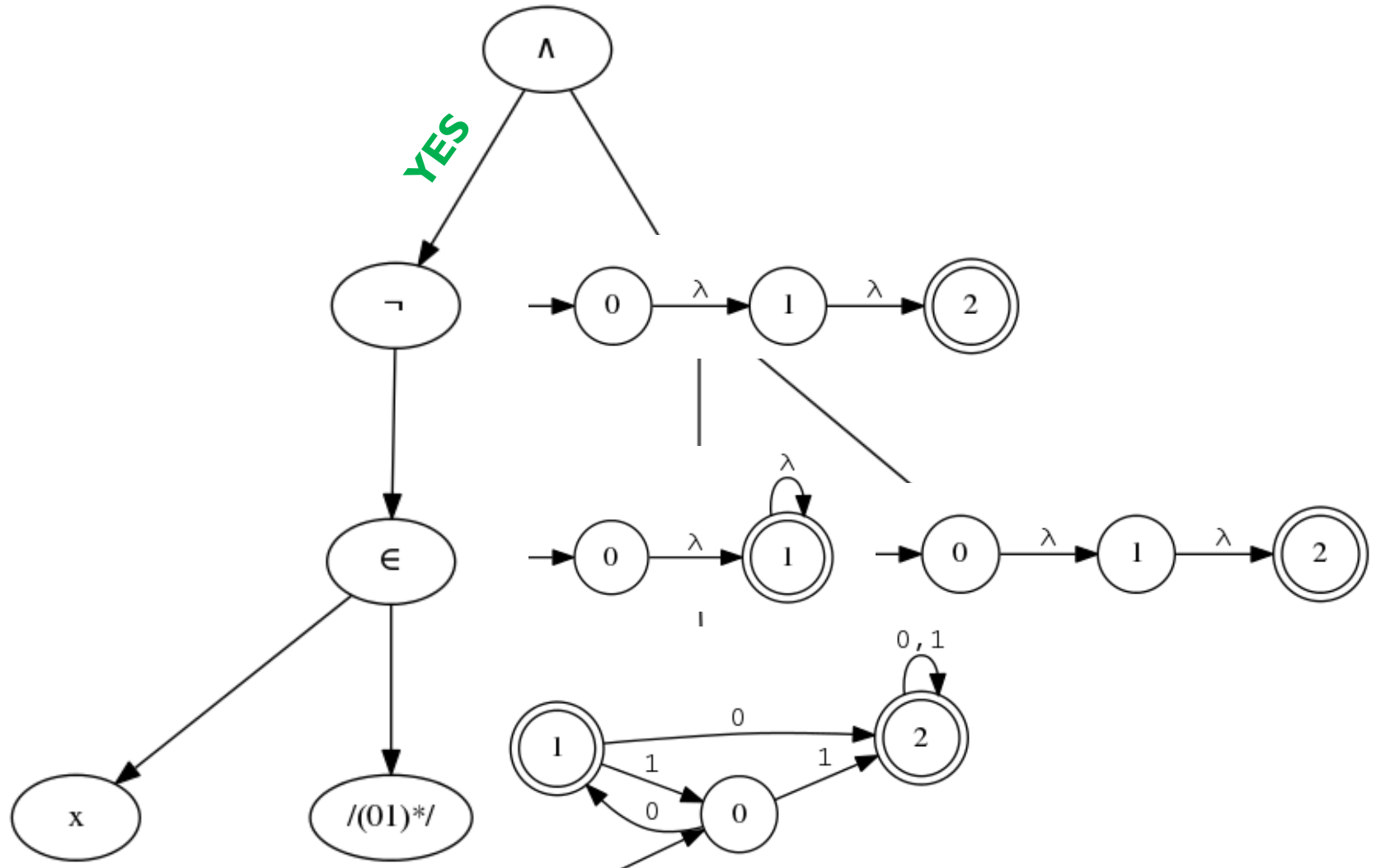
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



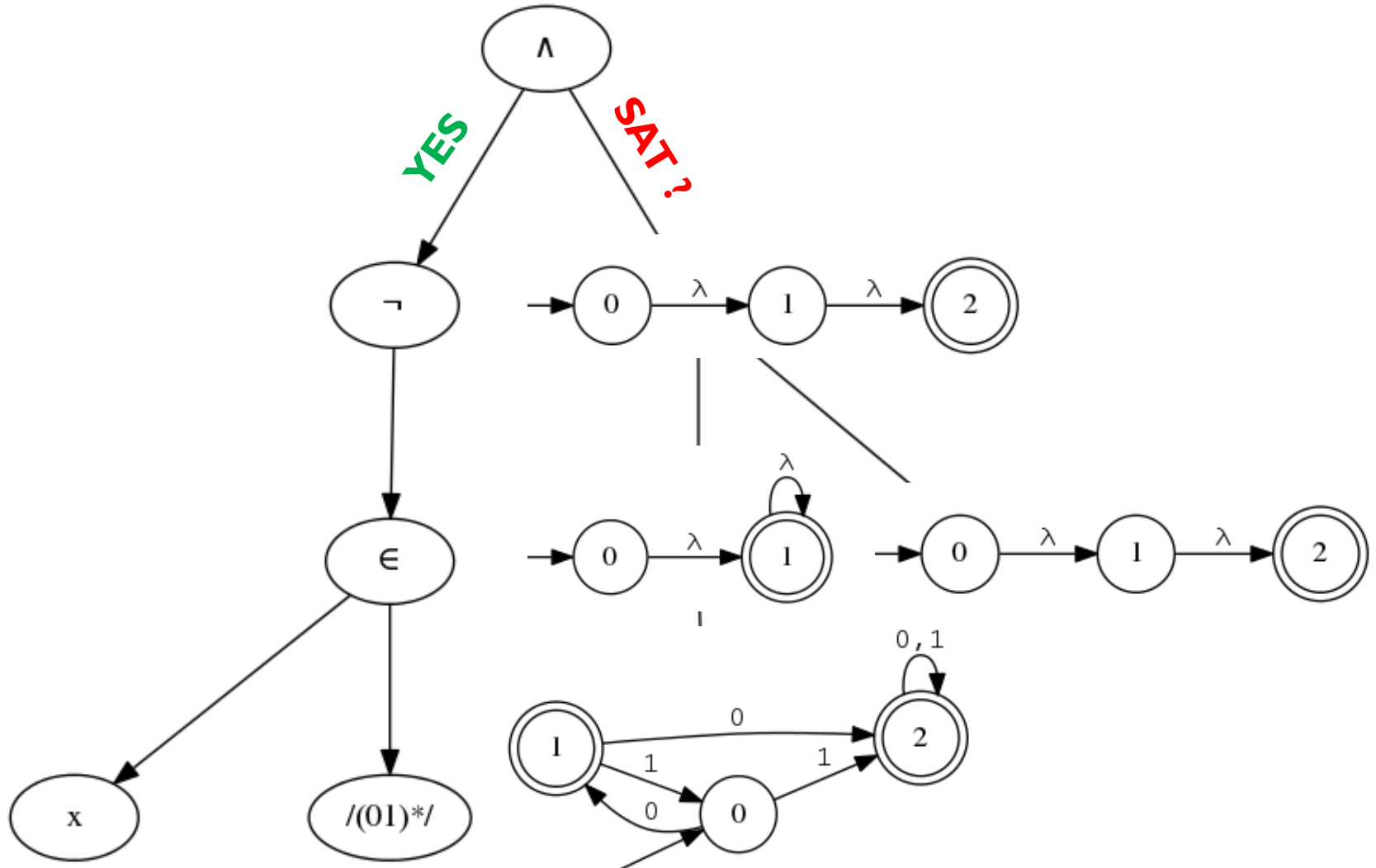
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



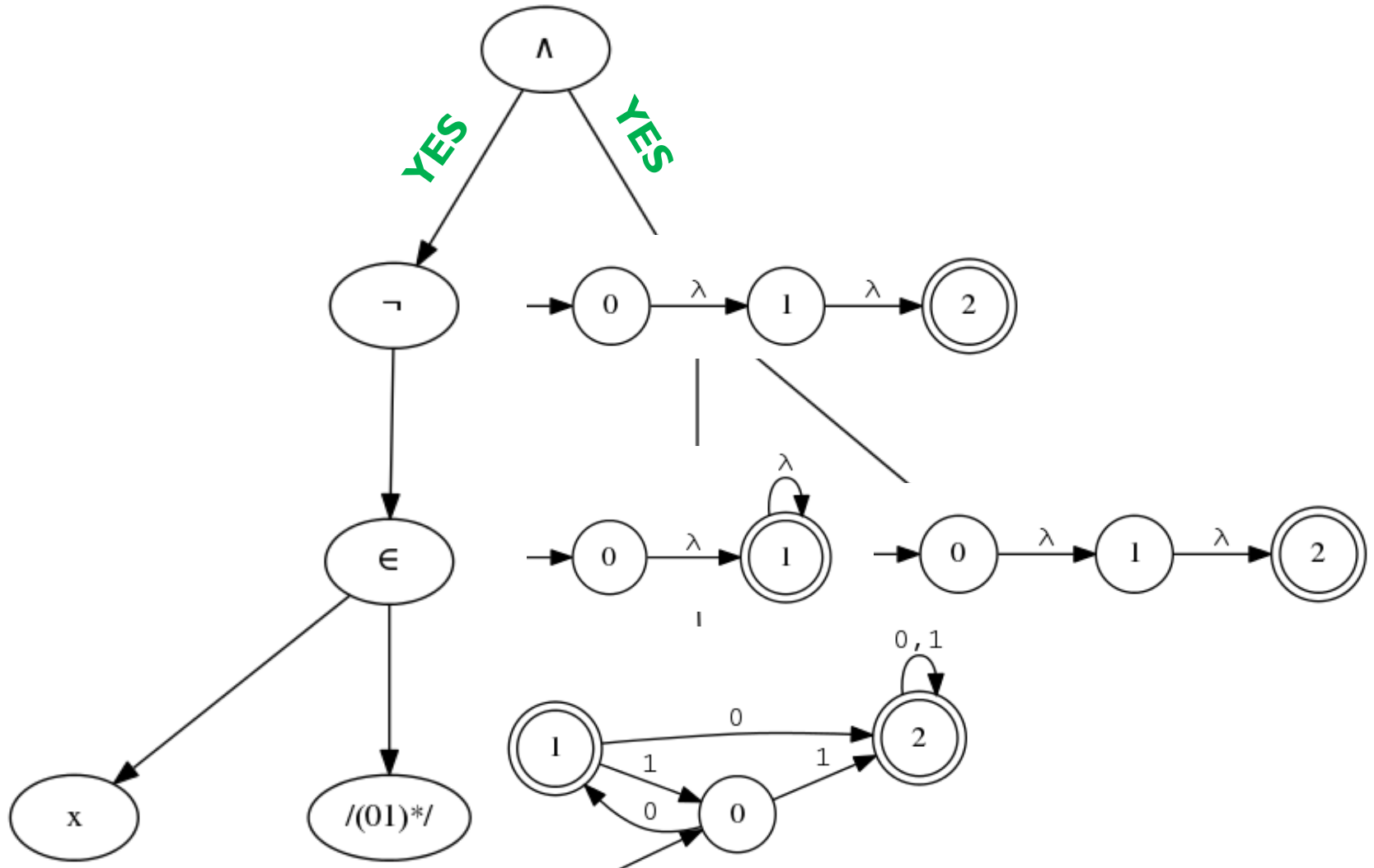
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



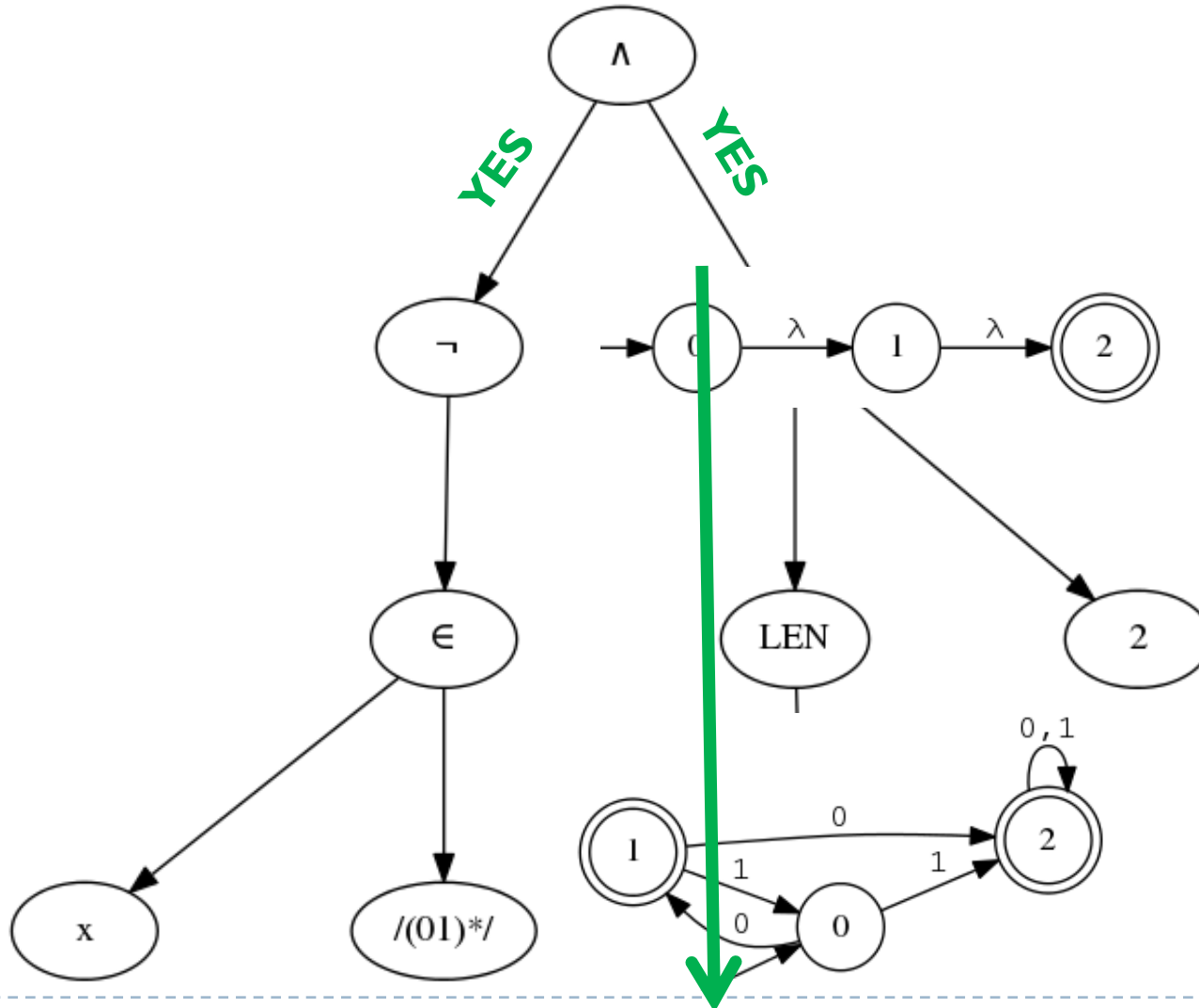
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



