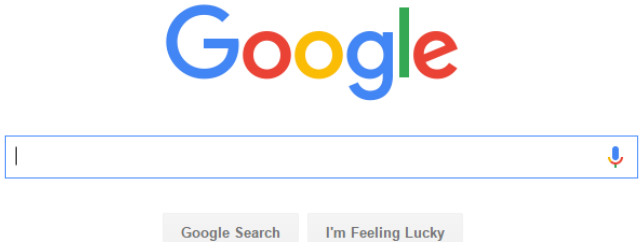
PLDI 2016 Tutorial
Automata-Based String Analysis

Tevfik Bultan, Abdulbaki Aydin, Lucas Bang
Verification Laboratory
http://vlab.cs.ucsb.edu
Department of Computer Science

Common Usages of Strings

- Input validation and sanitization



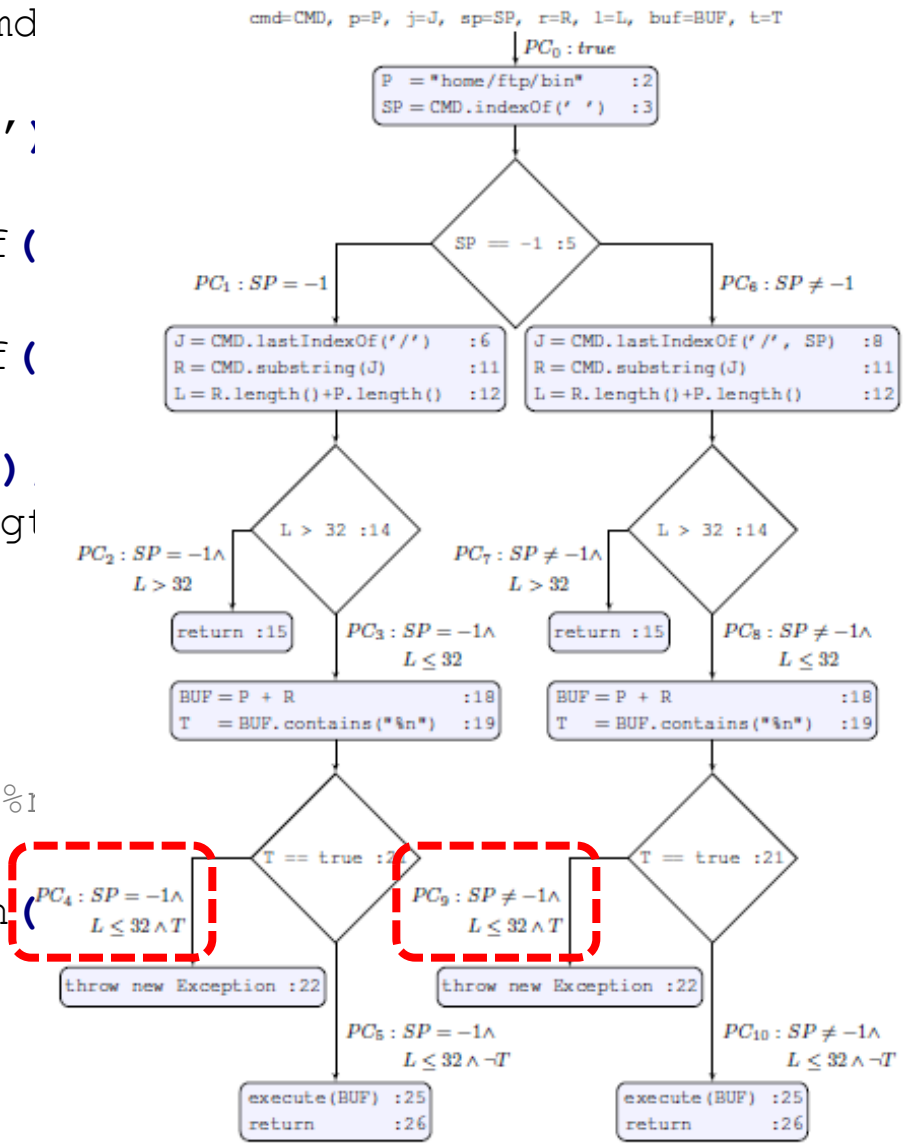
- Database query generation

- Formatted data generation

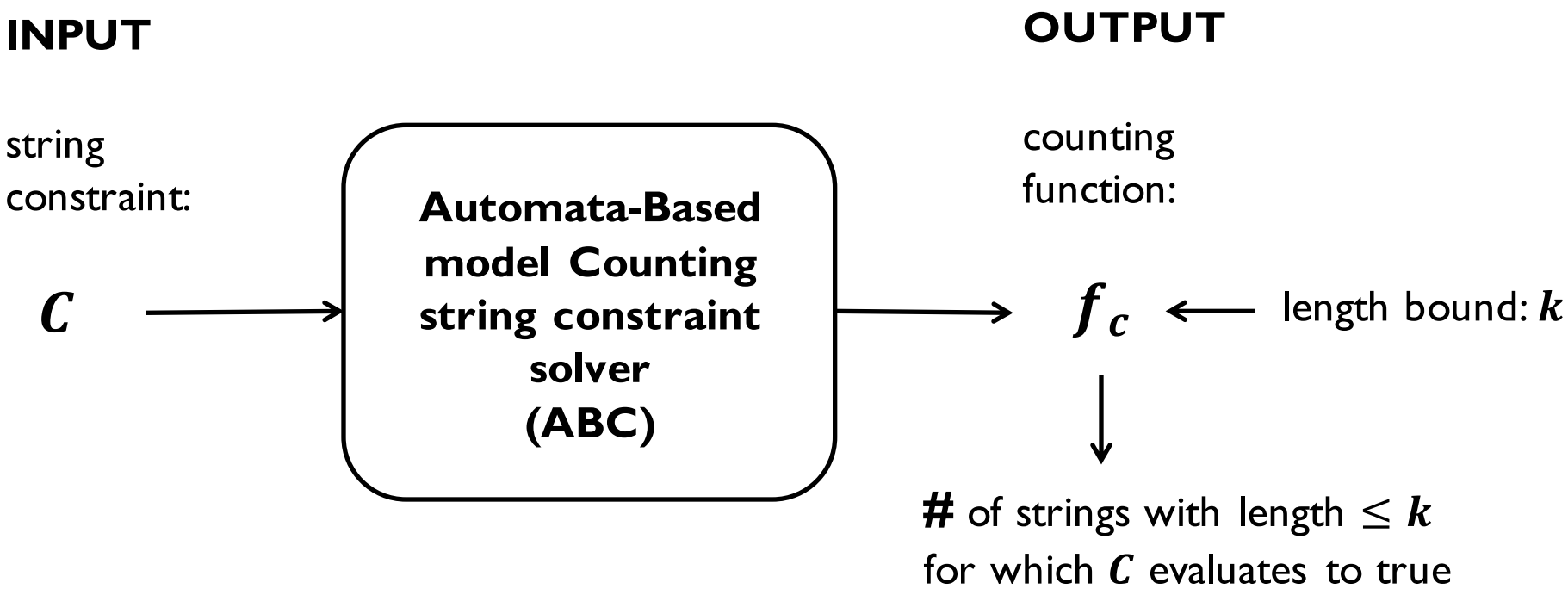
- Dynamic code generation

Symbolic Execution Example

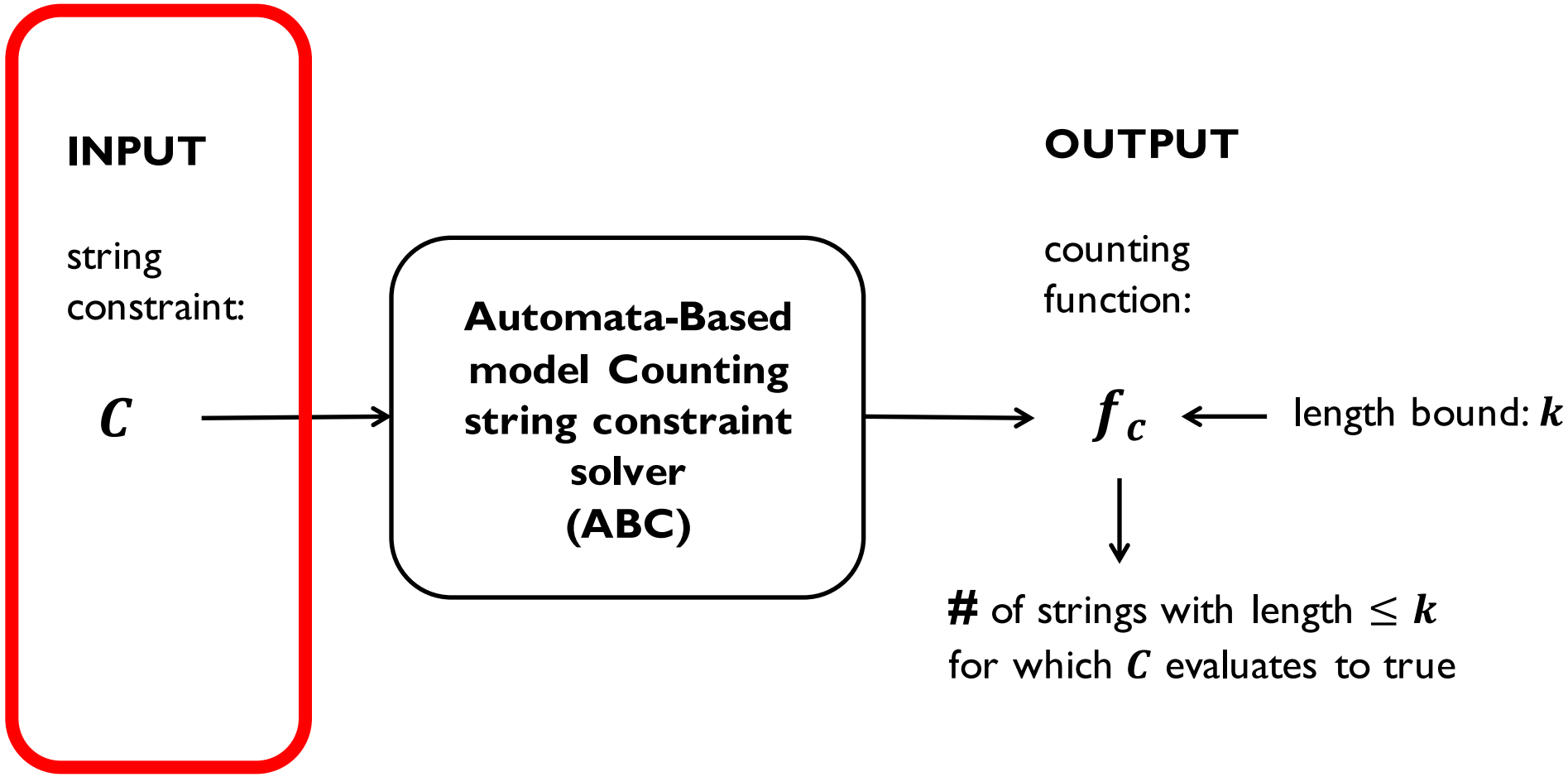
```
1  public void site_exec(String cmd
2        String p = "home/ftp/bin";
3        int j, sp = cmd.indexOf(' ');
5        if (sp == -1) {
6              j = cmd.lastIndexOf(
7        } else {
8              j = cmd.lastIndexOf(
9        }
11       String r = cmd.substring(j);
12       int l = r.length() + p.lengt
14       if (l > 32) {
15             return;
16       }
18       String buf = p + r;
19       boolean t = buf.contains("%
21       if (t == true) {
22             throw new Exception(
23       }
25       execute(buf);
26       return;
27 }
```