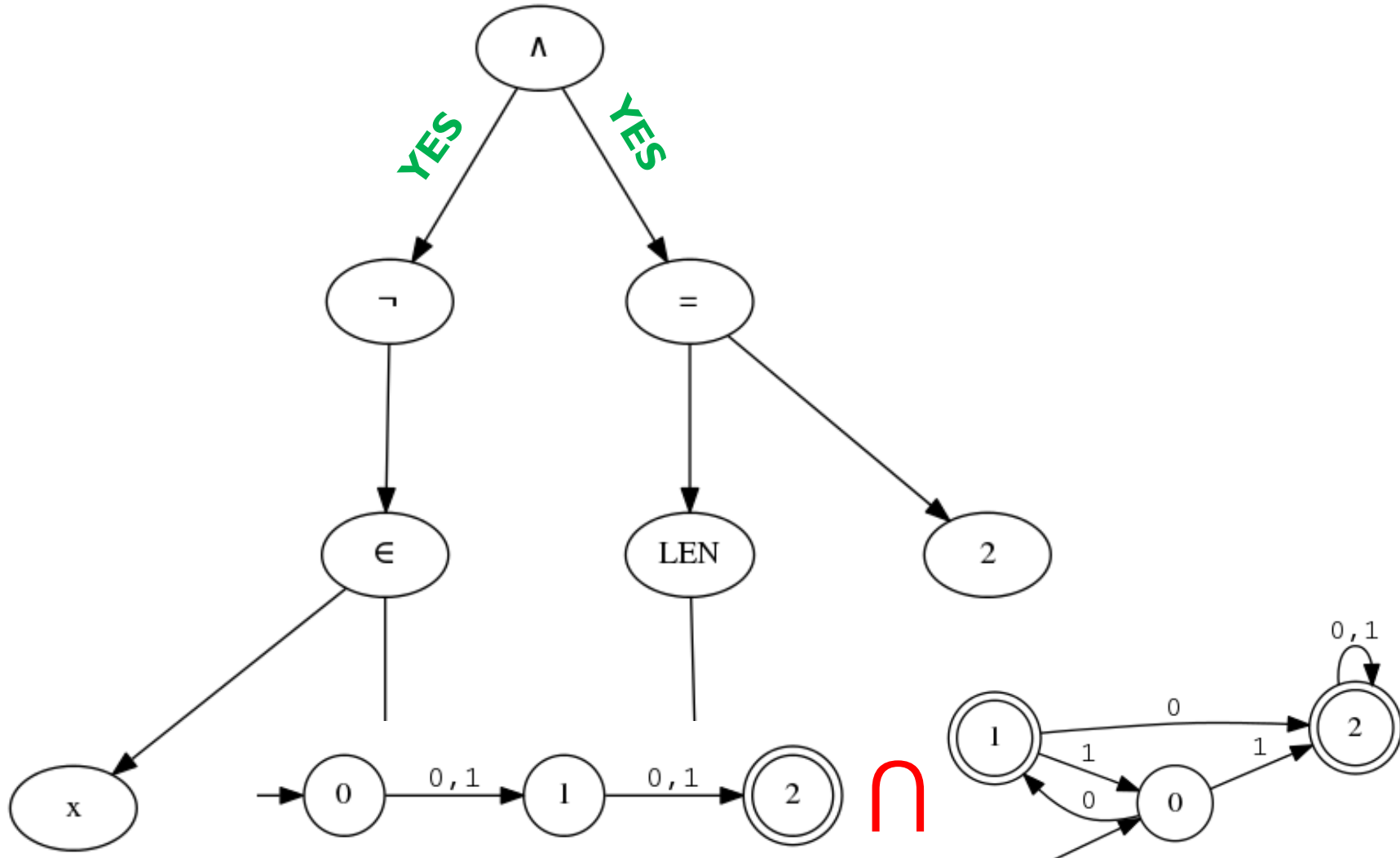
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



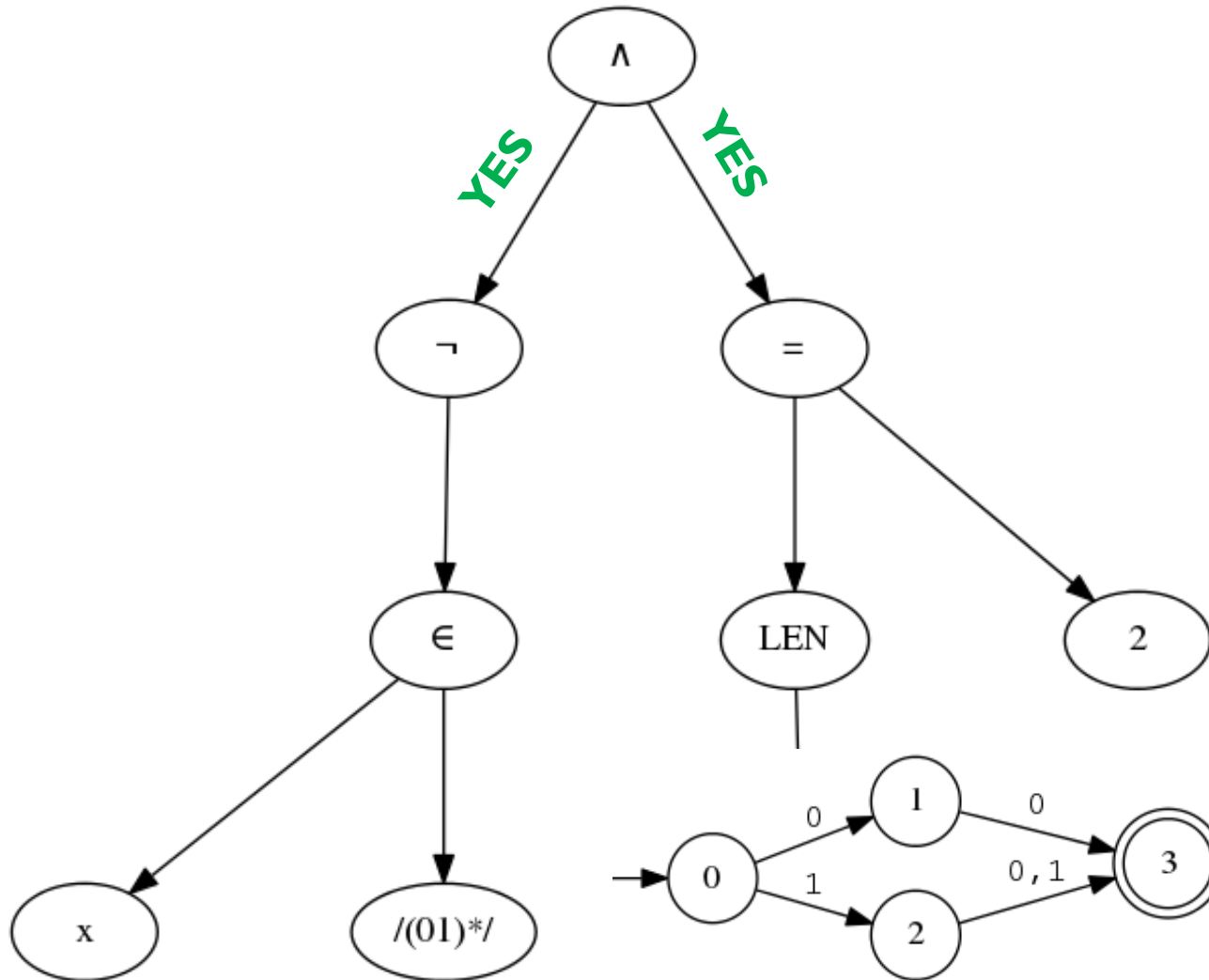
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



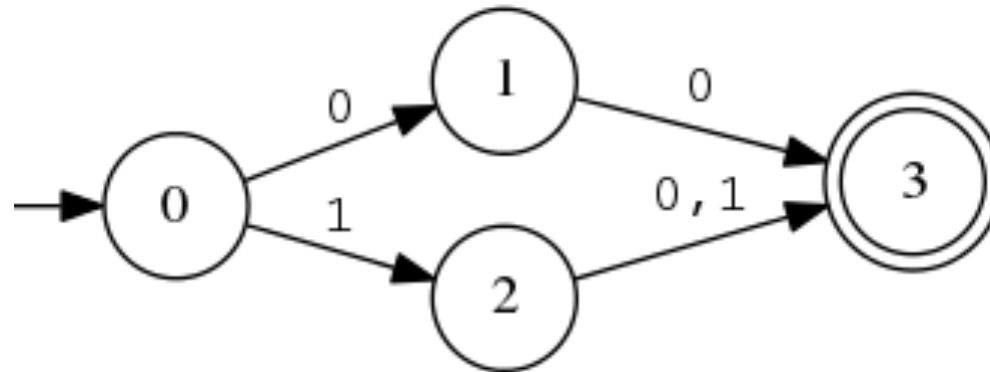
String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



String Automata Construction

$$C \equiv \neg(x \in (01)^*) \wedge \text{LEN}(x) = 2$$



00, 10, 11



Integer Constraints

$C \rightarrow bterm$

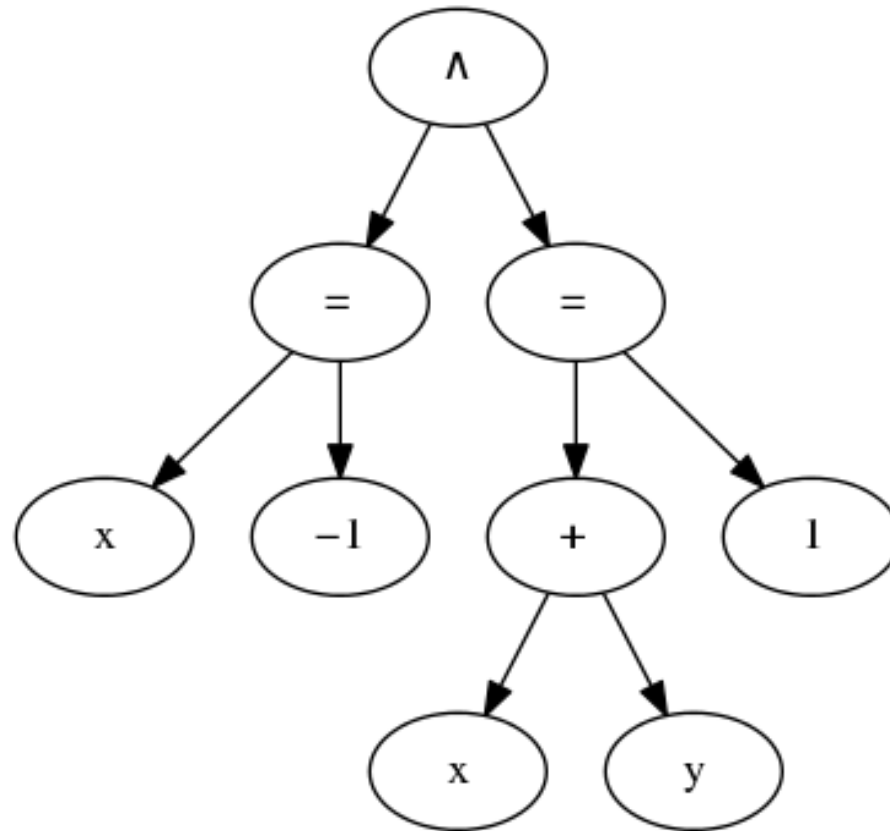
$bterm \rightarrow v \mid true \mid false$
| $\neg bterm \mid bterm \wedge bterm \mid bterm \vee bterm \mid (bterm)$
| $stern = stern$
| $match(stern, stern)$
| $contains(stern, stern)$
| $begins(stern, stern)$
| $ends(stern, stern)$
| $iterm = iterm \mid iterm < iterm \mid iterm > iterm$