Model Counting String Constraint Solver



Model Counting String Constraint Solver



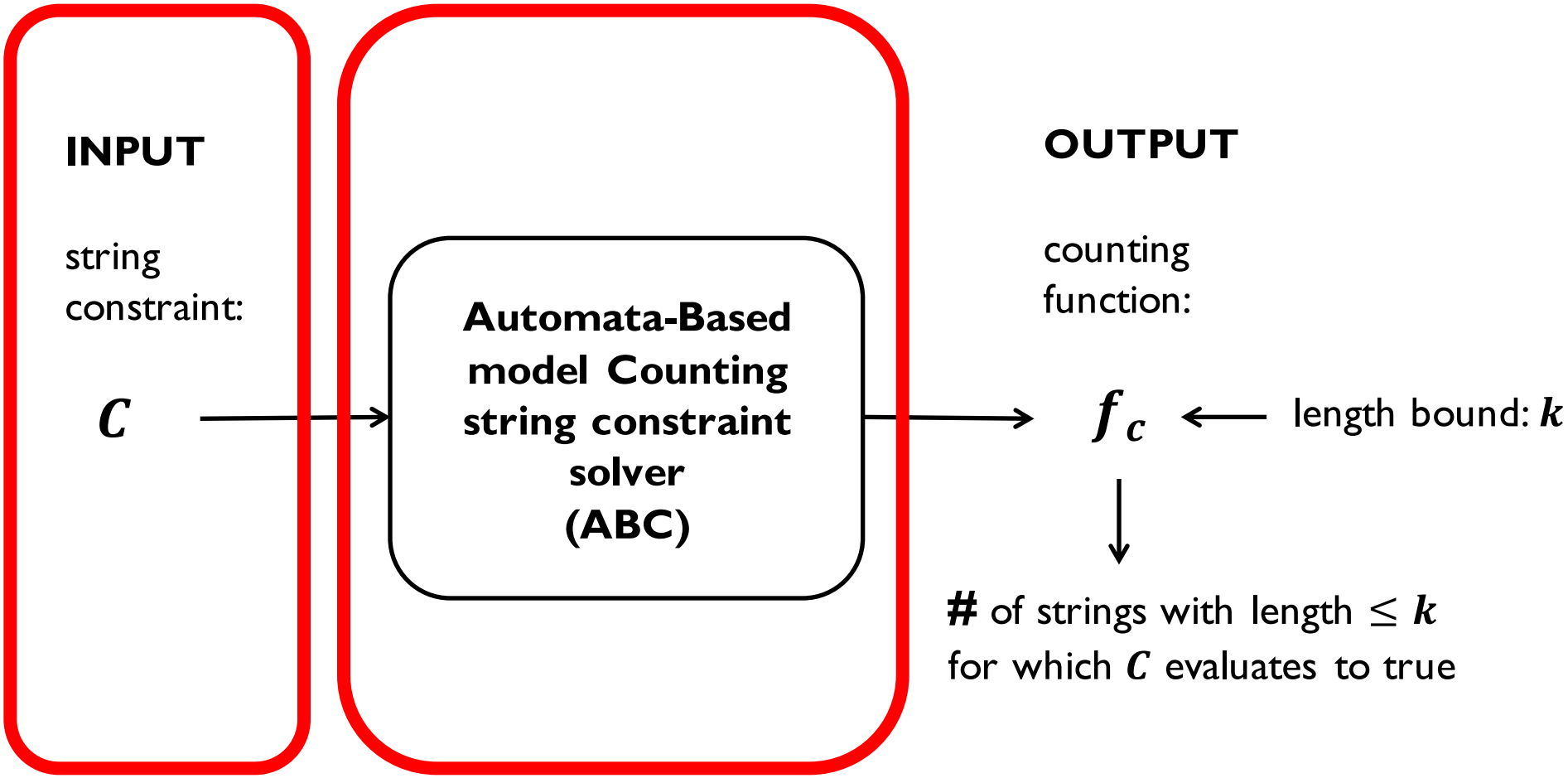
String Constraint Language

$$
\begin{array}{lll}
C & \longrightarrow & bterm \\
\\
bterm & \longrightarrow & \texttt{v} \mid \texttt{true} \mid \texttt{false} \\
 & \mid & \neg bterm \mid bterm \wedge bterm \mid bterm \vee bterm \mid (bterm) \\
 & \mid & sterm = sterm \\
 & \mid & \texttt{match}(sterm, sterm) \\
 & \mid & \texttt{contains}(sterm, sterm) \\
 & \mid & \texttt{begins}(sterm, sterm) \\
 & \mid & \texttt{ends}(sterm, sterm) \\
 & \mid & iterm = iterm \mid iterm < iterm \mid iterm > iterm \\
\\
iterm & \longrightarrow & \texttt{v} \mid \texttt{n} \\
 & \mid & iterm + iterm \mid iterm - iterm \mid iterm \times \texttt{n} \mid (iterm) \\
 & \mid & \texttt{length}(sterm) \mid \texttt{toint}(sterm) \\
 & \mid & \texttt{indexof}(sterm, sterm) \\
 & \mid & \texttt{lastindexof}(sterm, sterm) \\
\\
sterm & \longrightarrow & \texttt{v} \mid \varepsilon \mid \texttt{s} \\
 & \mid & sterm.sterm \mid sterm|sterm \mid sterm^* \mid (sterm) \\
 & \mid & \texttt{charat}(sterm, iterm) \mid \texttt{tostring}(iterm) \\
 & \mid & \texttt{toupper}(sterm) \mid \texttt{tolower}(sterm) \\
 & \mid & \texttt{substring}(sterm, iterm, iterm) \\
 & \mid & \texttt{replacefirst}(sterm, sterm, sterm) \\
 & \mid & \texttt{replacelast}(sterm, sterm, sterm) \\
 & \mid & \texttt{replaceall}(sterm, sterm, sterm)
\end{array}
$$

Example String Expressions

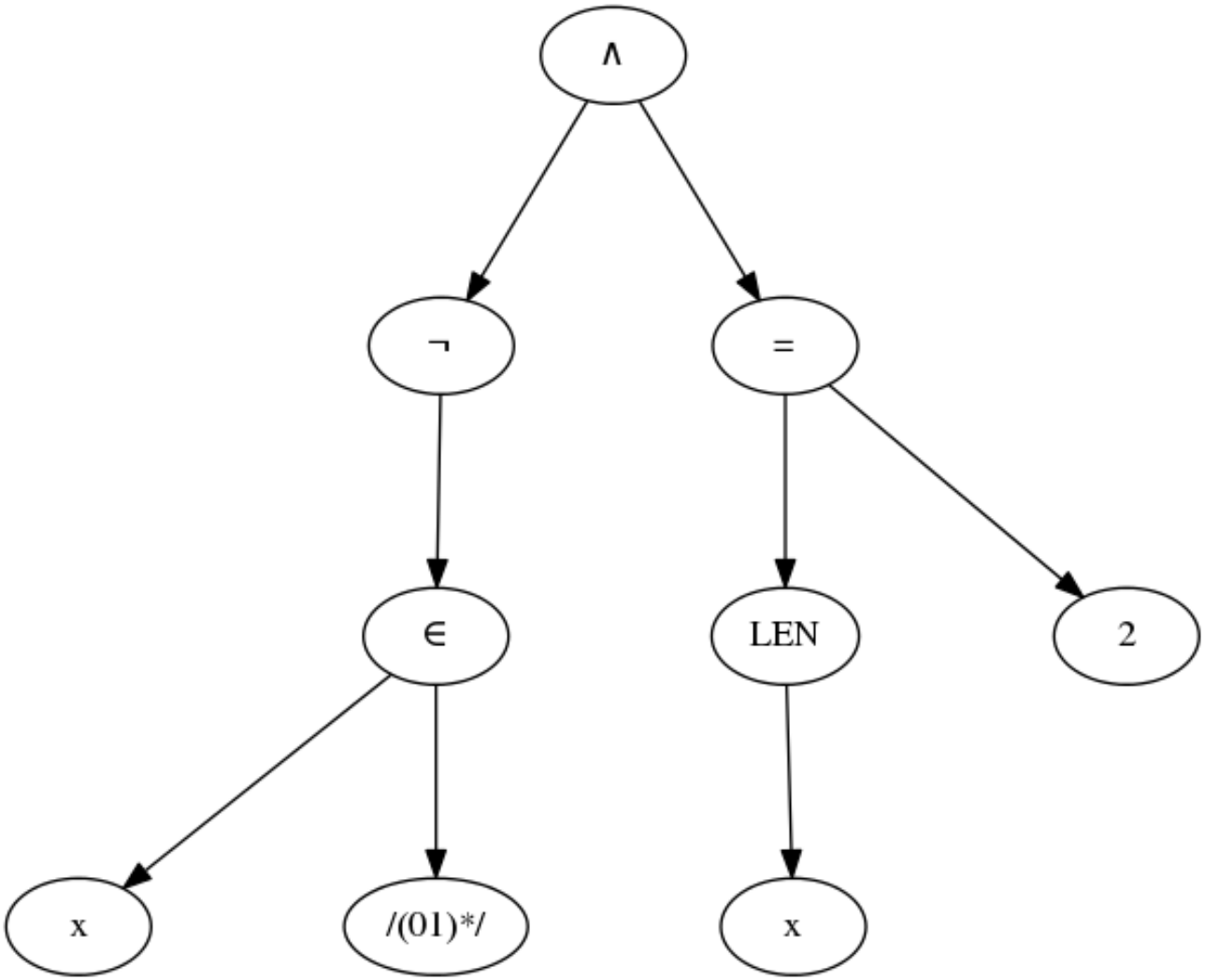
	String Expression	Constraint Language
Java	`s.length()`	`length(s)`
	`s.isEmpty()`	`length(s) == 0`
	`s.startsWith(t,n)`	`0 ≤ n ∧ n ≤ |s| ∧ begins(substring(s,n,|s|),t)`
	`s.indexOf(t,n)`	`indexof(substring(s,n,|s|),t)`
	`s.replaceAll(p,r)`	`replaceall(s,p,r)`
PHP	`strrpos(s, t)`	`lastindexof(s,t)`
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