$iterm \rightarrow v \mid n$
| $iterm + iterm \mid iterm - iterm \mid iterm \times n \mid (iterm)$
| $length(stern) \mid toint(stern)$
| $indexof(stern, stern)$
| $lastindexof(stern, stern)$

$stern \rightarrow v \mid \epsilon \mid s$
| $stern.stern \mid stern|stern \mid stern^* \mid (stern)$
| $charat(stern, iterm) \mid tostring(iterm)$
| $toupper(stern) \mid tolower(stern)$
| $substring(stern, iterm, iterm)$
| $replacefirst(stern, stern, stern)$
| $replacelast(stern, stern, stern)$
| $replaceall(stern, stern, stern)$

Integer Automata Construction

$$C \equiv x = -1 \wedge x + y = 1$$

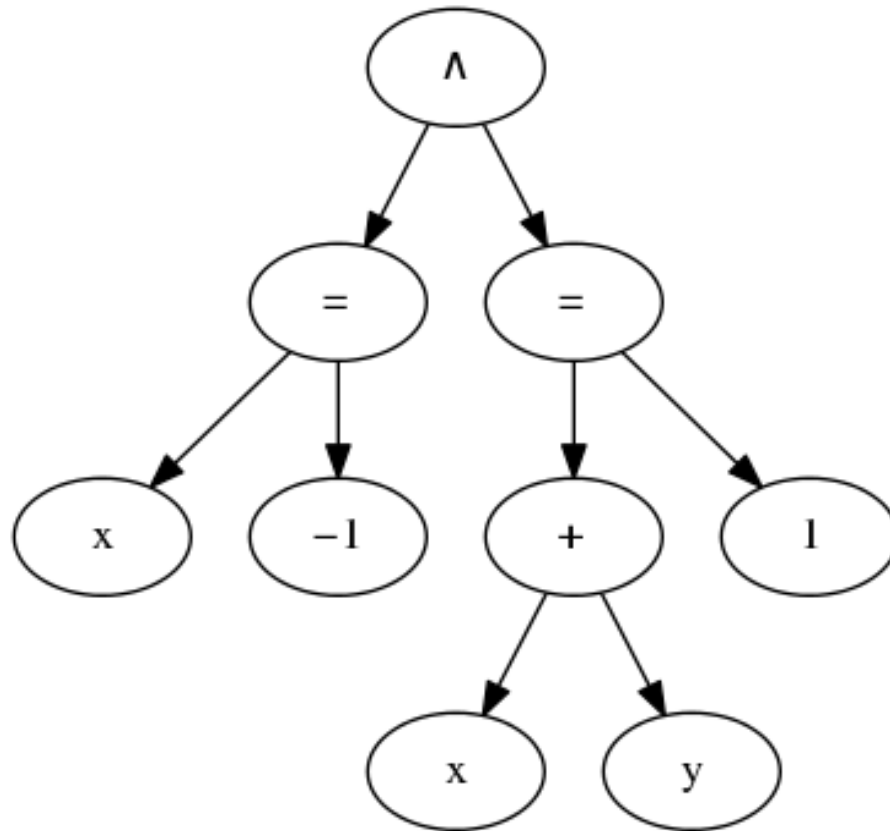


Integer Automata Construction

$$C \equiv x = -1 \wedge x + y = 1$$

$$C_1 \equiv x + 0 * y + 1 = 0 \Rightarrow [1 \ 0 \ 1]$$

$$C_2 \equiv x + y - 1 = 0 \Rightarrow [1 \ 1 \ -1]$$

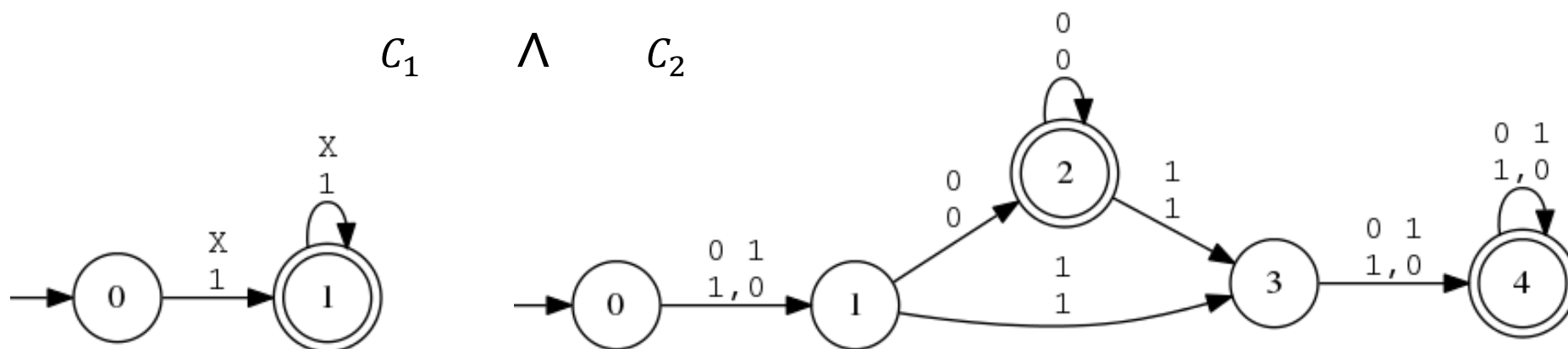


Integer Automata Construction

$$C \equiv x = -1 \wedge x + y = 1$$

$$C_1 \equiv x + 0 * y + 1 = 0 \Rightarrow [1 \ 0 \ 1]$$

$$C_2 \equiv x + y - 1 = 0 \Rightarrow [1 \ 1 \ -1]$$

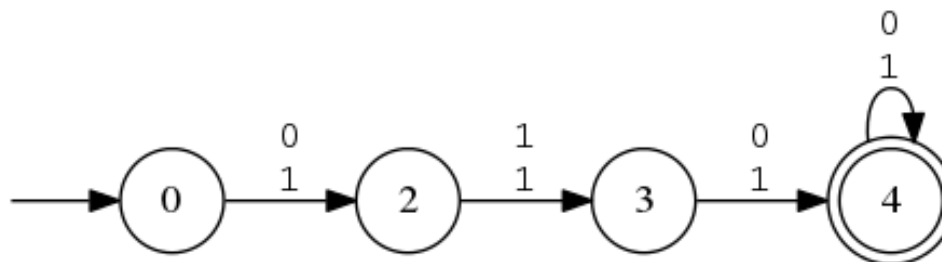


- ▶ Using automata construction techniques described in:

C. Bartzis and Tefvik Bultan. Efficient symbolic representations for arithmetic constraints in verification. *Int. J. Found. Comput. Sci.*, 2003

Integer Automata Construction

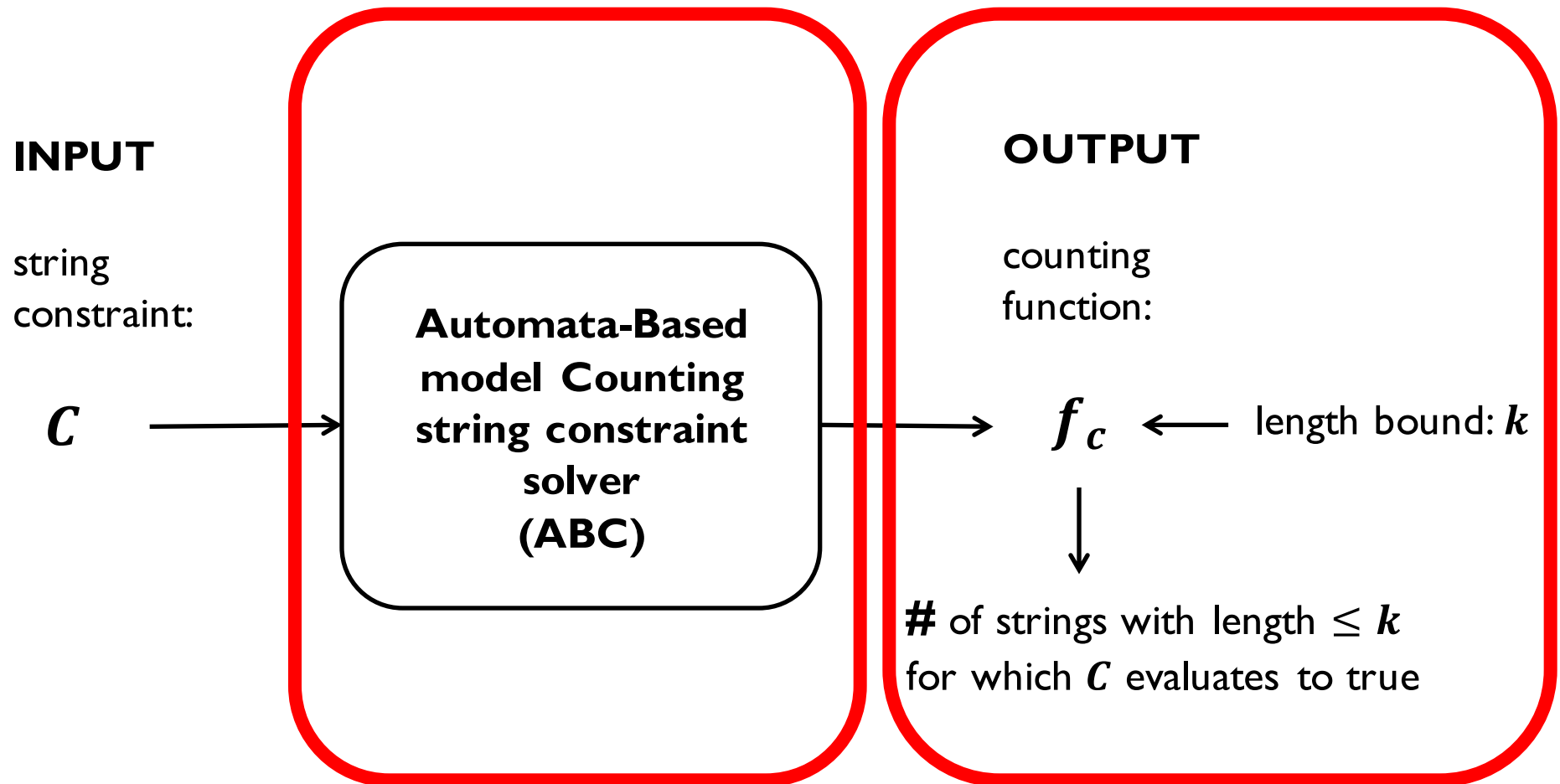
$$C \equiv x = -1 \wedge x + y = 1$$



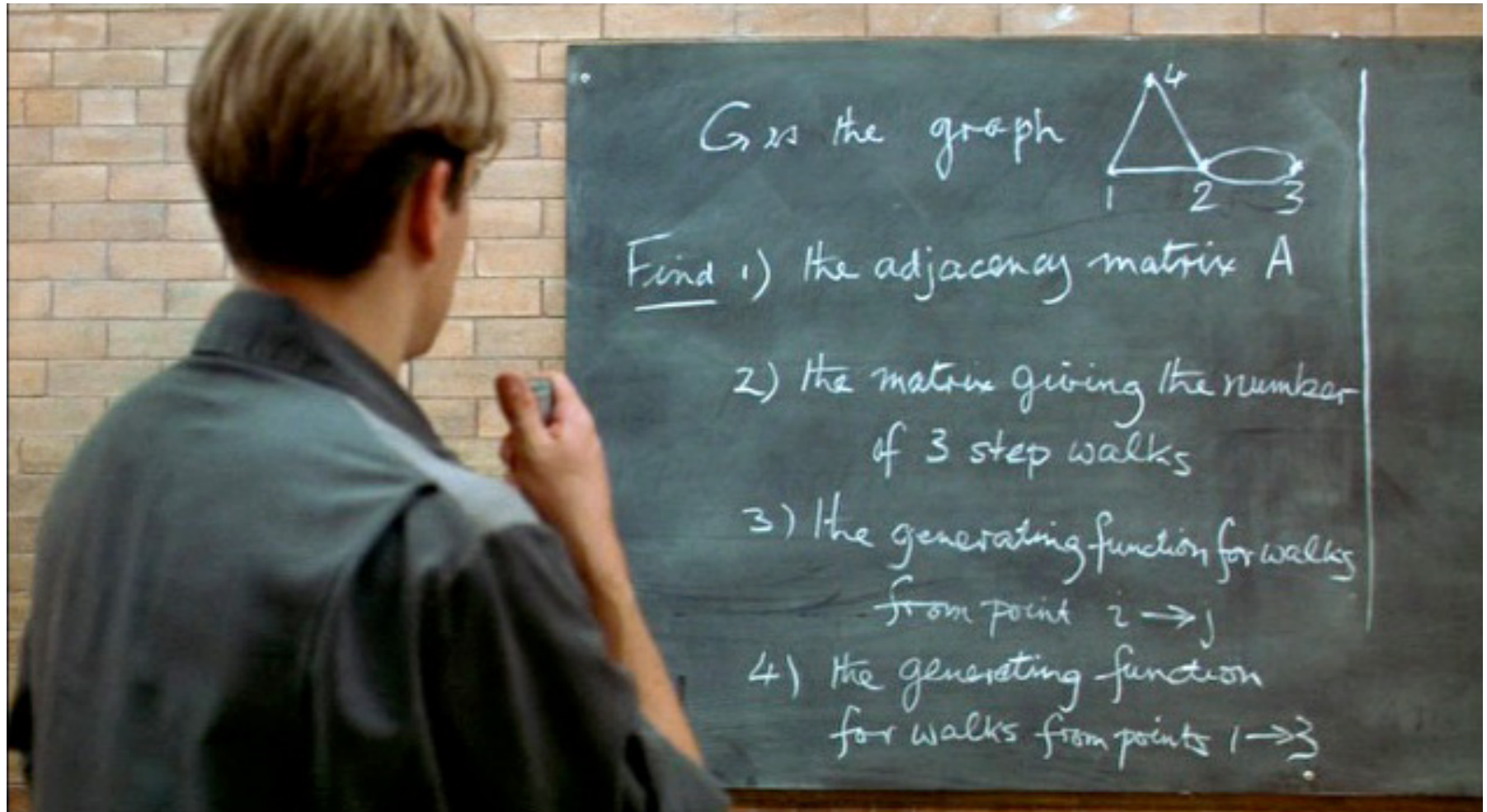
$$(111, 010) = (-1, 2)$$

- ▶ Conjunction and disjunction is handled by automata product, negation is handled by automata complement