Model Counting String Constraint Solver



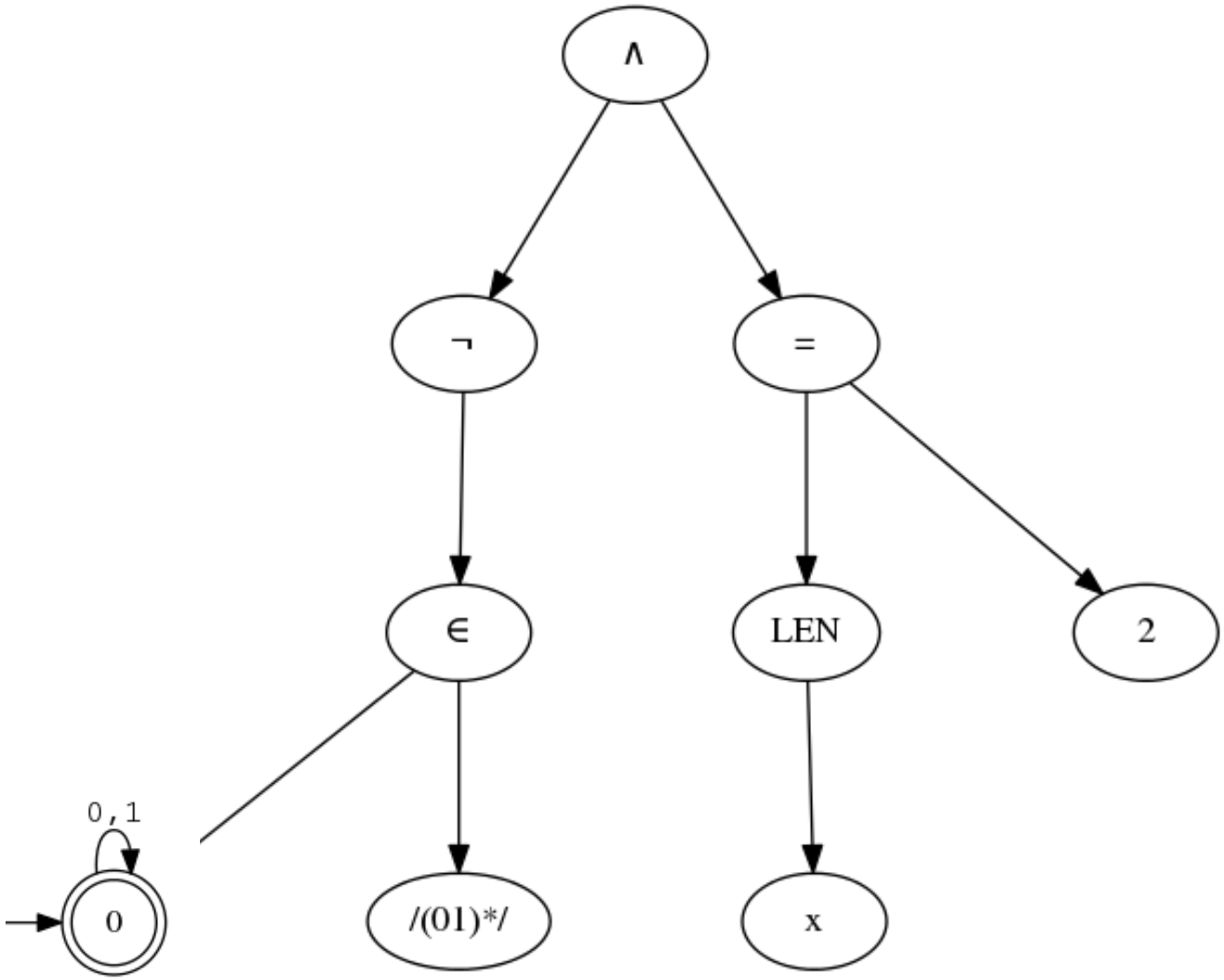
String Automata Construction

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



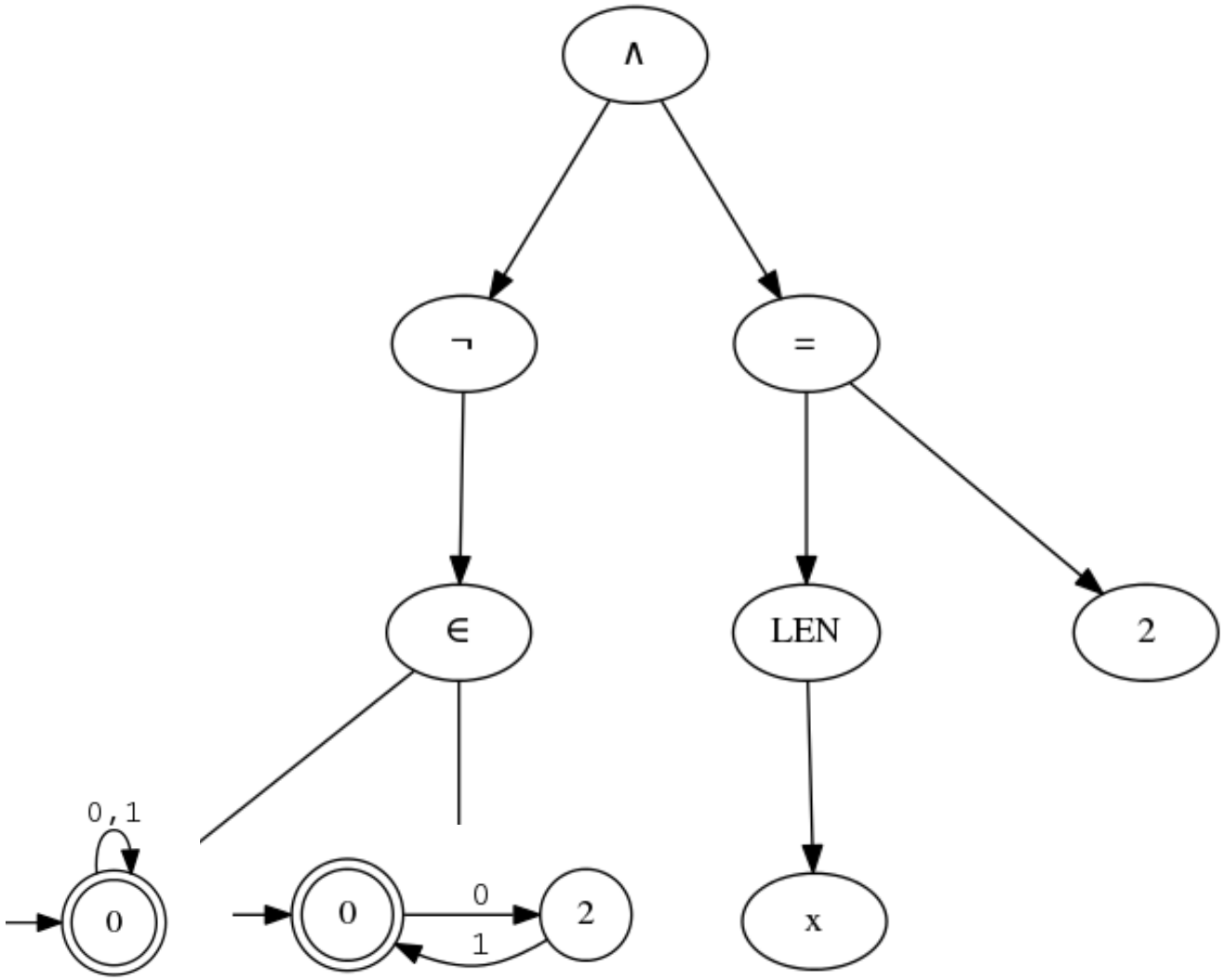
String Automata Construction

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



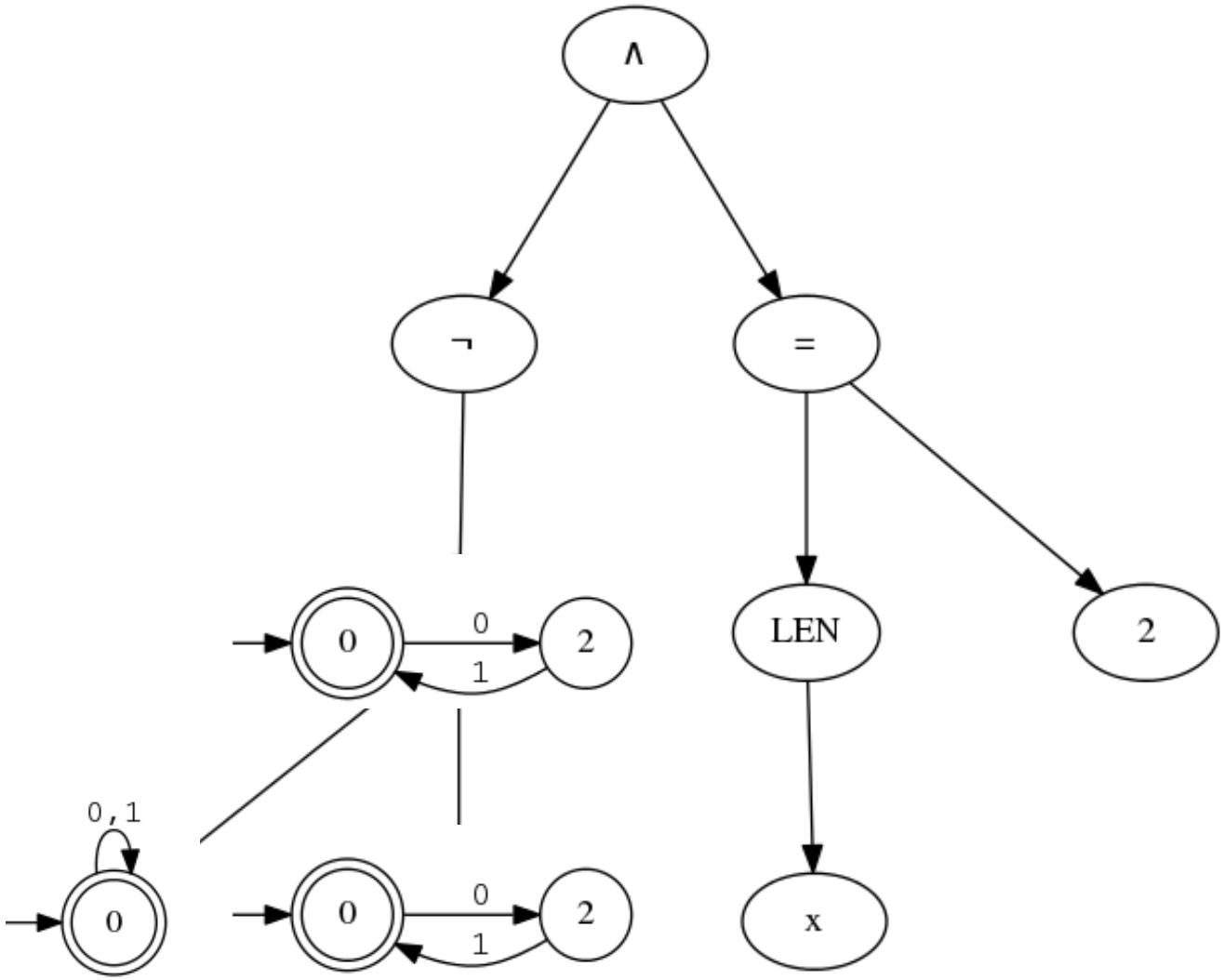
String Automata Construction

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



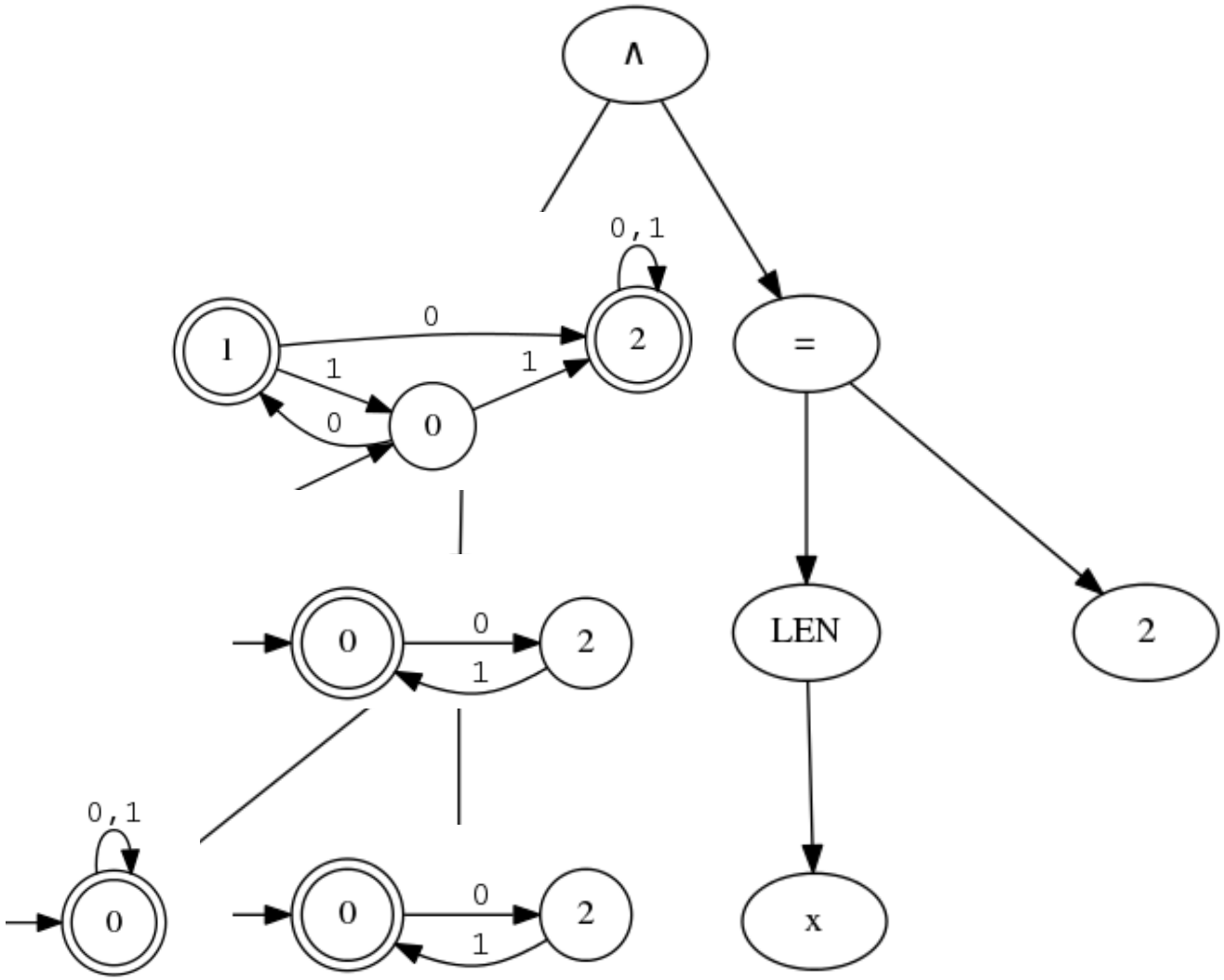
String Automata Construction

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



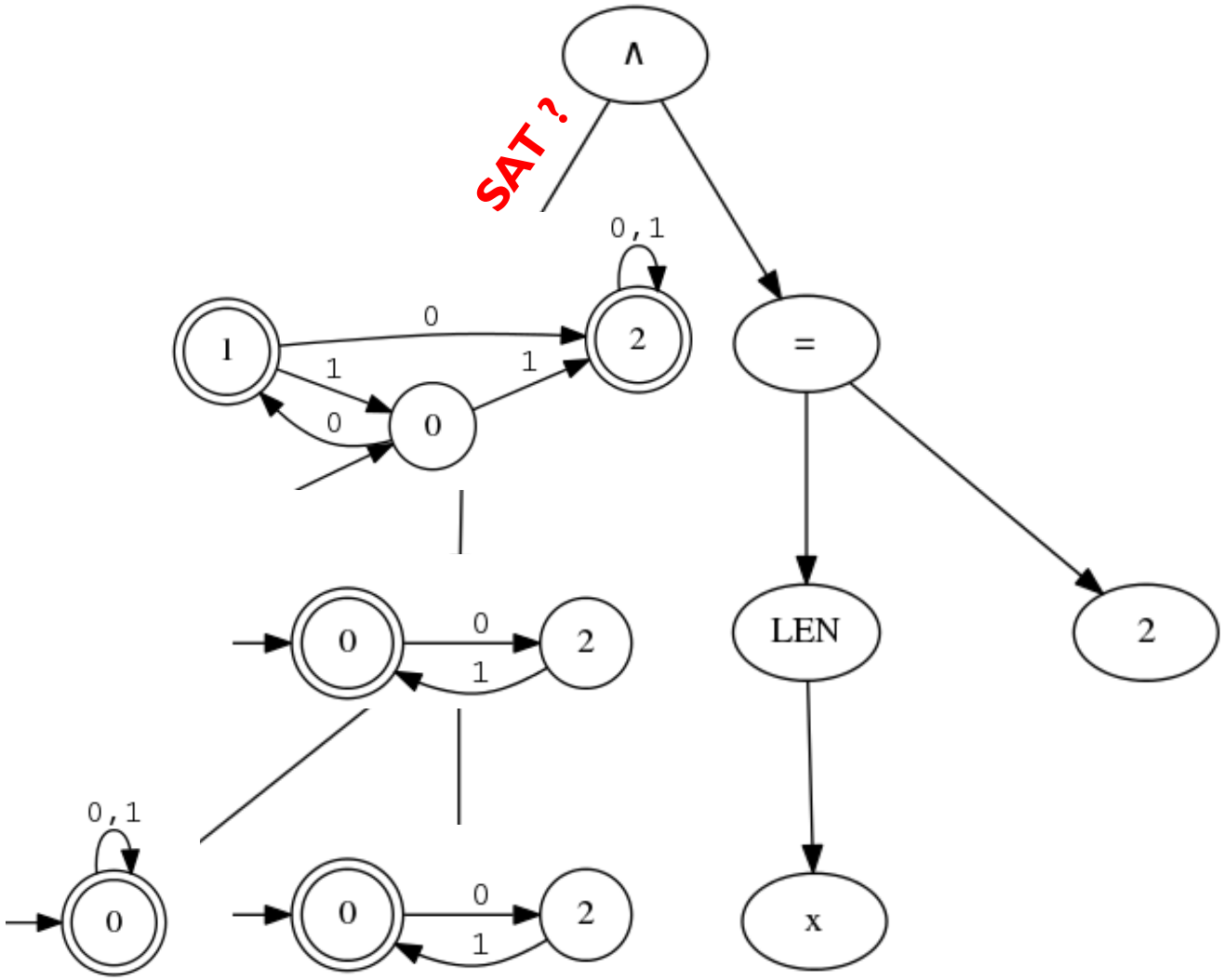
String Automata Construction

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



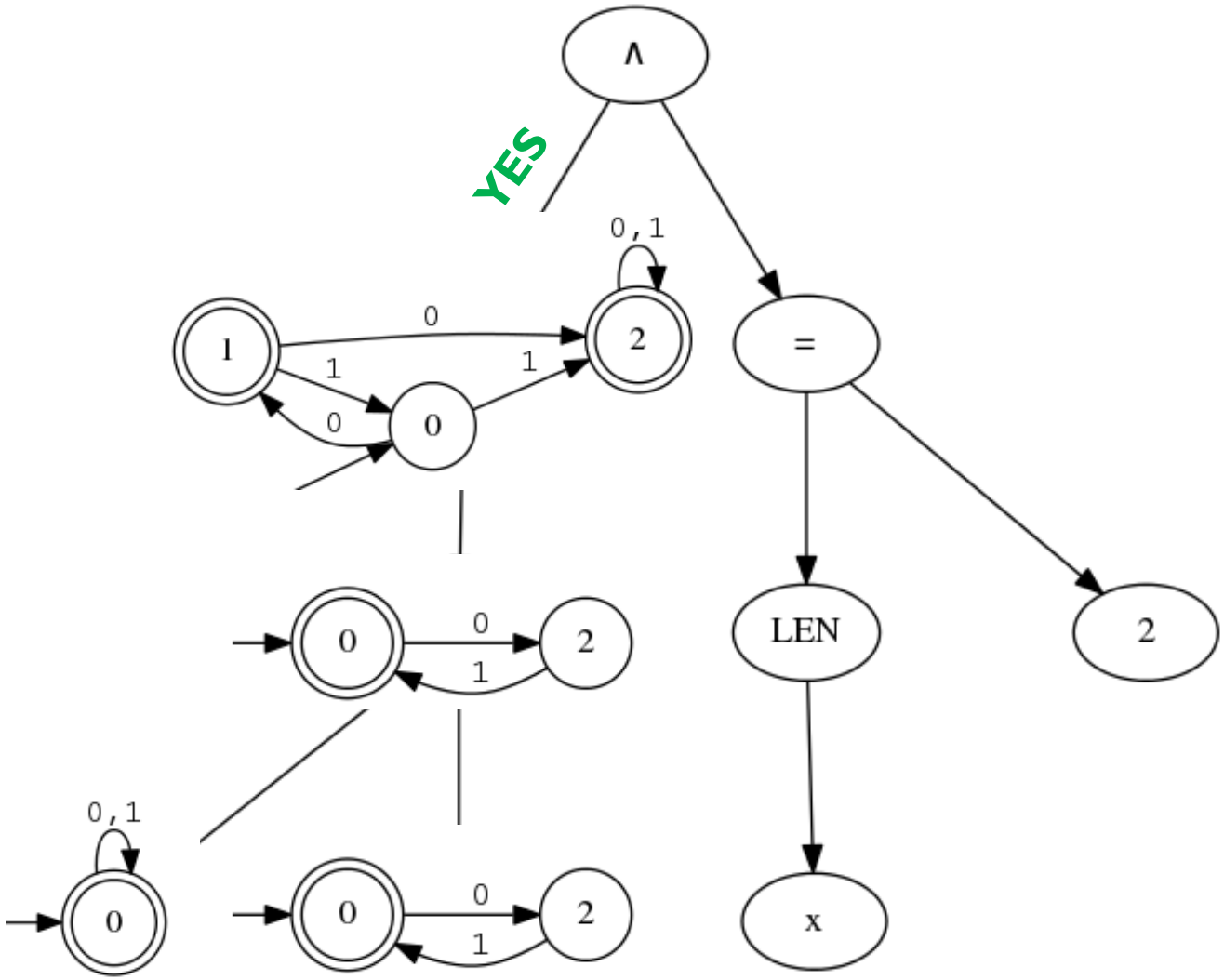
String Automata Construction

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