Model Counting String Constraints Solver

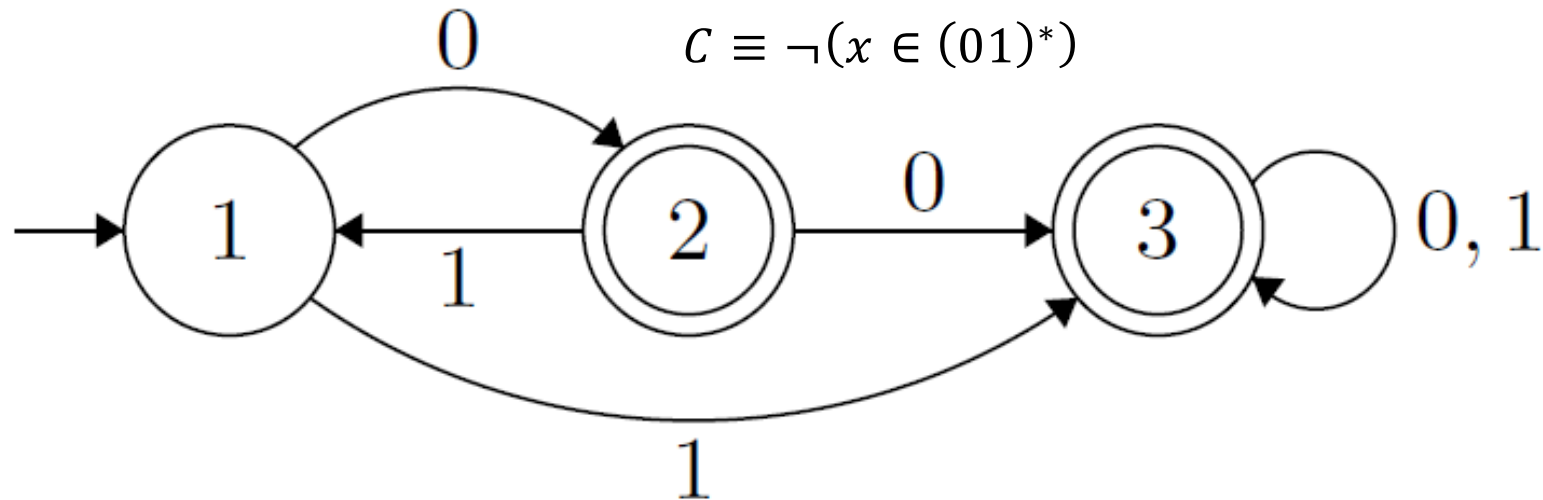


Can you solve it Will Hunting?



Automata-based Model Counting

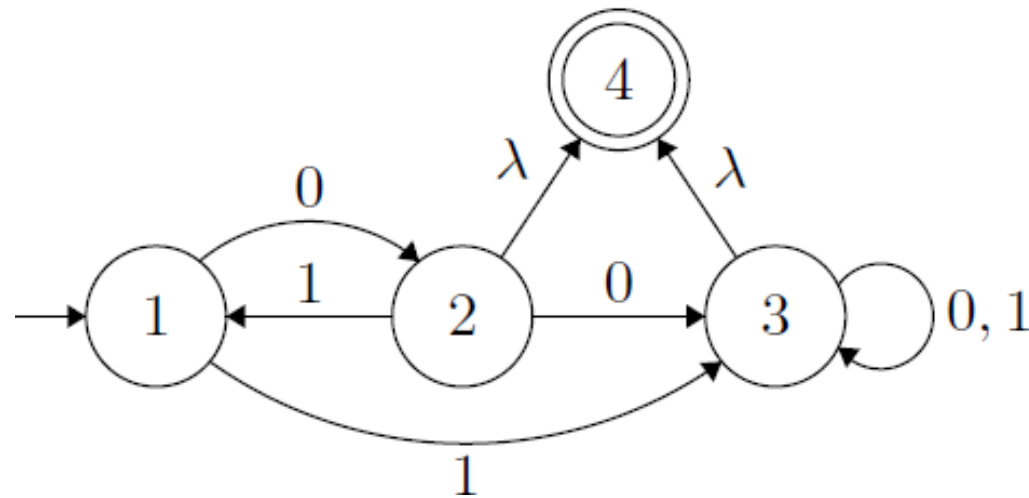
- ▶ Converting constraints to automata reduces the model counting problem to path counting problem in graphs



- ▶ We will generate a function $f(k)$
 - ▶ Given length bound k , it will count the number of paths with length k .
 - ▶ $f(0) = 0, \{\}$
 - ▶ $f(1) = 2, \{0,1\}$
 - ▶ $f(2) = 3, \{00,10,11\}$

Path Counting

$$C = \neg(x \in (01)^*)$$



$$T = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, T^2 = \begin{bmatrix} 1 & 0 & 3 & 2 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & 4 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix}, T^3 = \begin{bmatrix} 0 & 1 & 7 & 3 \\ 1 & 0 & 7 & 4 \\ 0 & 0 & 8 & 4 \\ 0 & 0 & 0 & 0 \end{bmatrix}, T^4 = \begin{bmatrix} 0 & 1 & 15 & 8 \\ 1 & 0 & 15 & 7 \\ 0 & 0 & 16 & 8 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

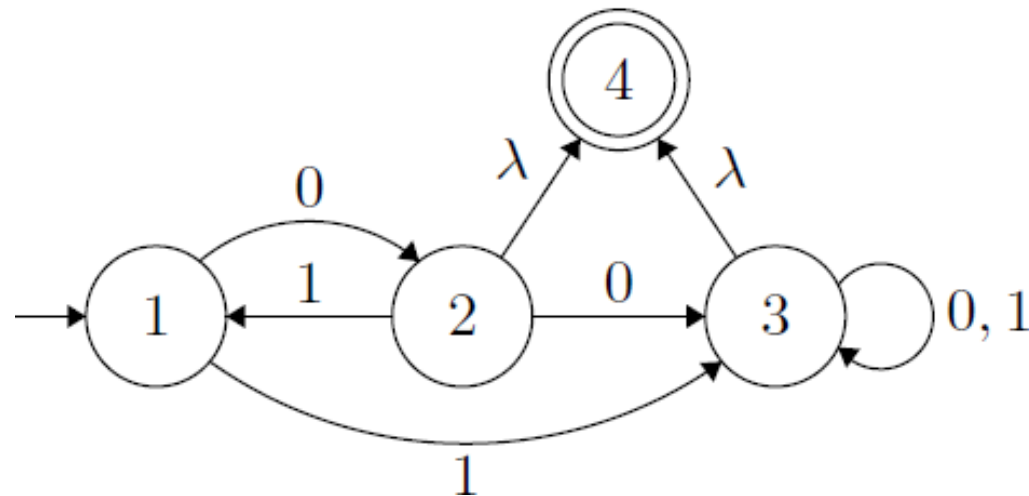
$$f(0) = 0$$

$$f(1) = 2$$

$$f(2) = 3$$

$$f(3) = 8$$

Recurrence Relation



$$f(4, k) = f(2, k - 1) + f(3, k - 1)$$

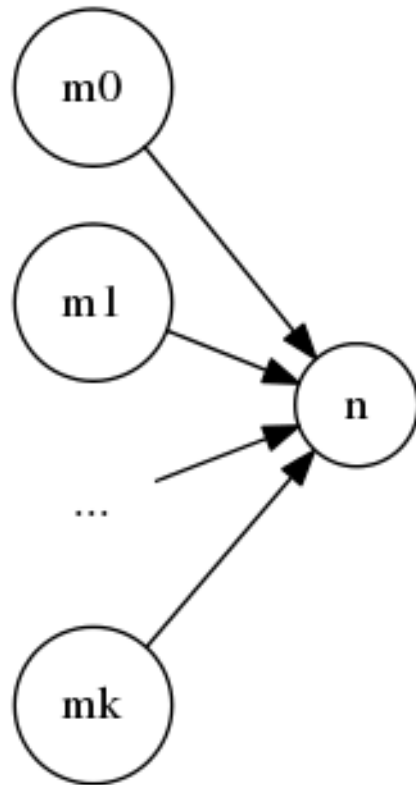
$$f(3, k) = f(1, k - 1) + f(2, k - 1) + f(3, k - 1)$$

$$f(2, k) = f(1, k - 1)$$

$$f(1, k) = f(2, k - 1)$$

$$f(1, 0) = 1, f(2, 0) = 0, f(3, 0) = 0, f(4, 0) = 0$$

Recurrence Relation



$$f(n, k) = \sum_{(m,n) \in E} f(m, k - 1)$$

$$f(0, 0) = 1$$

$$f(1, 0) = 0$$

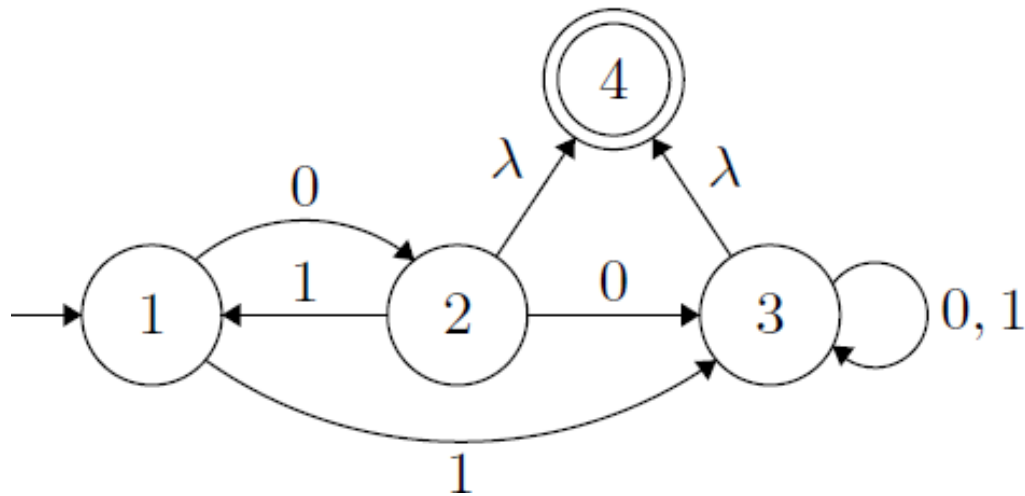
$$f(2, 0) = 0$$

\dots

$$f(i, 0) = 0$$

Recurrence Relation

- ▶ We can solve system of recurrence relations for final node

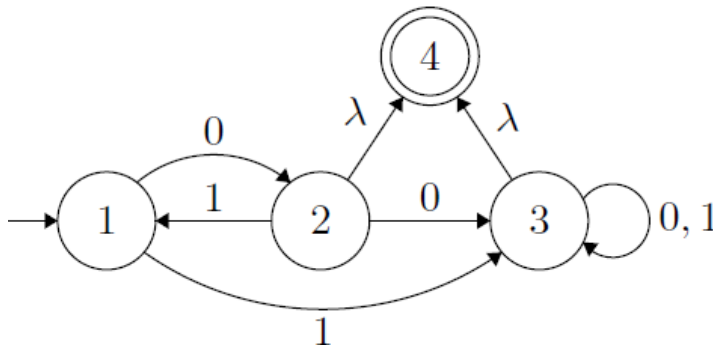


$$f(0) = 0, f(1) = 2, f(2) = 3$$

$$f(k) = 2f(k-1) + f(k-2) - 2f(k-3)$$

Counting Paths w Generating Functions

- ▶ We can compute a generating function, $g(z)$, for a DFA from the associated matrix



$$T = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$g(z) = (-1)^n \frac{\det(I - zT: n + 1, 1)}{z \times \det(I - zT)} = \frac{2z - z^2}{1 - 2z - z^2 + 2z^3}$$