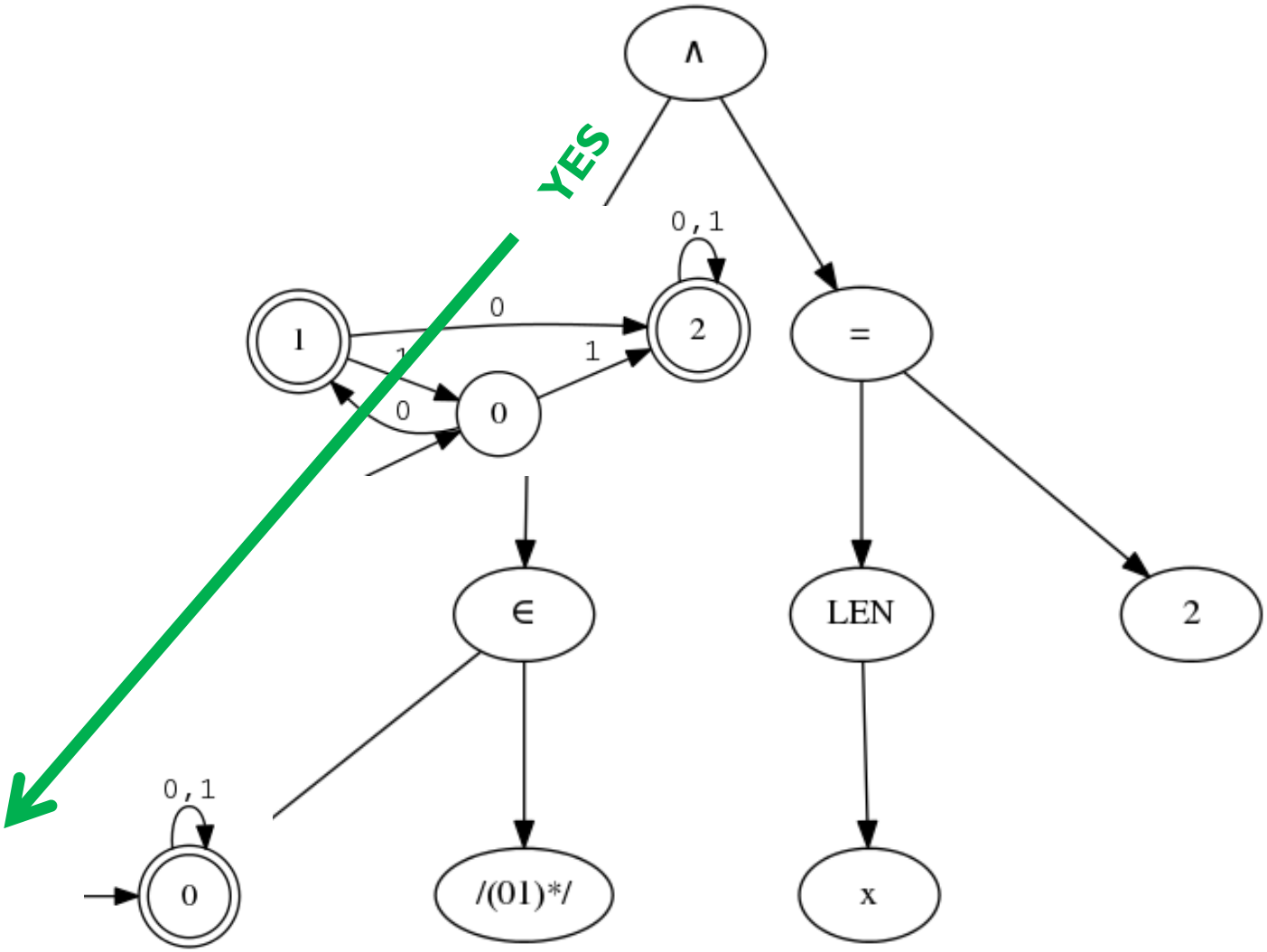
String Automata Construction

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



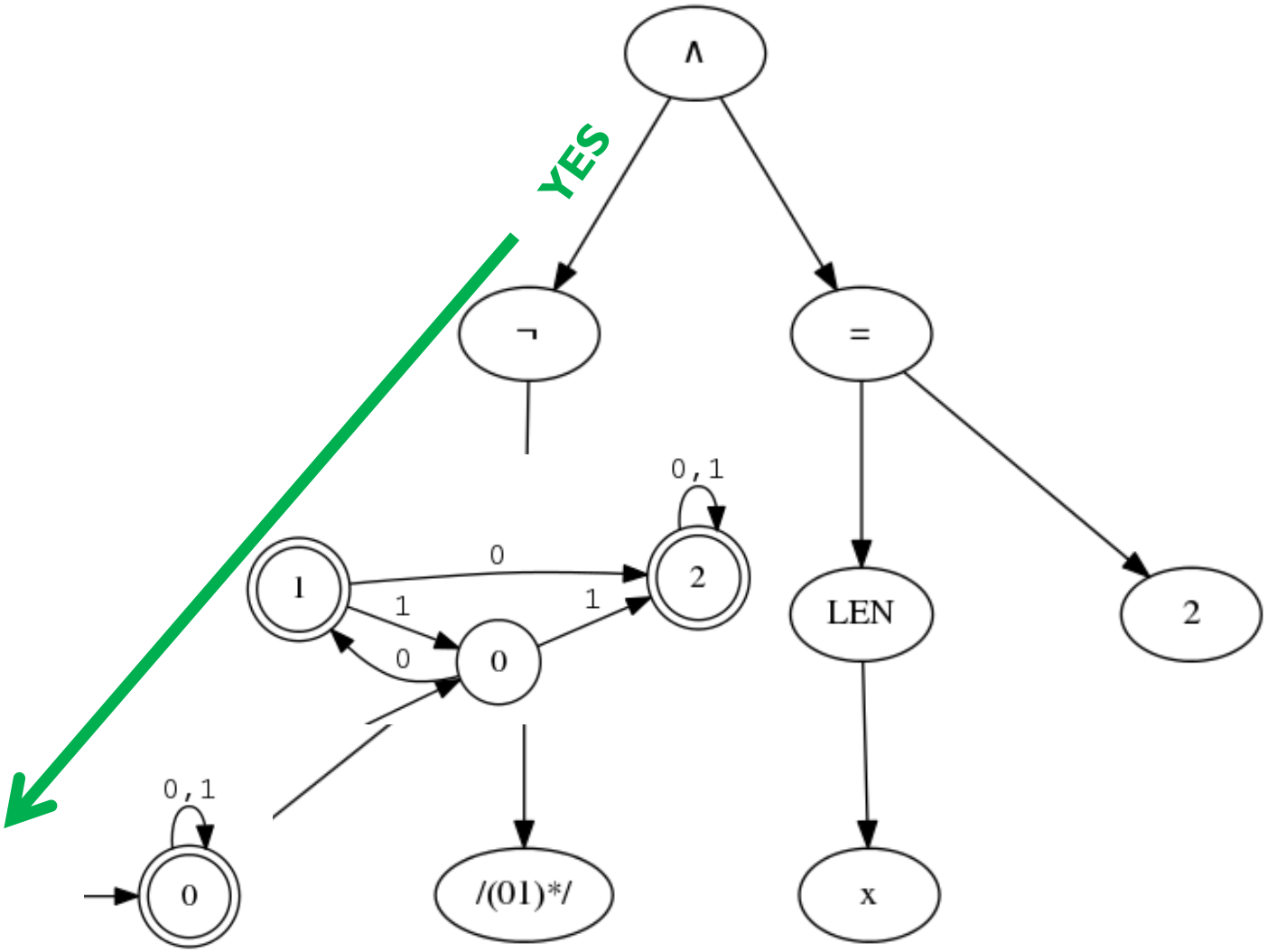
String Automata Construction

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



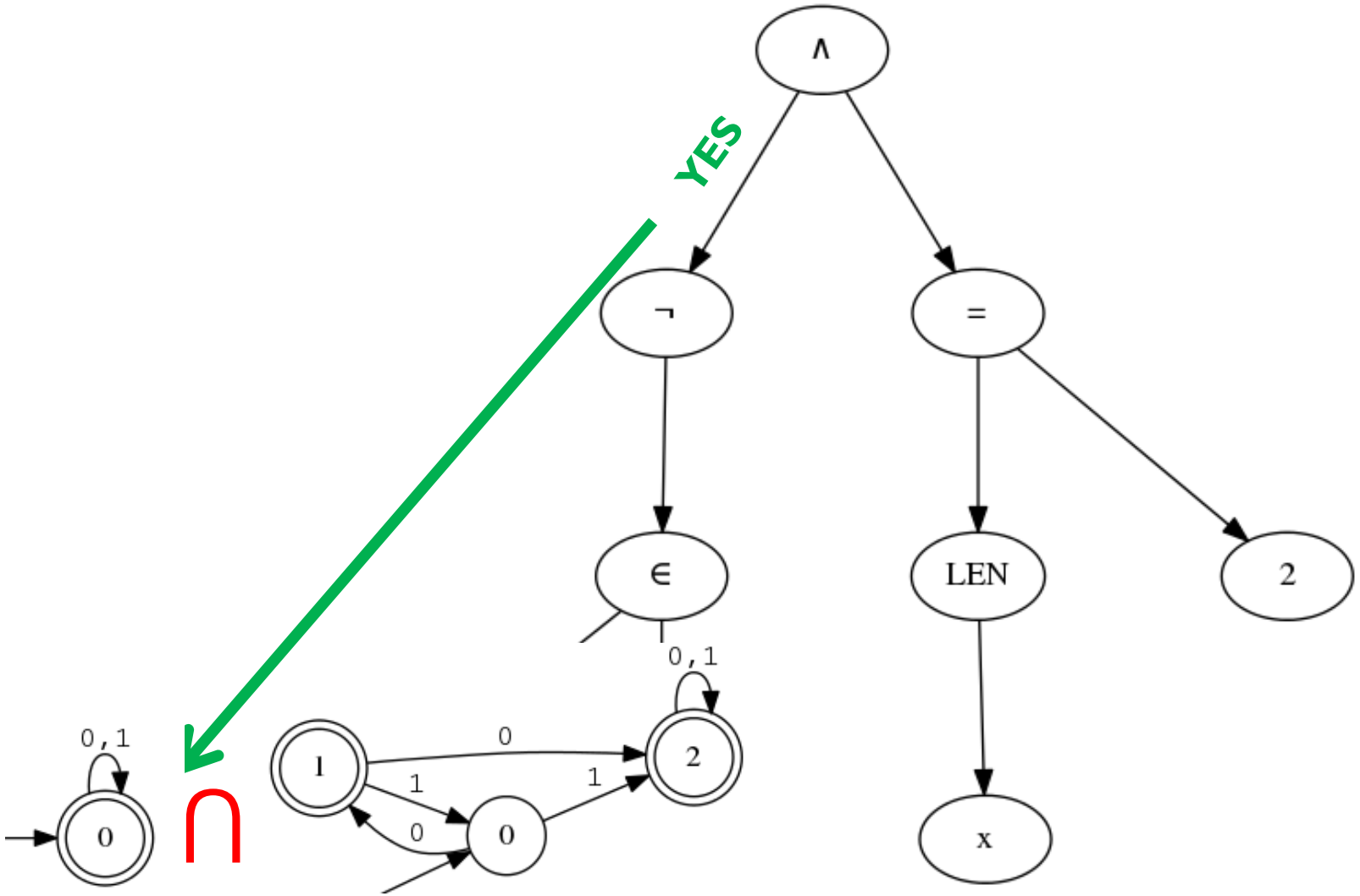
String Automata Construction

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



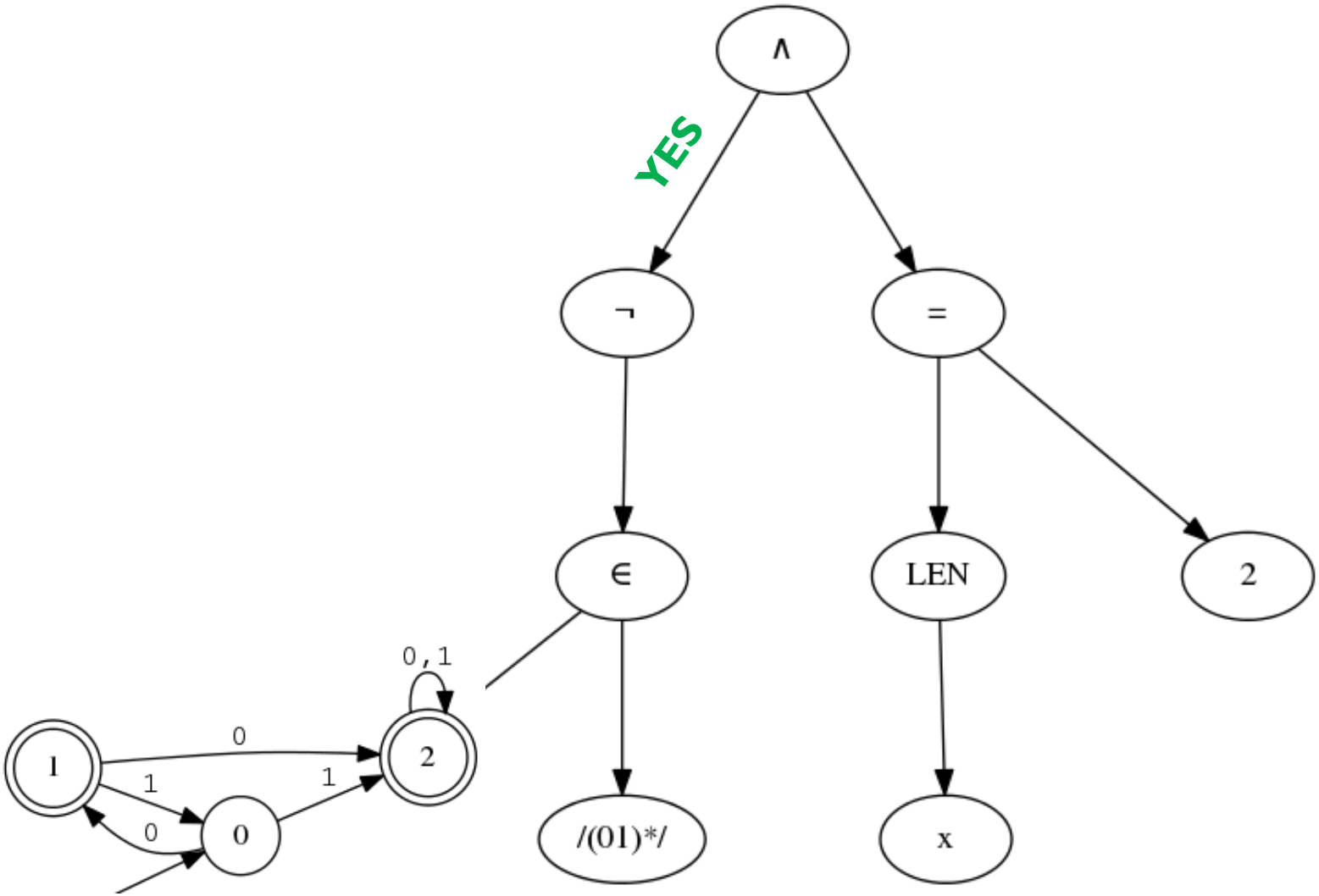
String Automata Construction

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



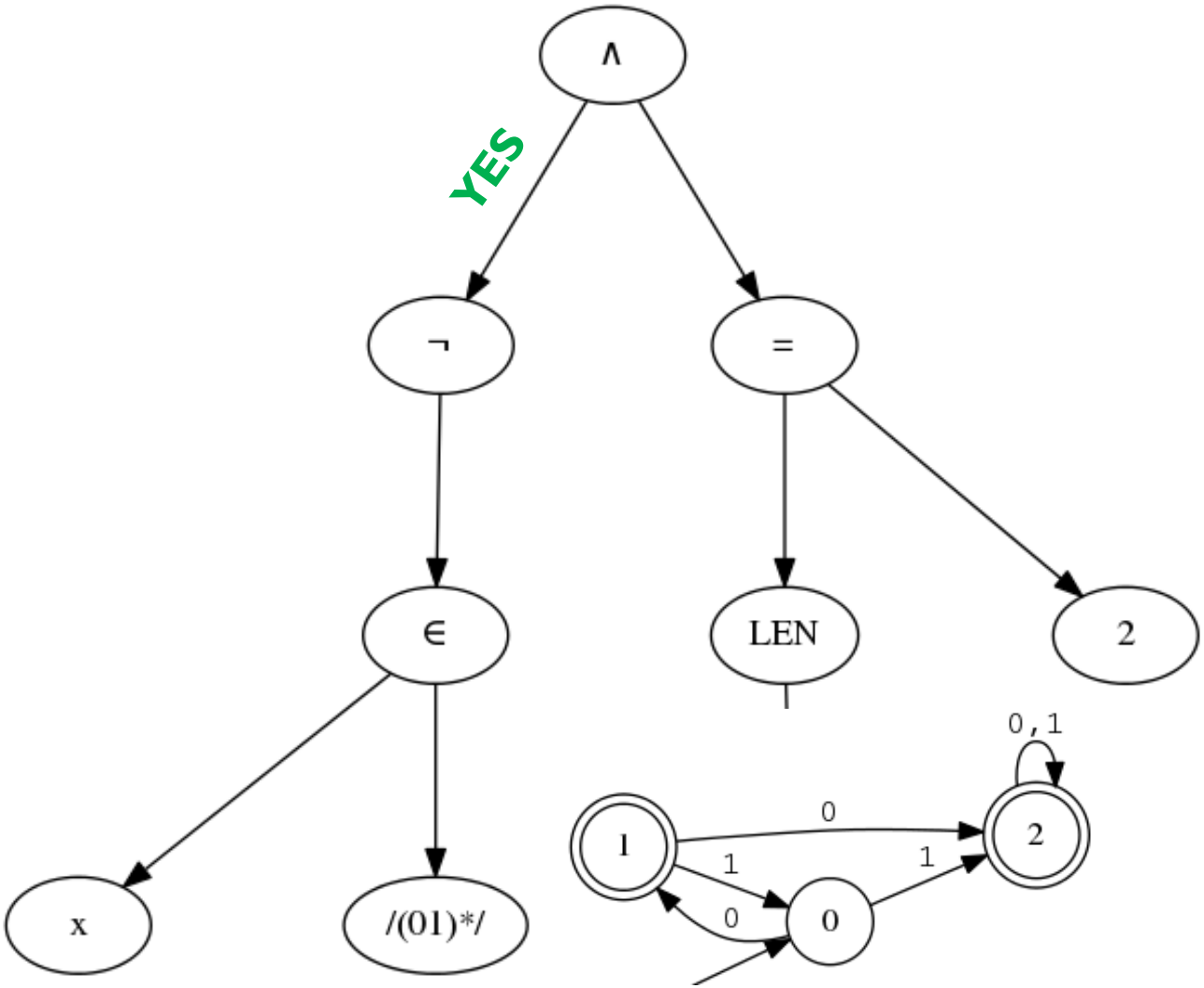
String Automata Construction

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



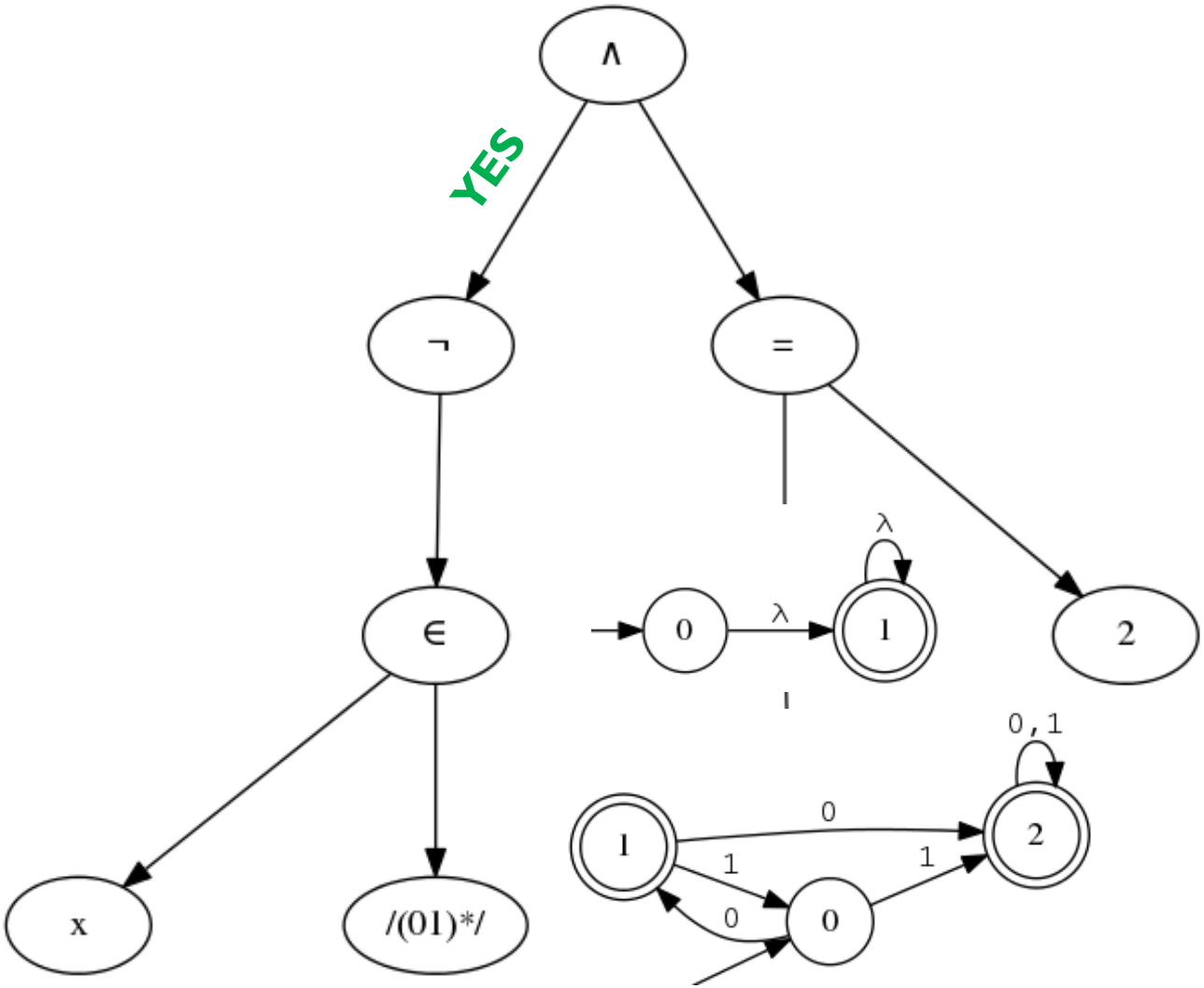
String Automata Construction

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



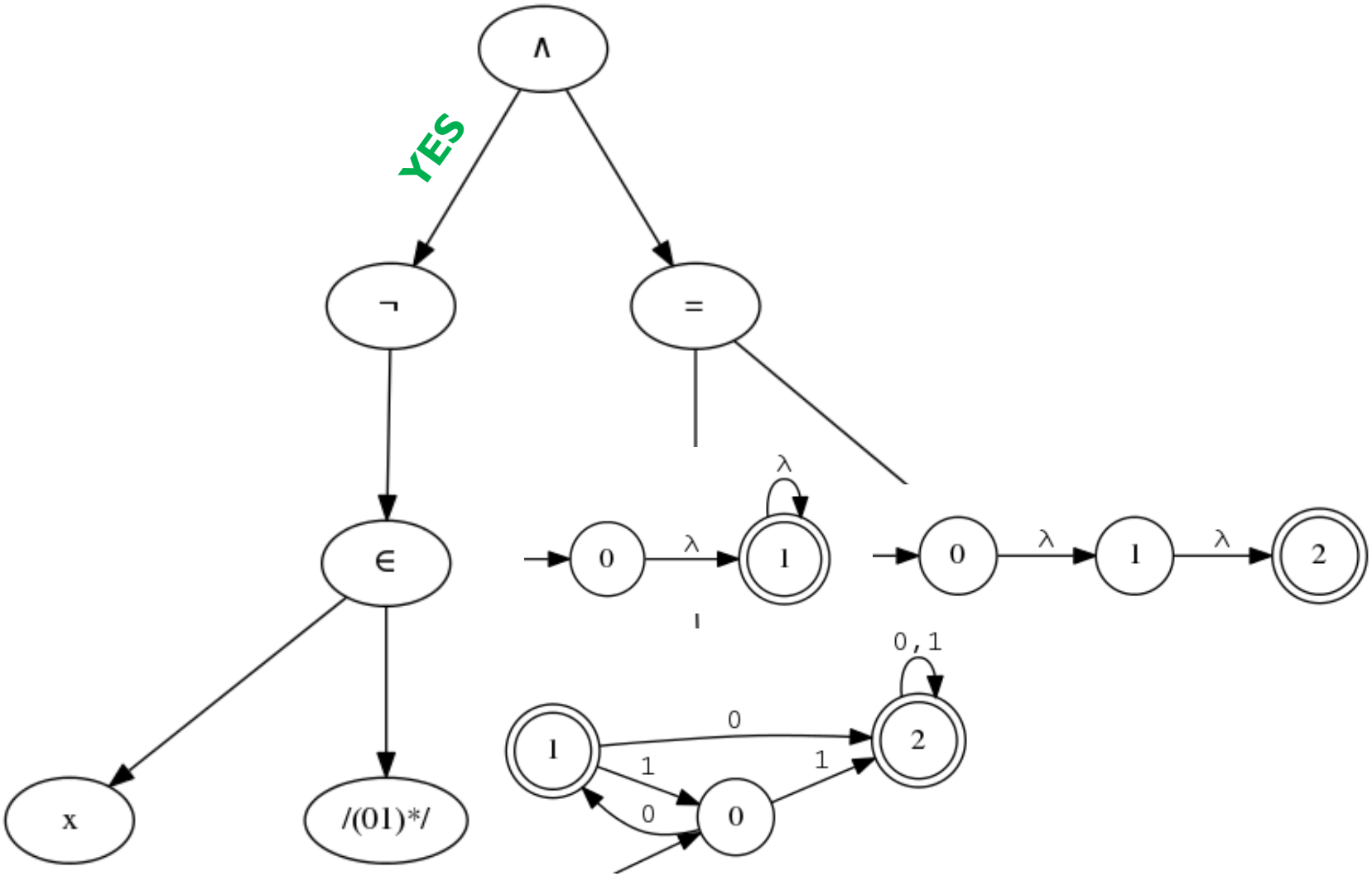
String Automata Construction

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



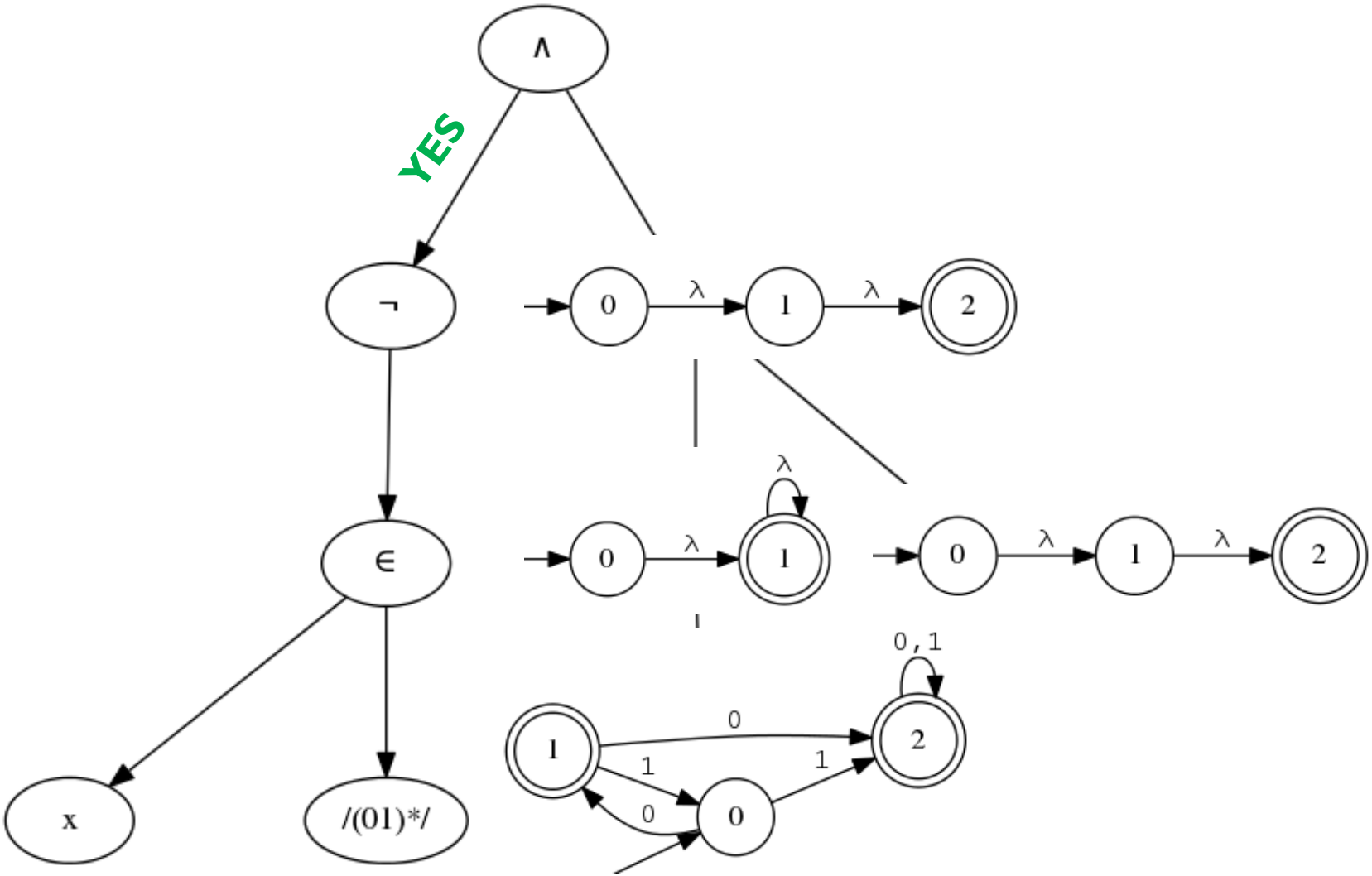
String Automata Construction

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



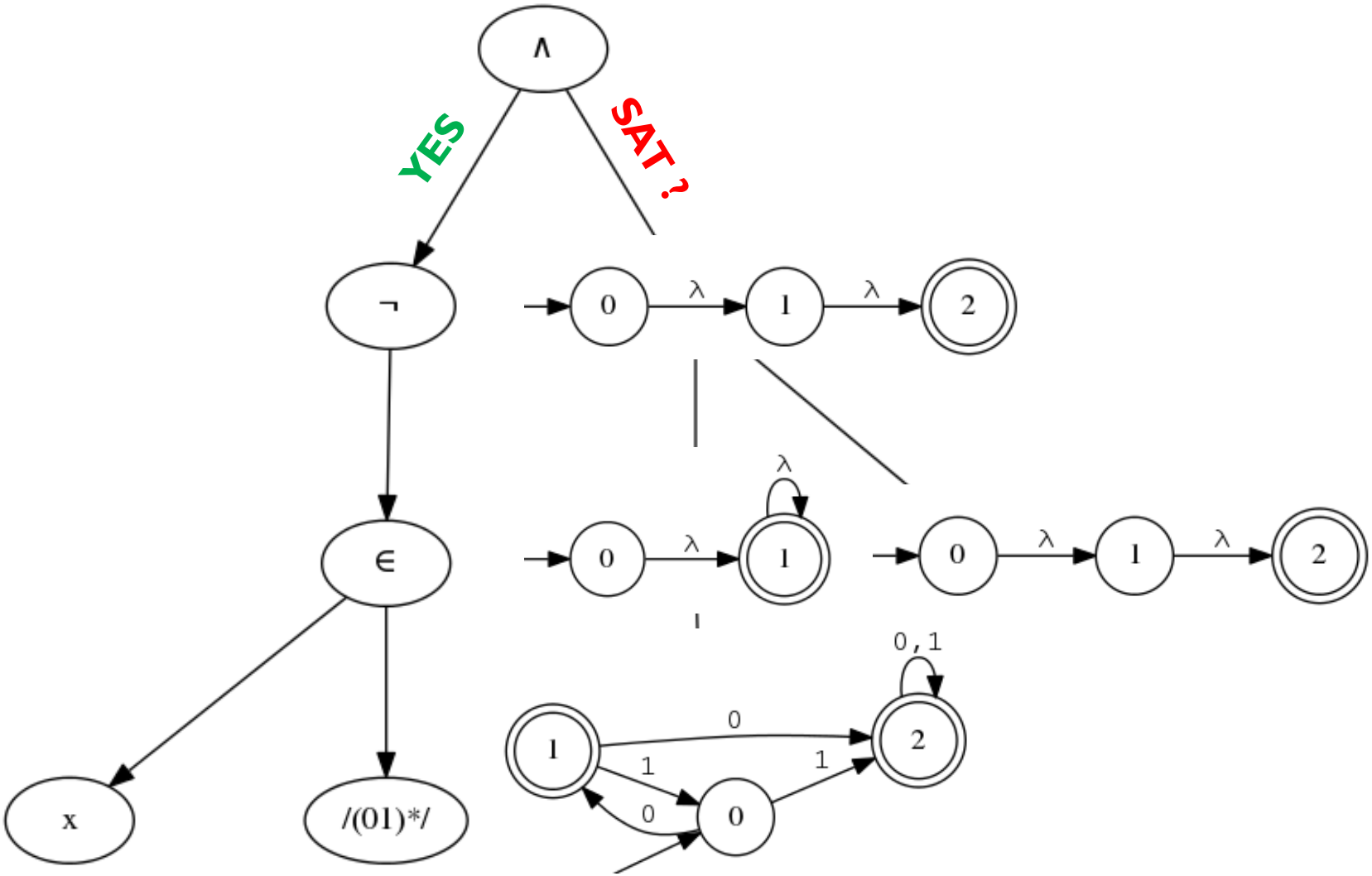
String Automata Construction

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



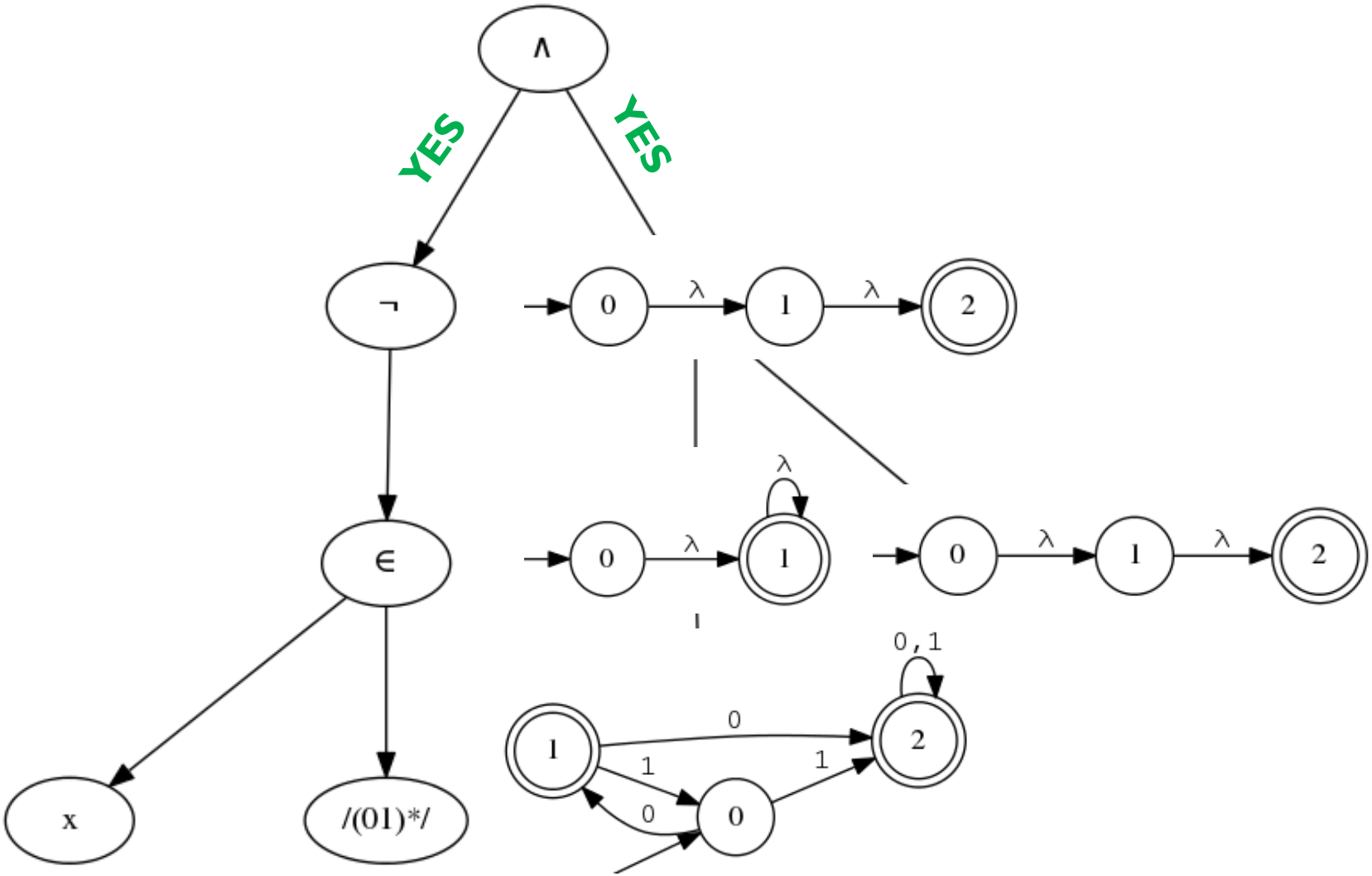
String Automata Construction

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



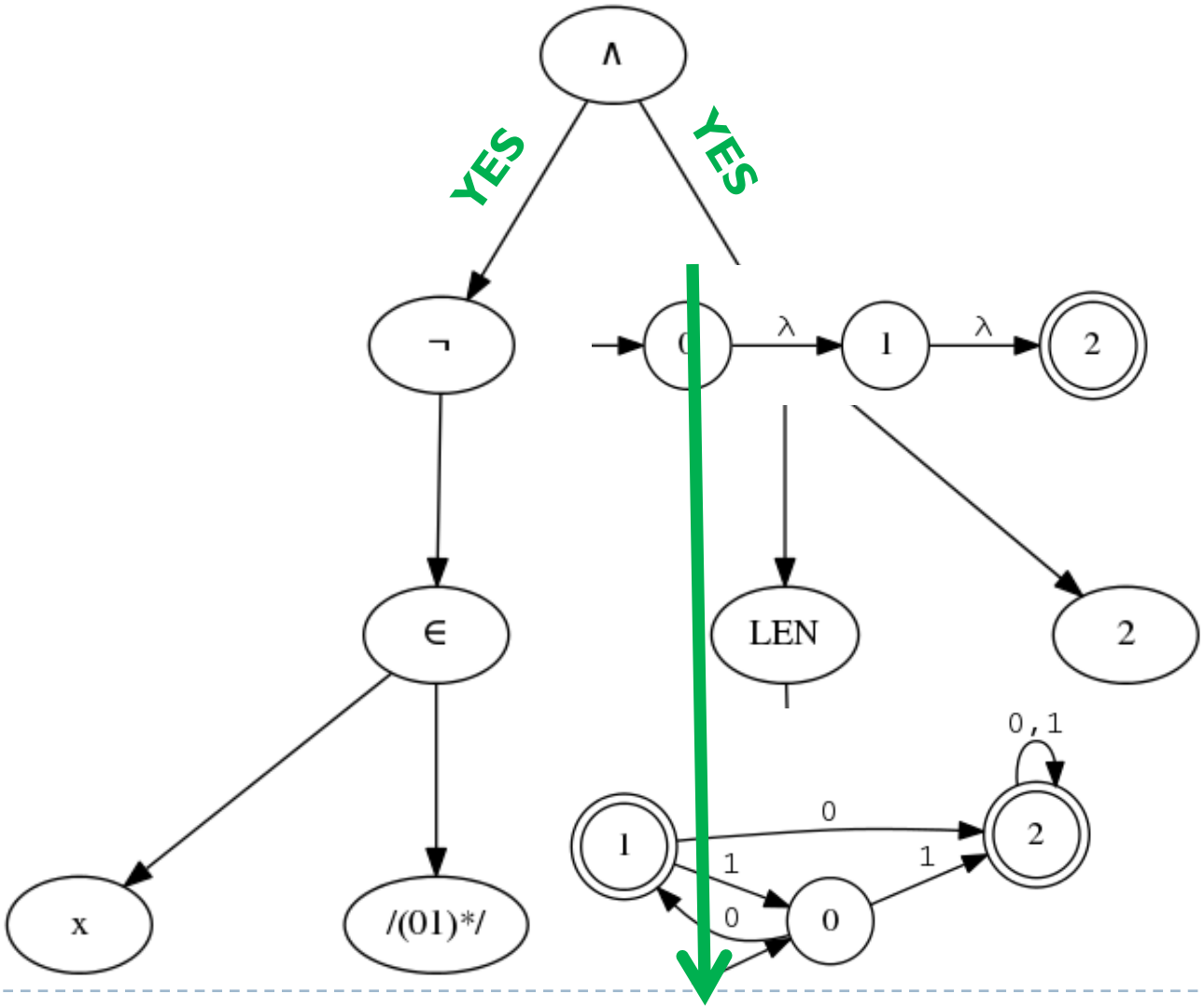
String Automata Construction

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



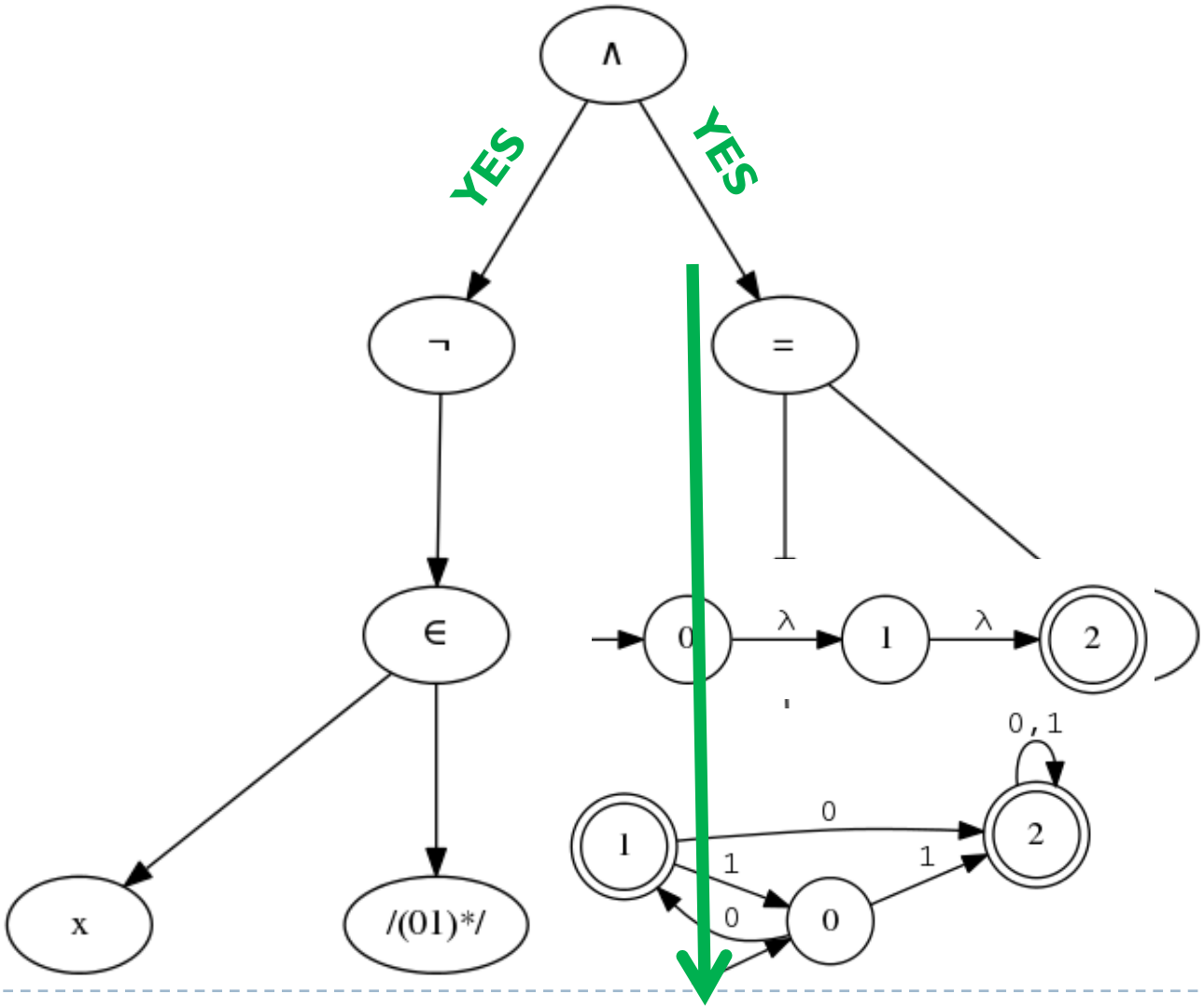
String Automata Construction

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



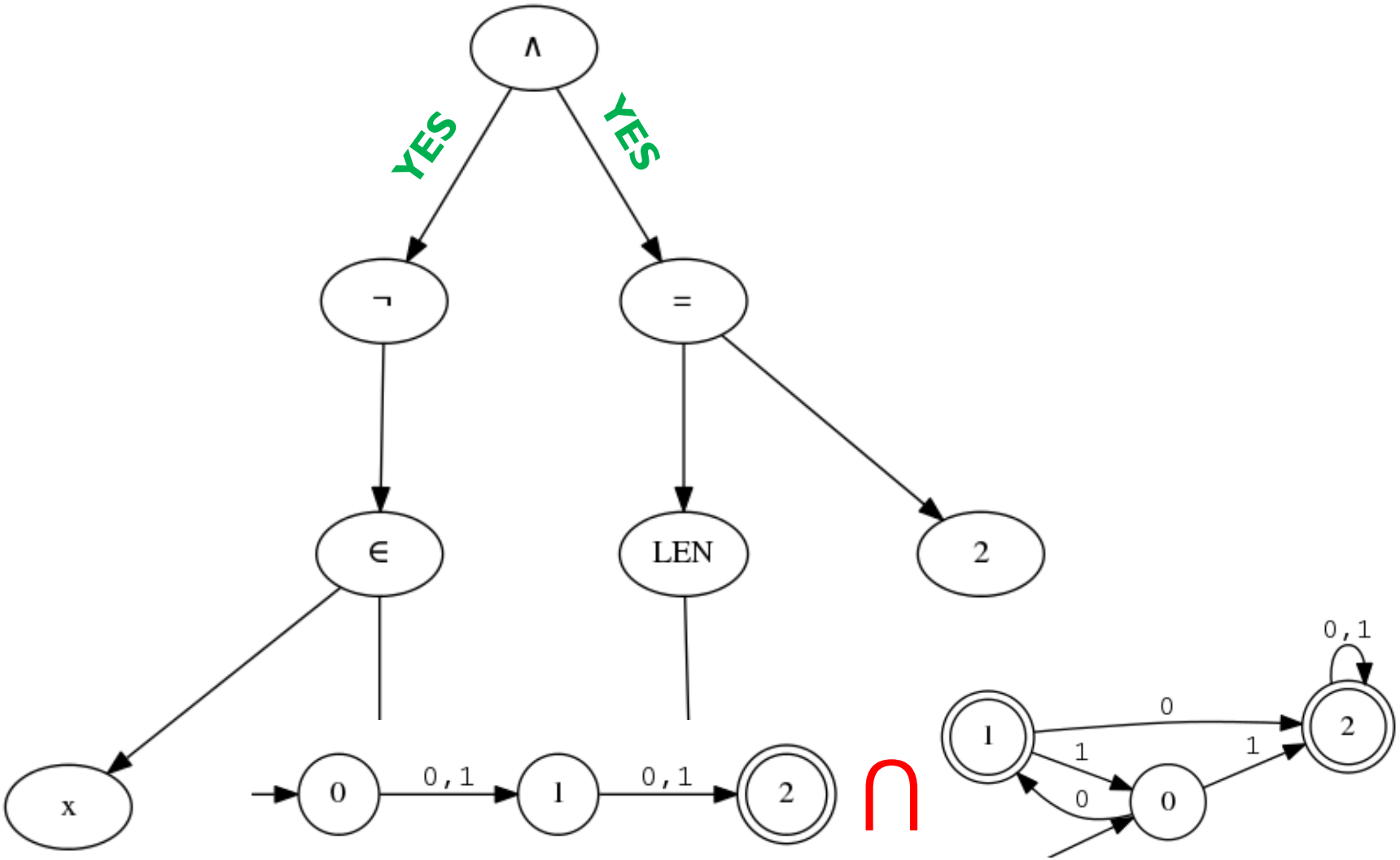