Counting Paths w Generating Functions

$$g(z) = \frac{2z - z^2}{1 - 2z - z^2 + 2z^3}$$

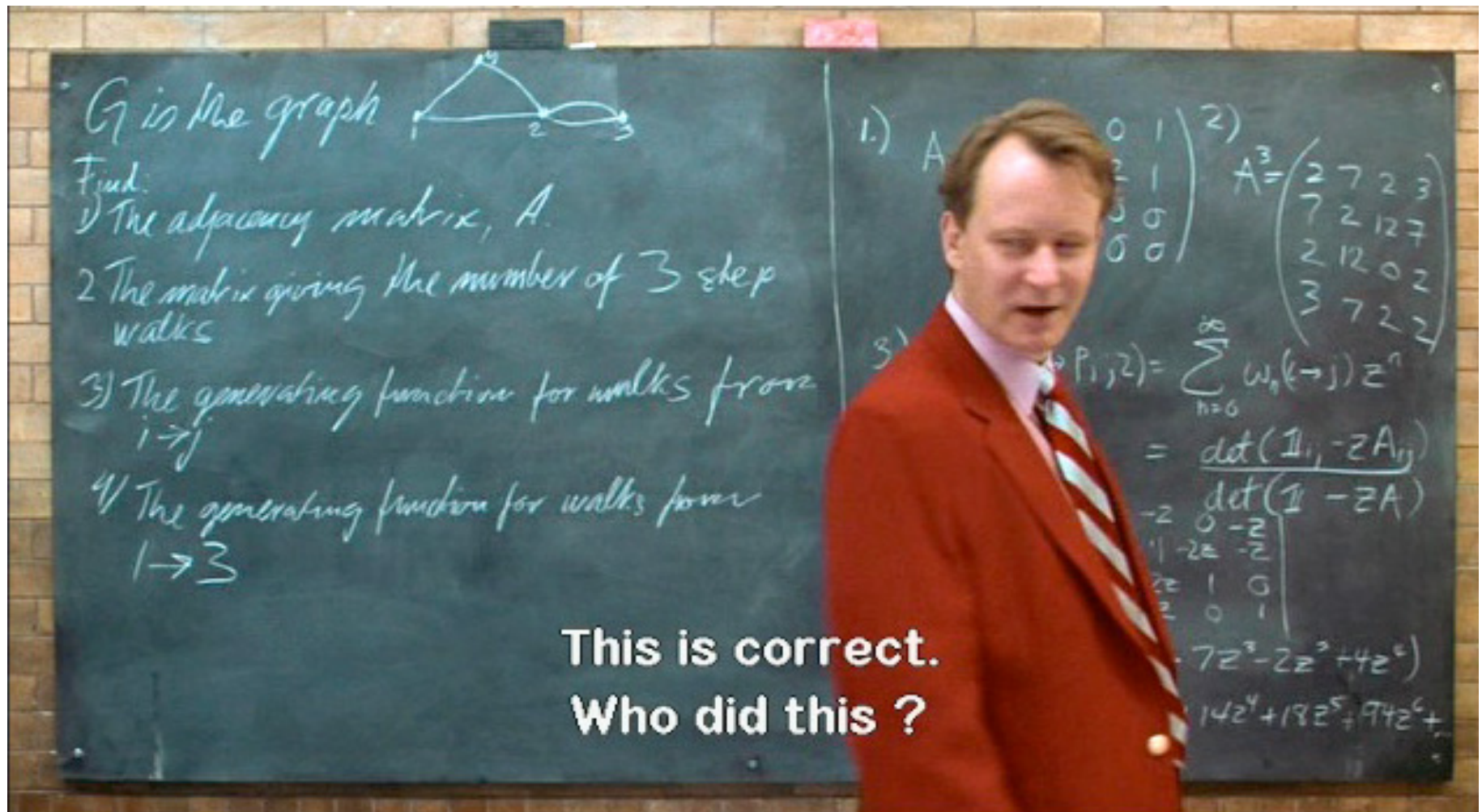
- ▶ Each $f(i)$ can be computed by Taylor expansion of $g(z)$


$$g(z) = \frac{g(0)}{0!}z^0 + \frac{g^{(1)}(0)}{1!}z^1 + \frac{g^{(2)}(0)}{2!}z^2 + \dots + \frac{g^{(n)}(0)}{n!}z^n + \dots$$

$$g(z) = 0z^0 + 2z^1 + 3z^2 + 8z^3 + 15z^4 + \dots$$

$$g(z) = f(0)z^0 + f(1)z^1 + f(2)z^2 + f(3)z^3 + f(4)z^4 + \dots$$

Good job Will Hunting!



G is the graph 

Find:

- 1) The adjacency matrix, A .
- 2) The matrix giving the number of 3 step walks
- 3) The generating function for walks from $i \rightarrow j$
- 4) The generating function for walks from $1 \rightarrow 3$

1.) $A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ 2.) $A^3 = \begin{pmatrix} 2 & 7 & 2 & 3 \\ 7 & 2 & 12 & 7 \\ 2 & 12 & 0 & 2 \\ 3 & 7 & 2 & 2 \end{pmatrix}$

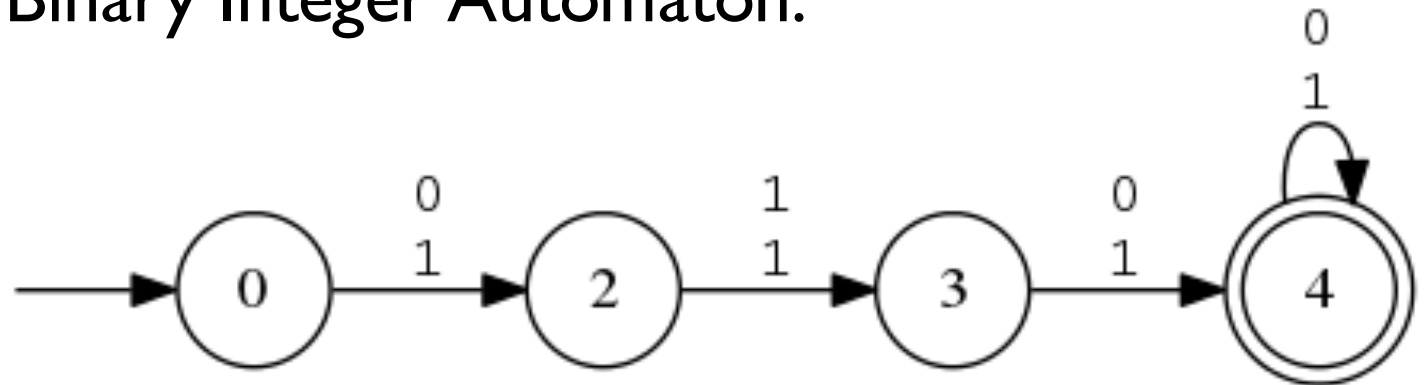
3.) $f_i(j; z) = \sum_{n=0}^{\infty} w_n(i \rightarrow j) z^n$
 $= \frac{\det(\mathbb{1}_i, -zA_{ij})}{\det(\mathbb{1} - zA)}$

4.) $\begin{vmatrix} -2 & 0 & -z \\ 1 & -2z & -z \\ z & 1 & 0 \\ z & 0 & 1 \end{vmatrix} = -7z^3 - 2z^2 + 4z^0$
 $14z^4 + 19z^5 + 94z^6 + \dots$

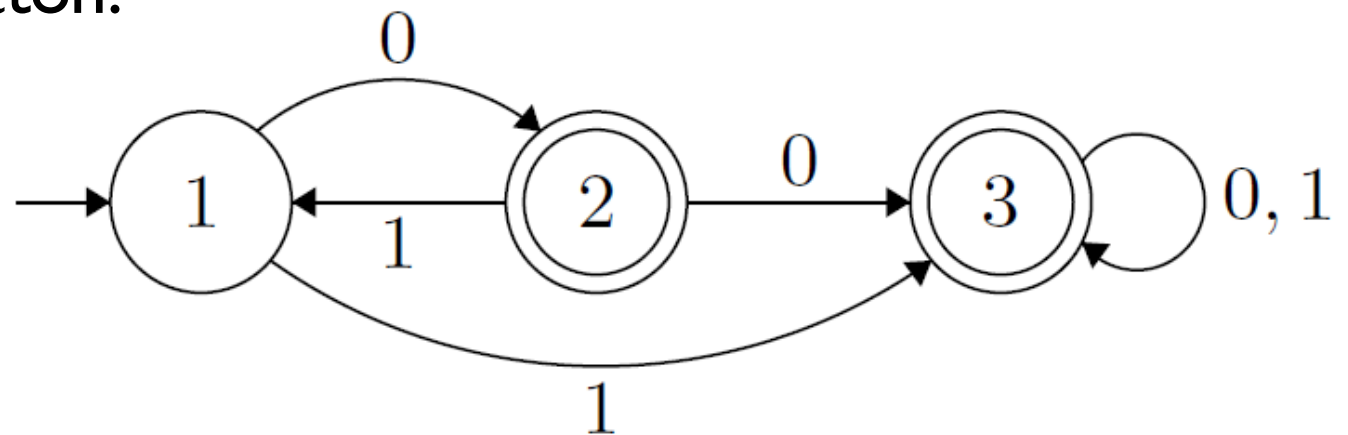
**This is correct.
Who did this ?**

Applicable to Both Automata

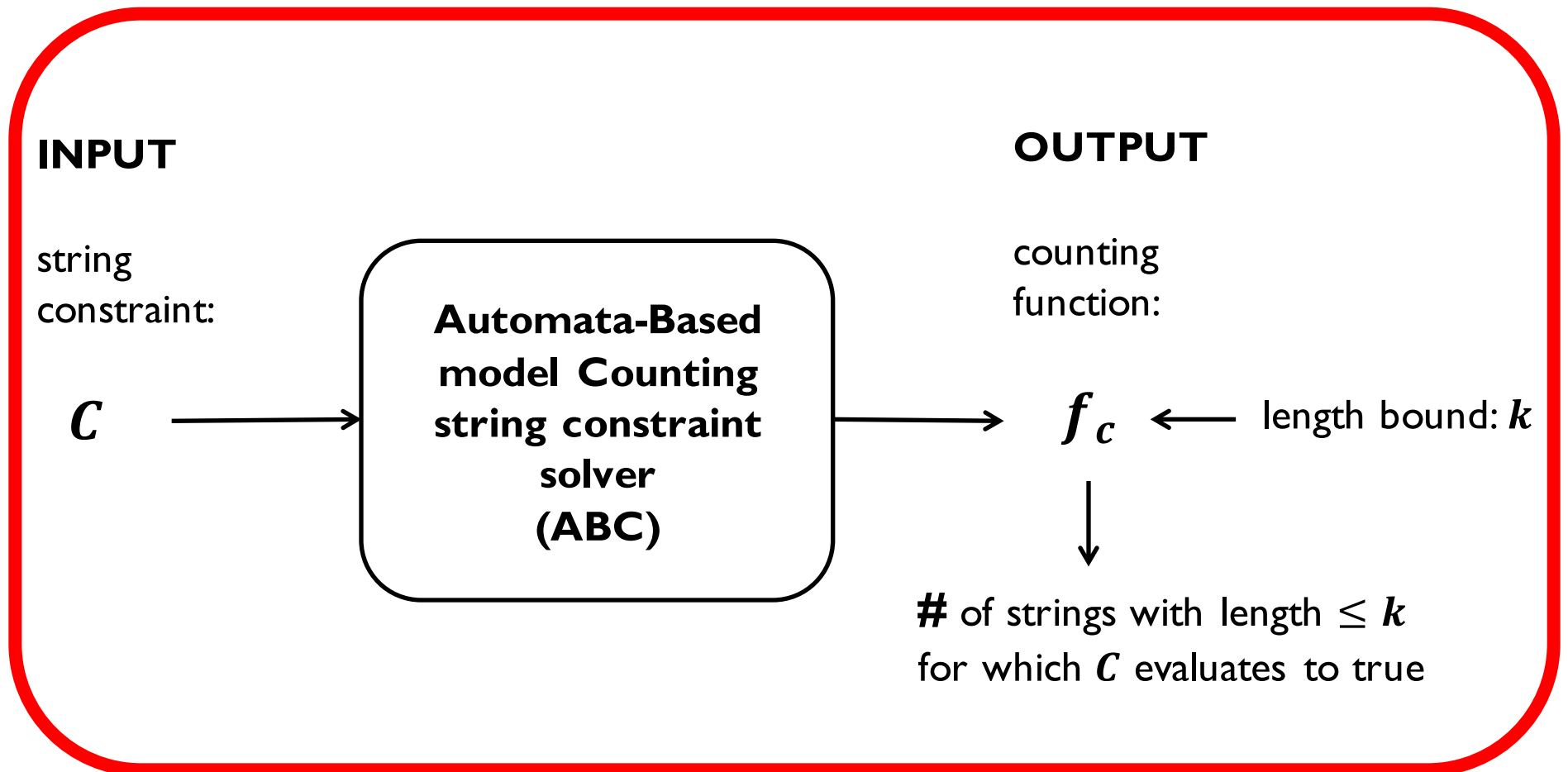
- ▶ Multi-track Binary Integer Automaton:



- ▶ String Automaton:



Model Counting String Constraints Solver



JPF and SPF

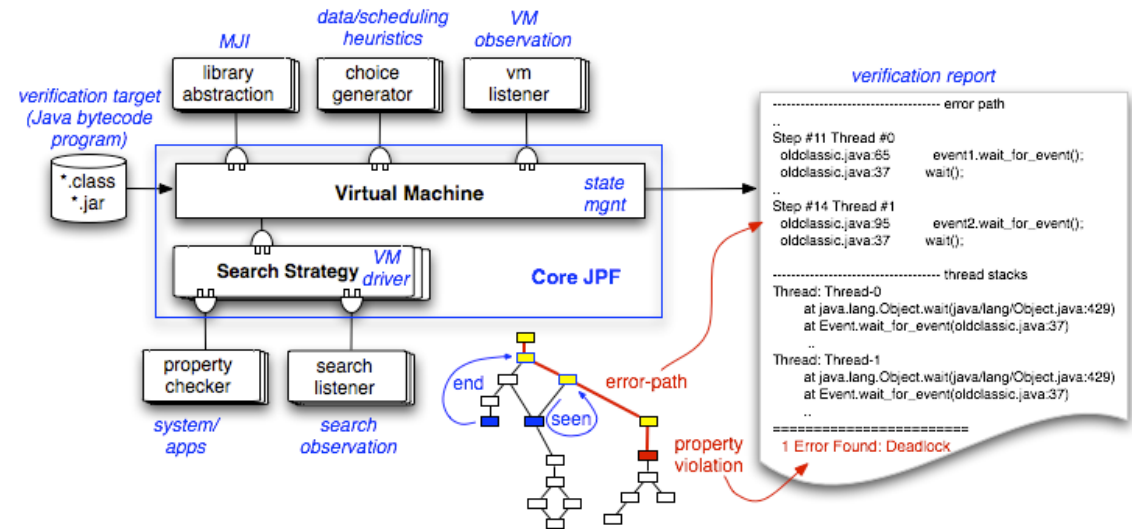
Java PathFinder

Extensible tool for Java bytecode verification

Uses specialized JVM

Developed at NASA Ames since 1999

Open-sourced



Symbolic PathFinder (SPF)

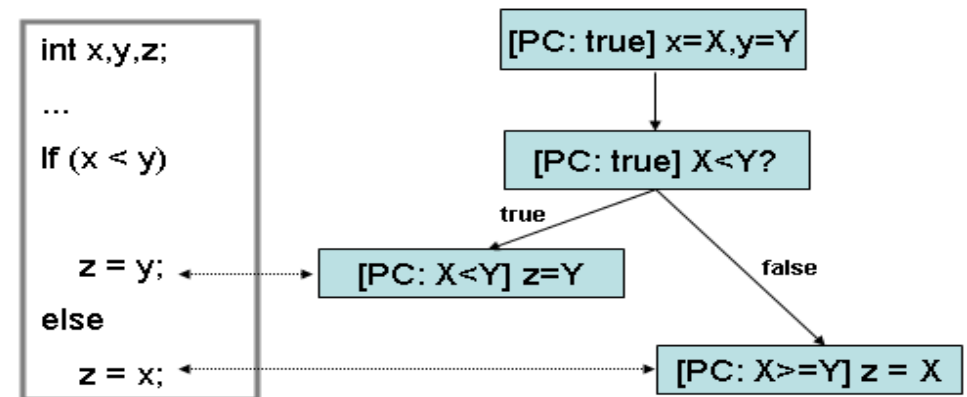
Symbolic execution tool for Java bytecode; open-sourced

Uses lazy initializations to handle complex data structures and arrays as inputs

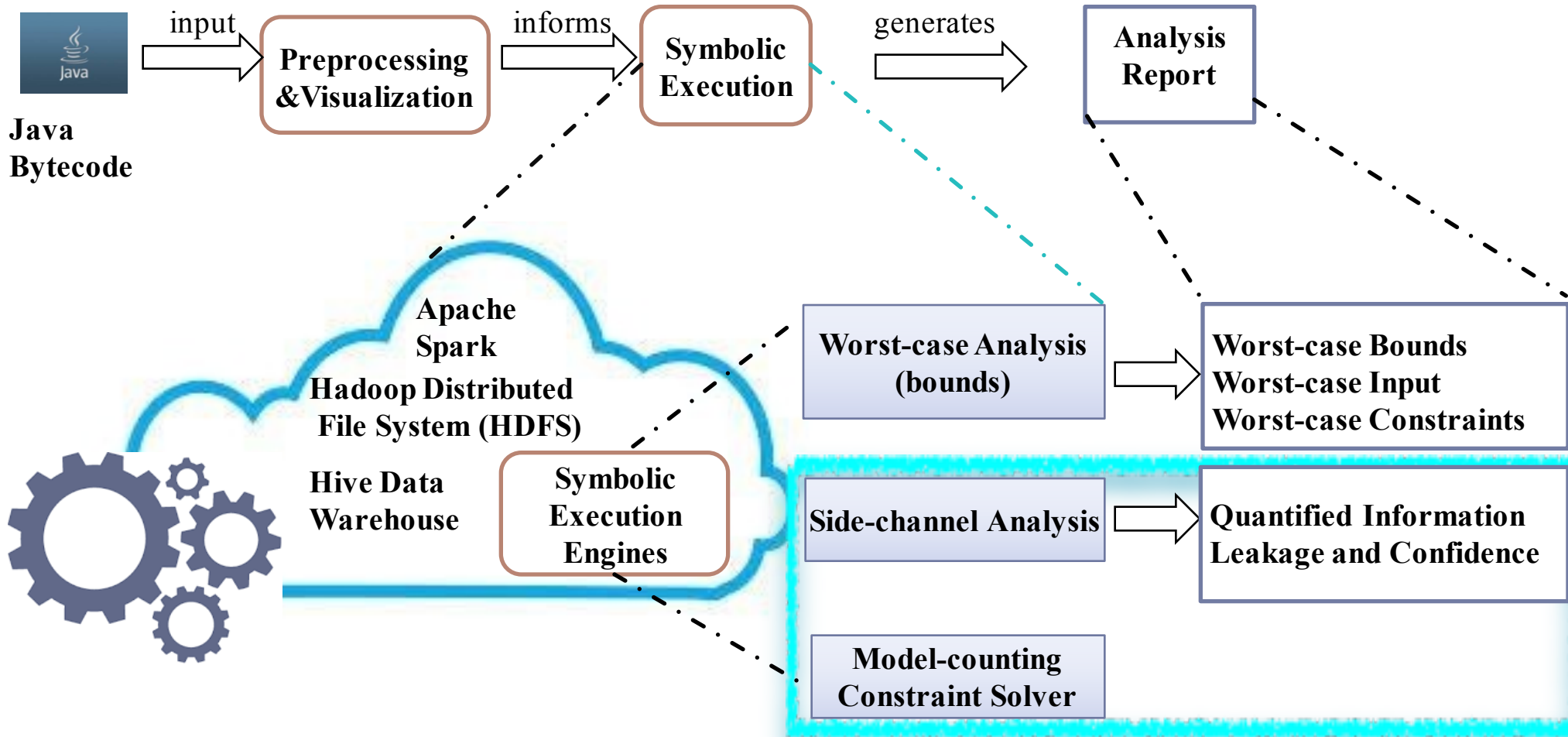
Handles multi-threading

Provides support for symbolic string operations

Supports quantitative reasoning



Side-Channel Analysis



Side Channel Analysis Process

- Use **code inspection** to figure out parts of the code that relate to the mentioned operations
- Write a **driver** to execute the identified code
- Run **symbolic execution** on the resulting system using the **time or memory listener**
 - **Remove or stub** out code that breaks symbolic execution (such as native libraries)
- **Path constraints** generated by symbolic execution identify the relationship between the secret and the observable
- **Devise an attack** based on the result of symbolic execution



Probabilistic Analysis and Entropy Calculation

- ▶ In order to quantify the amount of leakage, compute the **probability** of each observable value
- ▶ To compute observable probabilities:
- ▶ **Count** the number of input values that satisfy a path constraint and divide it by the size of the input domain.
- ▶ This results in the **probability of execution** for that path constraint
- ▶ Using path constraint probabilities compute the **observable probabilities**
- ▶ From the probabilities, compute the **entropy** reduction for each operation



A case study

- Database contains restricted & unrestricted employee information
- Supports SEARCH & INSERT queries
- Question: Is there a side channel in time that a third party can determine the value of a single Restricted ID in the database



Code Inspection

- Using code inspection we identified that the SEARCH and INSERT operations are implemented in:

```
class UDPServerHandler
```

```
method channelRead0
```

```
switch case 1: INSERT
```

```
switch case 8: SEARCH
```



SPF Driver

```
public class Driver {
    public static void main(String[] args) {
        BTree tree = new BTree(10);
        CheckRestrictedID checker = new CheckRestrictedID();
        // create two concrete unrestricted ids
        int id1 = 64, id2 = 85;
        tree.add(id1, null, false);
        tree.add(id2, null, false);
        // create one symbolic restricted id
        int h = Debug.makeSymbolicInteger("h");
        Debug.assume(h!=id1 && h!=id2);
        tree.add(h, null, false);
        checker.add(h);
        UDPServerHandler handler = new UDPServerHandler(tree, checker);
        int key = Debug.makeSymbolicInteger("key");
        handler.channelRead0(8, key); // send a search query with
        // with search range 50 to 100
    }
}
```



SPF Output

>>>>> There are 5 path conditions and 5 observables

cost: 9059

(assert (<= h 100))

(assert (> h 85))

(assert (> h 64))

(assert (not (= h 85)))

(assert (not (= h 64)))

Count = 15

cost: 8713

(assert (<= h 85))

(assert (> h 64))

(assert (not (= h 85)))

(assert (not (= h 64)))

Count = 20

cost: 7916

(assert (> h 100))

(assert (> h 85))

(assert (> h 64))

(assert (not (= h 85)))

(assert (not (= h 64)))

Count = 923

cost: 8701

(assert (>= h 50))

(assert (<= h 64))

(assert (not (= h 85)))

(assert (not (= h 64)))

Count = 14

cost: 7951

(assert (< h 50))

(assert (<= h 64))

(assert (not (= h 85)))

(assert (not (= h 64)))

Count = 50

PC equivalence class model counting results.

Cost: 9059 Count: 15 Probability: 0.014677

Cost: 8713 Count: 20 Probability: 0.019569

Cost: 7916 Count: 923 Probability: 0.903131

Cost: 8701 Count: 14 Probability: 0.013699

Cost: 7951 Count: 50 Probability: 0.048924

Domain Size: 1022

Single Run Leakage: 0.6309758112933285



Observation & Proposed Attack

- ▶ SEARCH operation:

takes longer when the secret is within the search range
(9059, 8713, 8701 byte code instructions)

as opposed to the case when the secret is out of the search range (7916, 7951 byte code instructions)

- ▶ Proposed attack:

Measure the time it takes for the search operation to figure out if there is a secret within the search range.



Attack

- Binary search on the ranges of the IDs
- Send two search queries at a time and compare their execution time.
- Refine the search range based on the result.

```
min= 0; max=MAX_ID      //assume MAX_ID is a power of 2
while ( min < max )
{
  half = (max-min-1)/2;
  if (time(search(min.. min+half-1) > time(search(min+half .. max)))
    max = min+half-1;
  else
    min = min+half;
}
```

Attack Output

Running [0,40000000] at 0.
Comparing 467821 vs 612252...
Running [20000000,40000000] at 2.
Comparing 400377 vs 333665...
Running [20000000,30000000] at 4.
Comparing 200603 vs 237025...
Running [25000000,30000000] at 6.
Comparing 163564 vs 115072...
Running [25000000,27500000] at 8.
Comparing 95736 vs 37388...
Running [25000000,26250000] at 10.
Comparing 85305 vs 30118...
Running [25000000,25625000] at 12.
Comparing 22765 vs 72958...
Running [25312500,25625000] at 14.
Comparing 2147483647 vs 19353...
Running [25312500,25468750] at 16.
Comparing 517 vs 2147483647...
Running [25390625,25468750] at 18.
Comparing 317 vs 2147483647...
Running [25429687,25468750] at 20.
Comparing 2147483647 vs 302...
Running [25429687,25449218] at 22.
Comparing 2147483647 vs 287...
Running [25429687,25439452] at 24.
Comparing 336 vs 2147483647...

Running [25434569,25439452] at 26.
Comparing 300 vs 2147483647...
Running [25437010,25439452] at 28.
Comparing 2147483647 vs 265...
Running [25437010,25438231] at 30.
Comparing 2147483647 vs 328...
Running [25437010,25437620] at 32.
Comparing 280 vs 2147483647...
Running [25437315,25437620] at 34.
Comparing 293 vs 2147483647...
Running [25437467,25437620] at 36.
Comparing 2147483647 vs 281...
Running [25437467,25437543] at 38.
Comparing 2147483647 vs 613...
Running [25437467,25437505] at 40.
Comparing 2147483647 vs 258...
Running [25437467,25437486] at 42.
Comparing 2147483647 vs 291...
Running [25437467,25437476] at 44.
Comparing 362 vs 2147483647...
Running [25437471,25437476] at 46.
Comparing 311 vs 2147483647...
Running [25437473,25437476] at 48.
Comparing 2147483647 vs 2147483647...
Checking oracle for:25437474...true
Checking oracle for:25437475...false

Multi-Run Analysis

- The side channel analysis I discussed so far is for analyzing a single execution of a program
- Can we do model multi-run analysis?
- Adversary runs the program on multiple inputs one after another
- Can we determine the amount of information leakage in such a scenario?