String Automata Construction

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



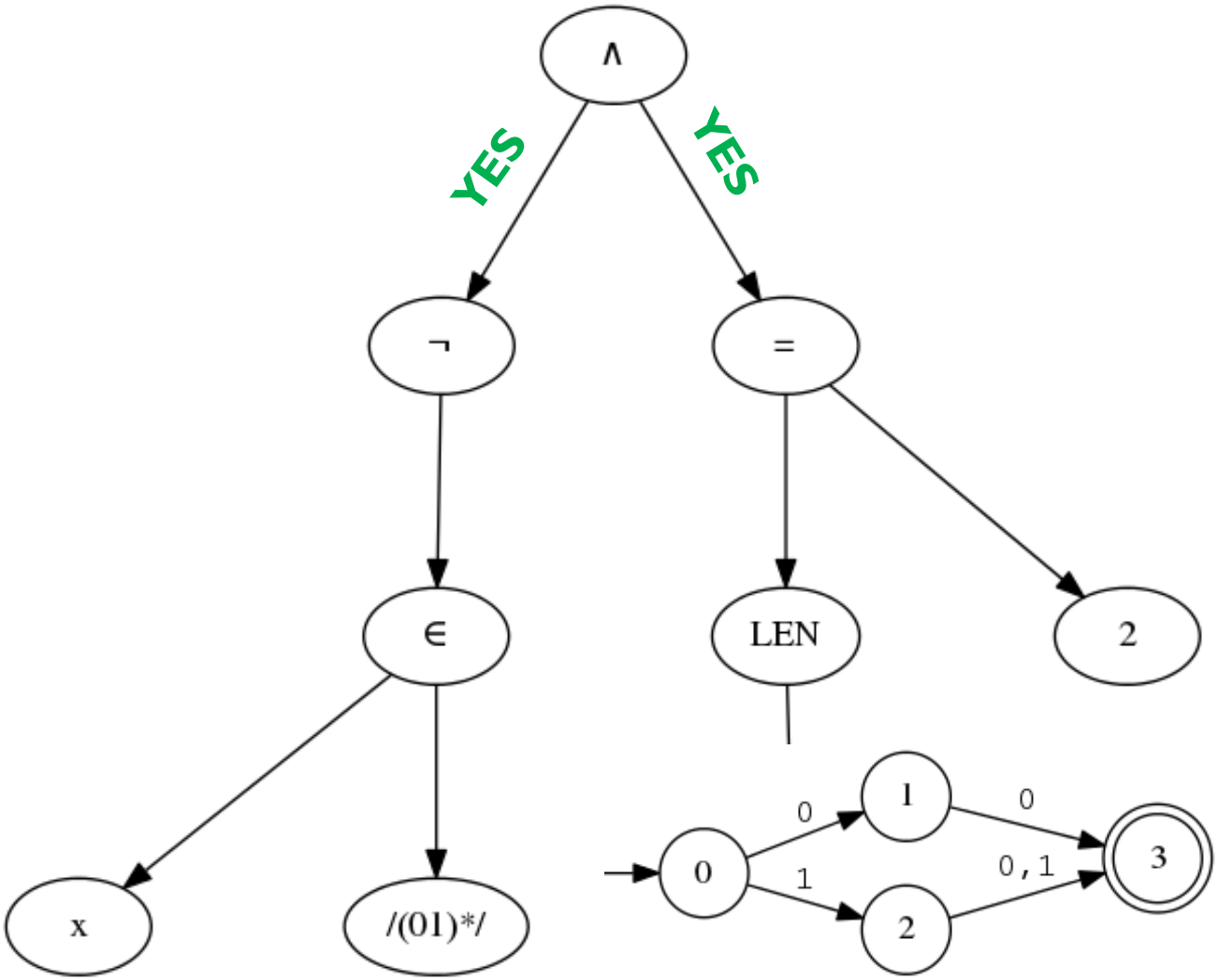
String Automata Construction

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



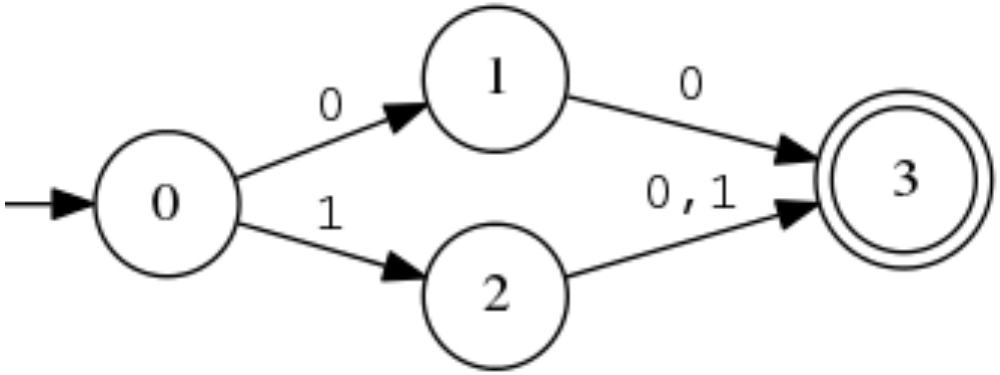
String Automata Construction

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



String Automata Construction

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



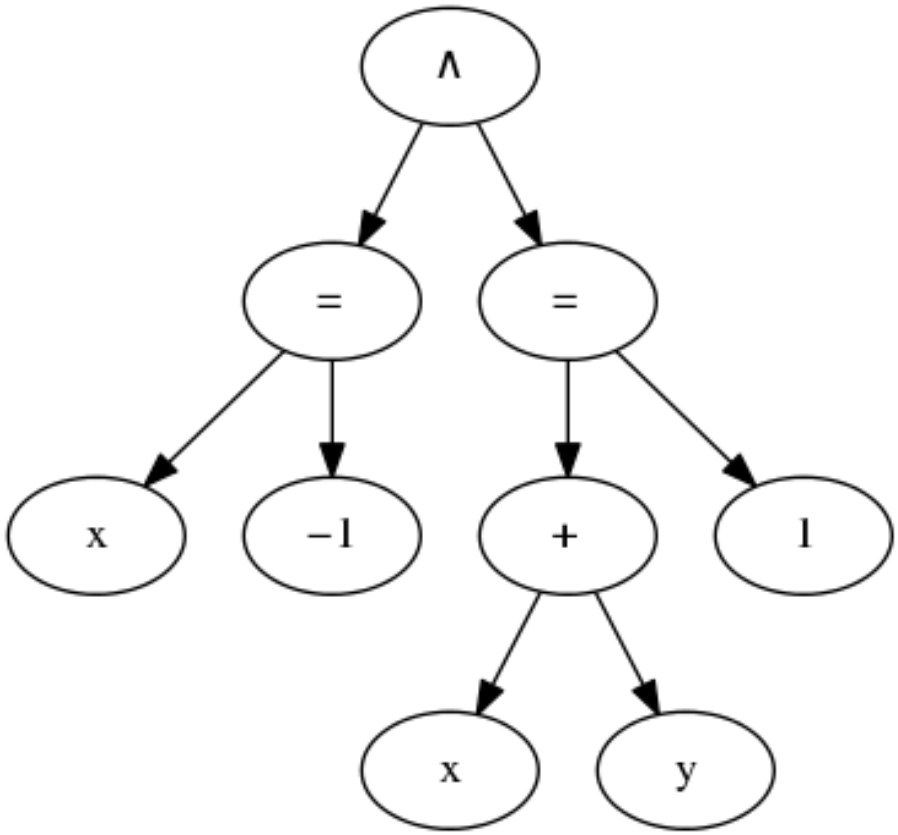
00, 10, 11

Integer Constraints

$$
\begin{array}{lll}
C & \longrightarrow & bterm \\
\\
bterm & \longrightarrow & \texttt{v} \mid \texttt{true} \mid \texttt{false} \\
 & \mid & \neg bterm \mid bterm \wedge bterm \mid bterm \vee bterm \mid (bterm) \\
 & \mid & sterm = sterm \\
 & \mid & \texttt{match}(sterm, sterm) \\