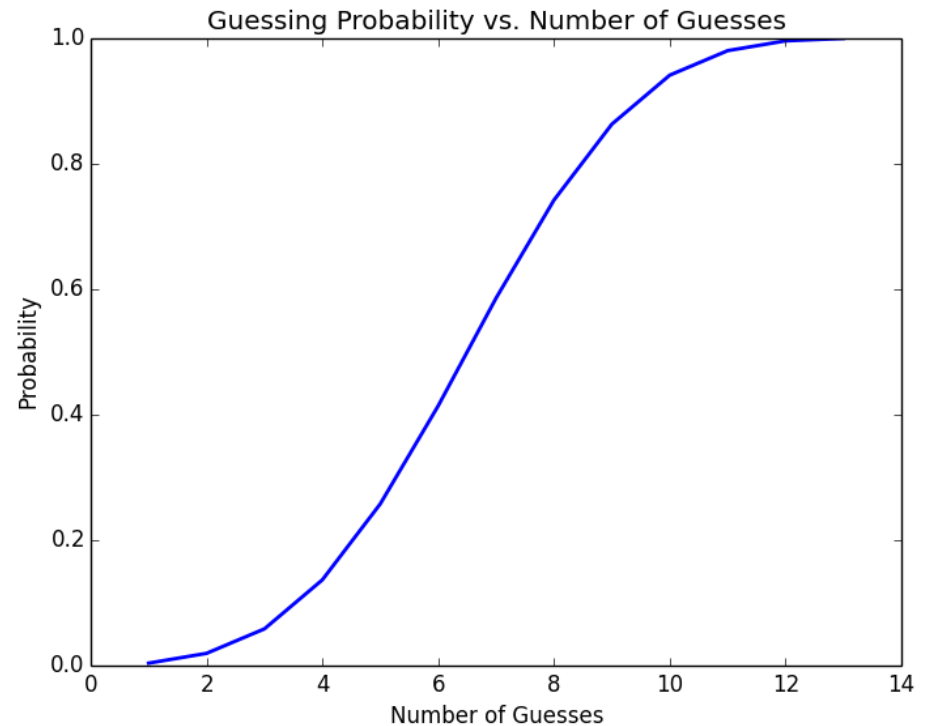
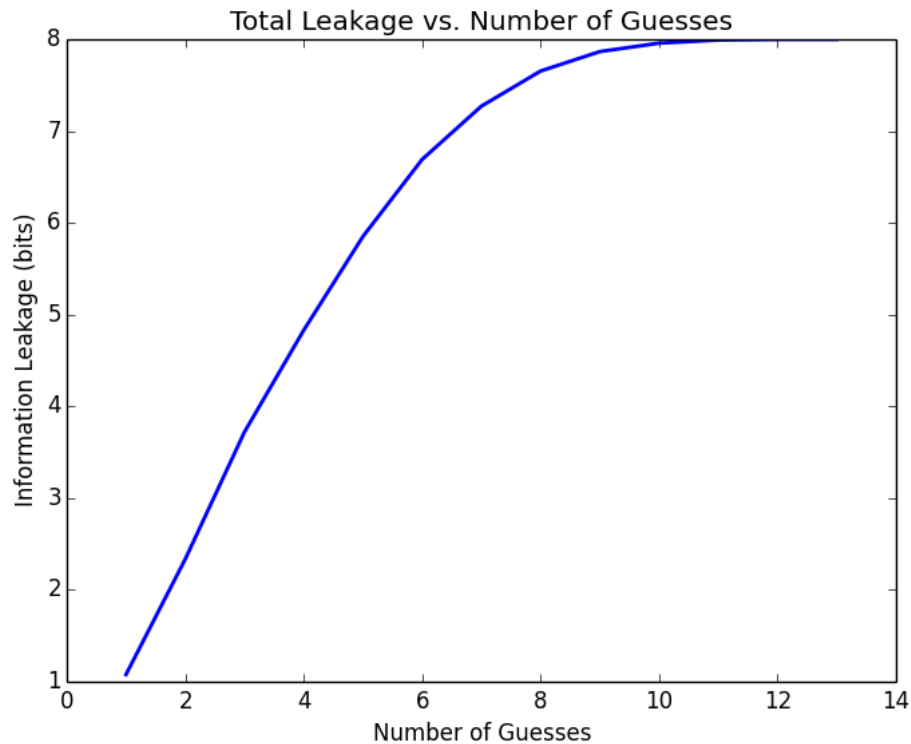
Multi-Run Analysis

- For multi-run analysis we need an adversary model
 - Adversary behavior influences the analysis
- It would make sense to calculate the leakage for the best adversary
- For a class of side channels called “segmented oracles” we can use symbolic execution and entropy calculation from a single run to compute the change in the entropy for multiple runs
- This can be used to automatically compute how many tries it will take to reveal the secret.



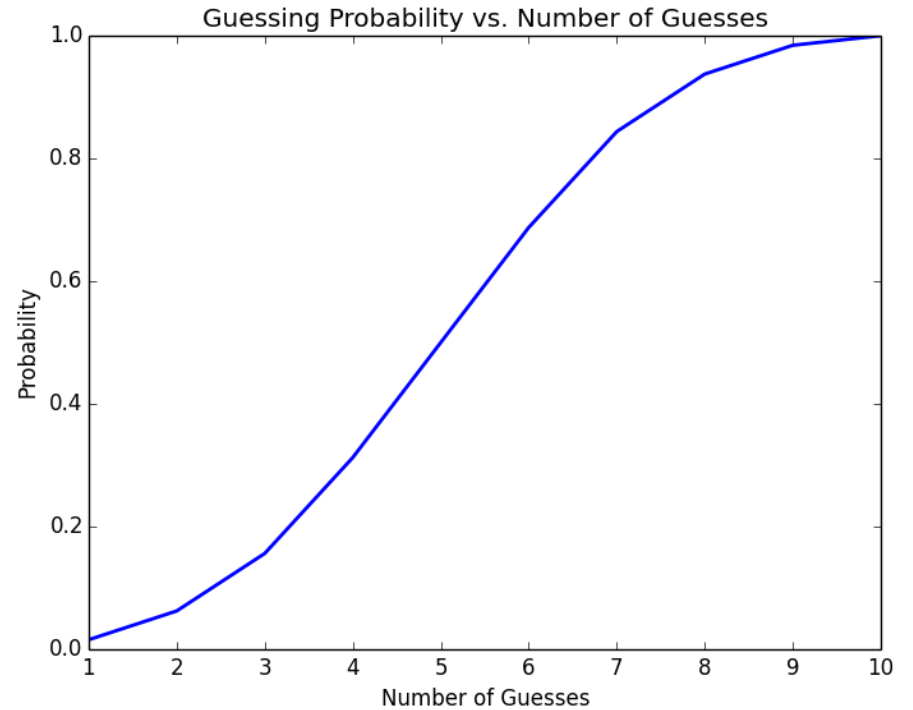
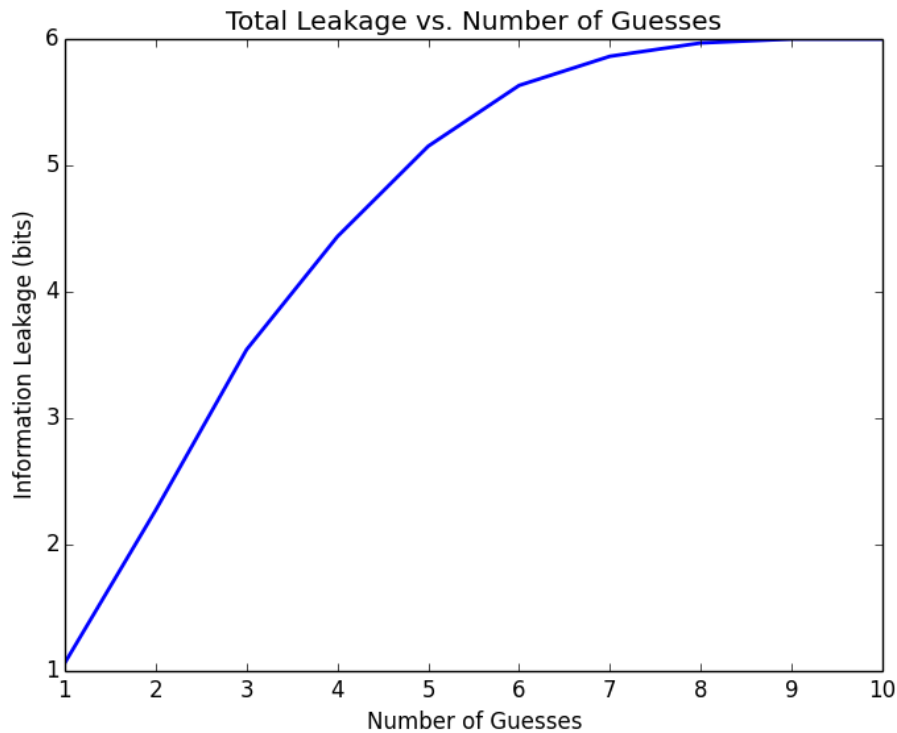
Results for Password Check

Results for 4 segments with 4 values (8 bits of information)



Results for CRIME

Results for 3 segments with 4 values (6 bits of information)

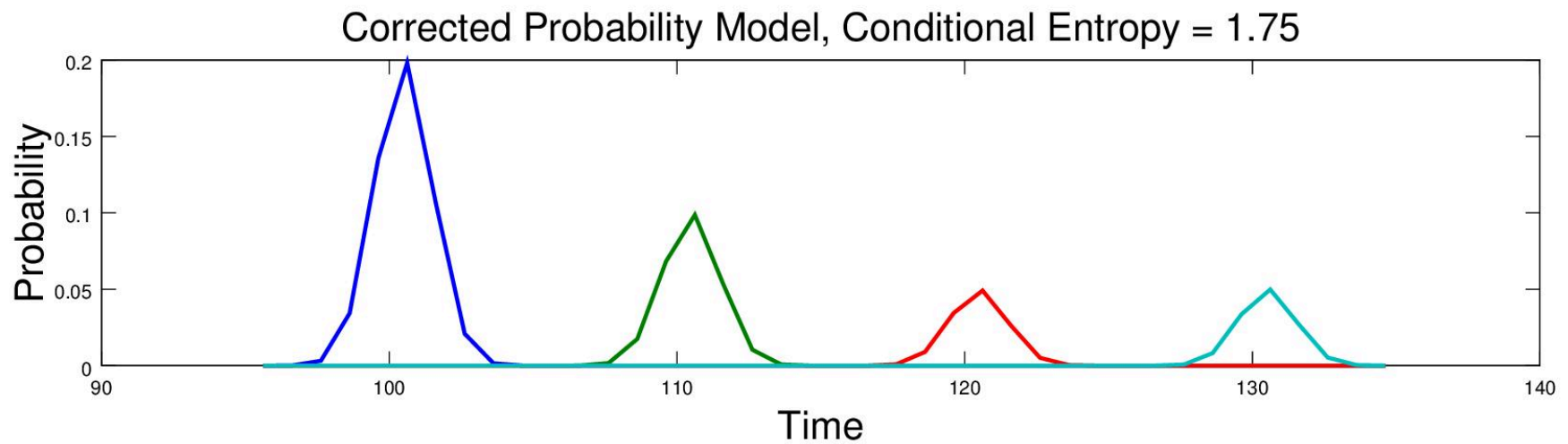
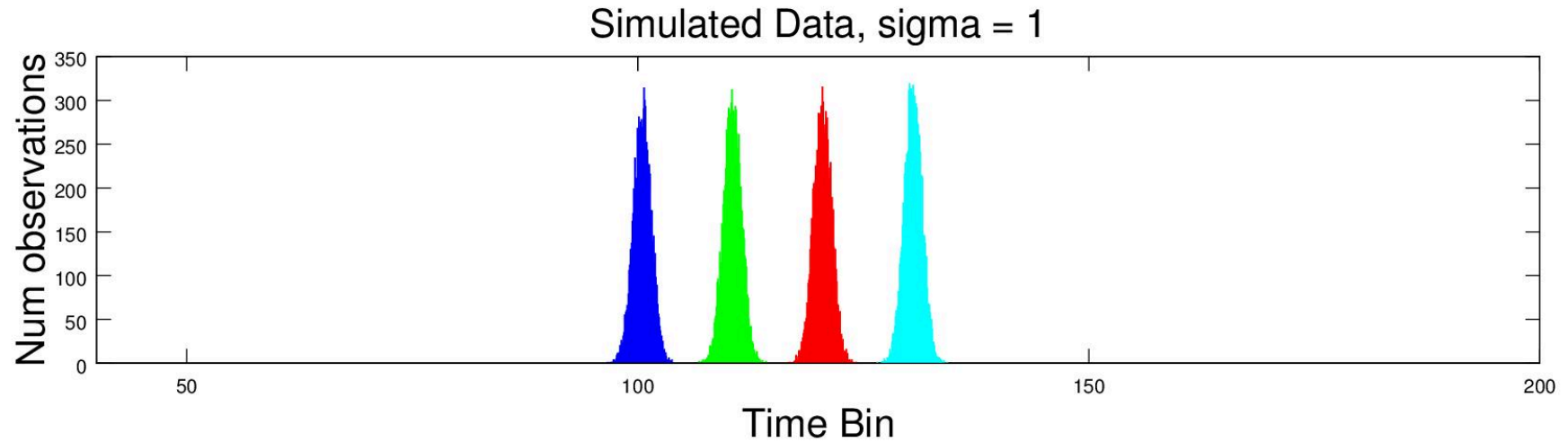


Noisy Observations

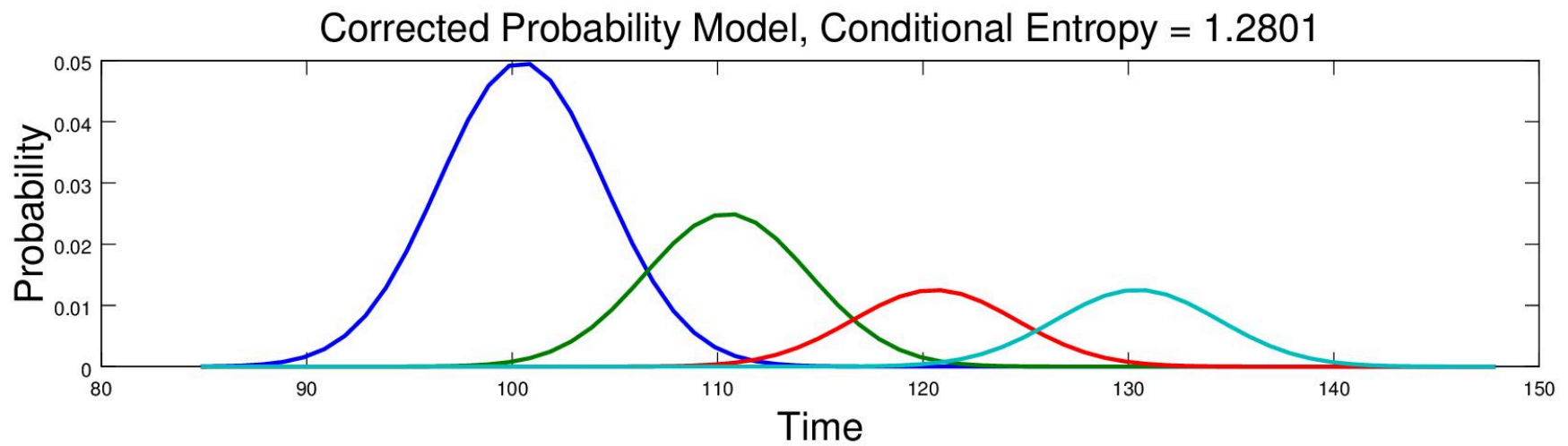
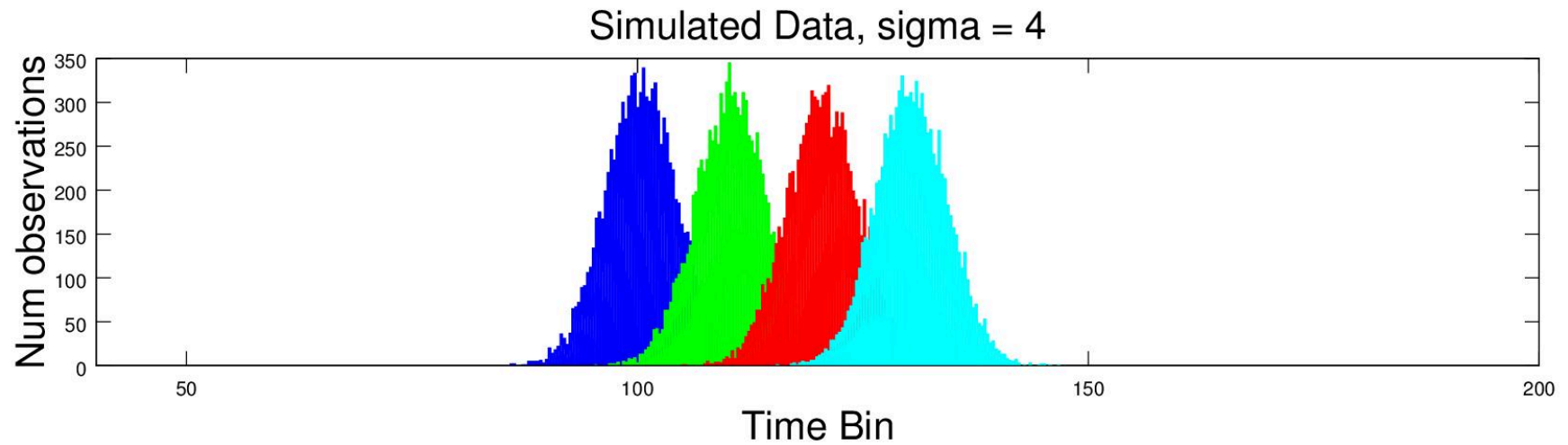
- ▶ Entropy computations we have shown so far do not take observation noise into account
- ▶ One approach we are investigating to handle noise:
 - Assume a noise distribution (for example normal distribution)
 - Run fuzzing to observe parameters of the distribution (mean and standard deviation)
 - Update entropy calculations using the noise model



Noisy Observation Simulation

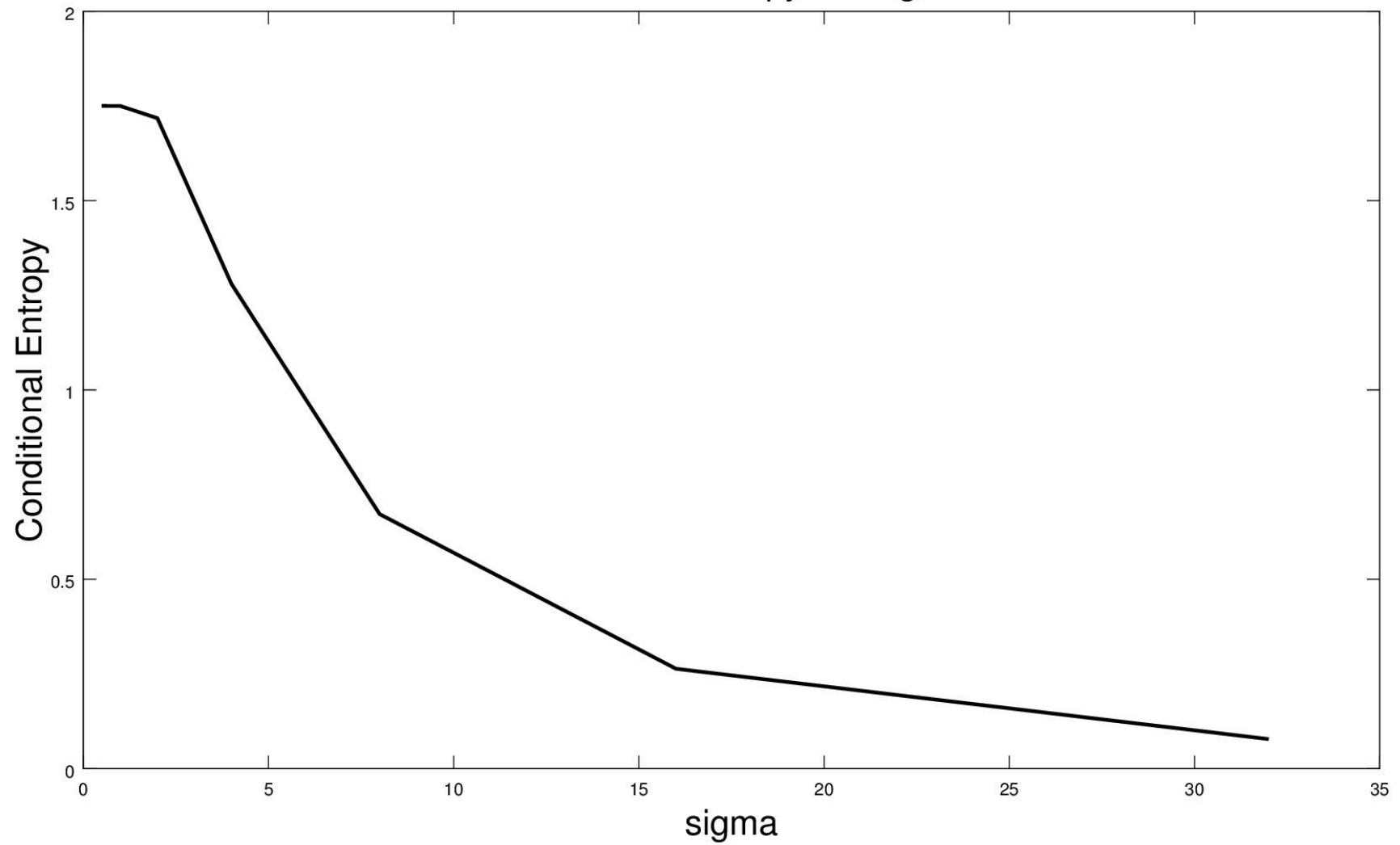


Noisy Observation Simulation

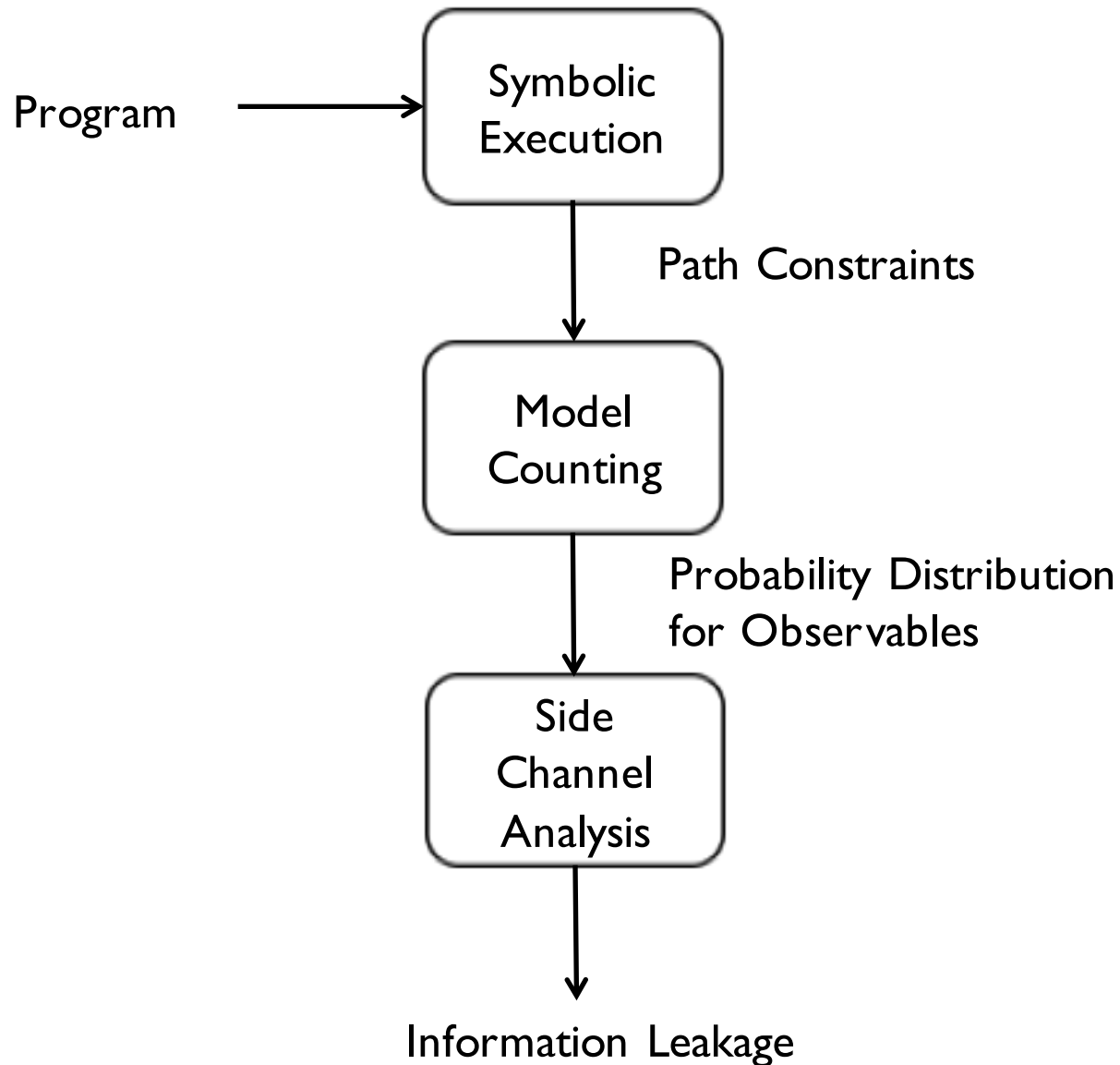


Entropy vs. Noise

Conditional Entropy vs. Sigma



Summary



Related work: Quantitative Information Flow

- ▶ Geoffrey Smith. [On the Foundations of Quantitative Information Flow](#). FOSSACS 2009:288-302
- ▶ Pasquale Malacaria. [Assessing security threats of looping constructs](#). POPL 2007:225-235
- ▶ David Clark, Sebastian Hunt, Pasquale Malacaria. [A static analysis for quantifying information flow in a simple imperative language](#). Journal of Computer Security 15(3): 321-371 (2007)
- ▶ Jonathan Heusser, Pasquale Malacaria. [Quantifying information leaks in software](#). ACSAC 2010: 261-269
- ▶ Quoc-Sang Phan, Pasquale Malacaria, Oksana Tkachuk, Corina S. Pasareanu. [Symbolic quantitative information flow](#). ACM SIGSOFT Software Engineering Notes 37(6): 1-5 (2012)
- ▶ Quoc-Sang Phan, Pasquale Malacaria, Corina S. Pasareanu, Marcelo d'Amorim. [Quantifying information leaks using reliability analysis](#). SPIN 2014:105-108
- ▶ Stephen McCamant, Michael D. Ernst. [Quantitative information flow as network flow capacity](#). PLDI 2008:193-205
- ▶ Michael Backes, Boris Köpf, Andrey Rybalchenko. [Automatic Discovery and Quantification of Information Leaks](#). IEEE Symposium on Security and Privacy 2009:141-153
- ▶ Shuo Chen, Rui Wang, XiaoFeng Wang, Kehuan Zhang. [Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow](#). IEEE Symposium on Security and Privacy 2010:191-206
- ▶ Goran Doychev, Dominik Feld, Boris Köpf, Laurent Mauborgne, Jan Reineke. [CacheAudit: A Tool for the Static Analysis of Cache Side Channels](#). USENIX Security 2013:431-446

Related work: Model Counting

- ▶ SMC
- ▶ ACM
- ▶ Latte
- ▶ Barvinok