 & \mid & \texttt{contains}(sterm, sterm) \\
 & \mid & \texttt{begins}(sterm, sterm) \\
 & \mid & \texttt{ends}(sterm, sterm) \\
 & \mid & iterm = iterm \mid iterm < iterm \mid iterm > iterm \\
\\
iterm & \longrightarrow & \texttt{v} \mid \texttt{n} \\
 & \mid & iterm + iterm \mid iterm - iterm \mid iterm \times \texttt{n} \mid (iterm) \\
 & \mid & \texttt{length}(sterm) \mid \texttt{toint}(sterm) \\
 & \mid & \texttt{indexof}(sterm, sterm) \\
 & \mid & \texttt{lastindexof}(sterm, sterm) \\
\\
sterm & \longrightarrow & \texttt{v} \mid \varepsilon \mid \texttt{s} \\
 & \mid & sterm.sterm \mid sterm|sterm \mid sterm^* \mid (sterm) \\
 & \mid & \texttt{charat}(sterm, iterm) \mid \texttt{tostring}(iterm) \\
 & \mid & \texttt{toupper}(sterm) \mid \texttt{tolower}(sterm) \\
 & \mid & \texttt{substring}(sterm, iterm, iterm) \\
 & \mid & \texttt{replacefirst}(sterm, sterm, sterm) \\
 & \mid & \texttt{replacelast}(sterm, sterm, sterm) \\
 & \mid & \texttt{replaceall}(sterm, sterm, sterm)
\end{array}
$$

Integer Automata Construction

$$C \equiv x = -1 \wedge \mathrm{x} + \mathrm{y} = 1$$

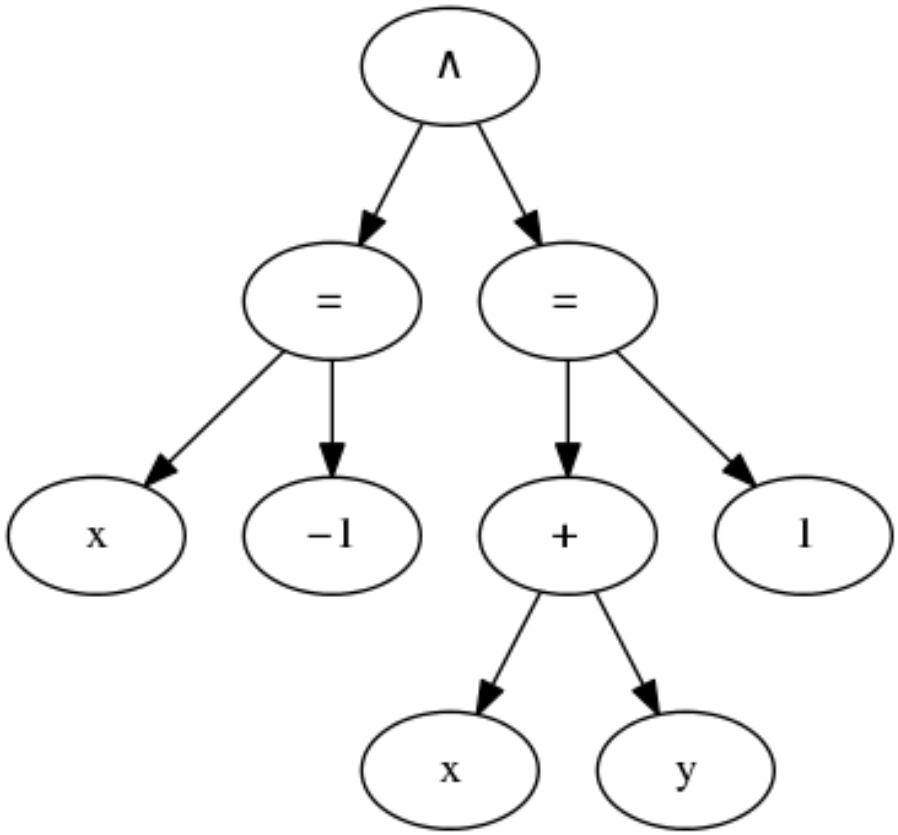


Integer Automata Construction

$$C \equiv x = -1 \wedge \mathrm{x} + \mathrm{y} = 1$$

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

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

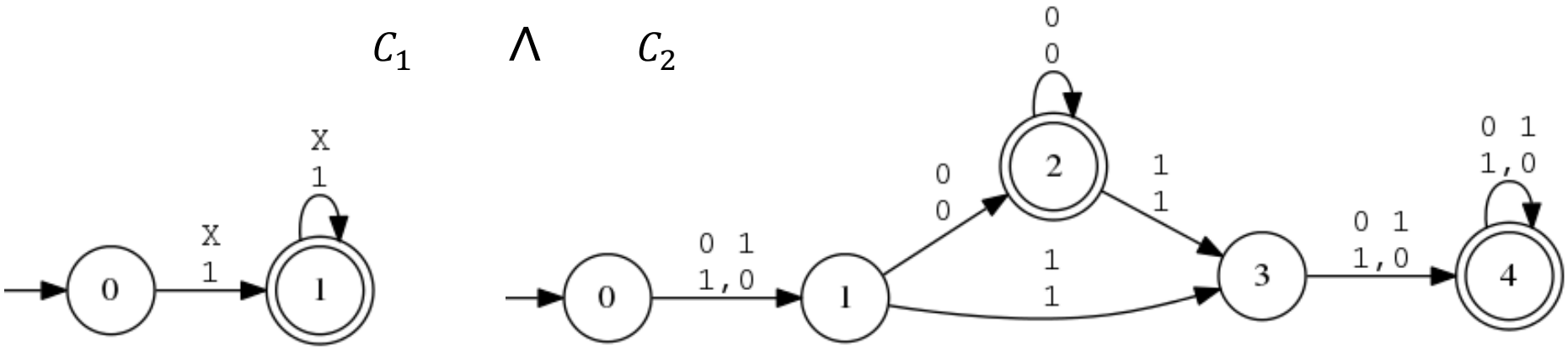


Integer Automata Construction

$$C \equiv x = -1 \wedge \mathrm{x} + \mathrm{y} = 1$$

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

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

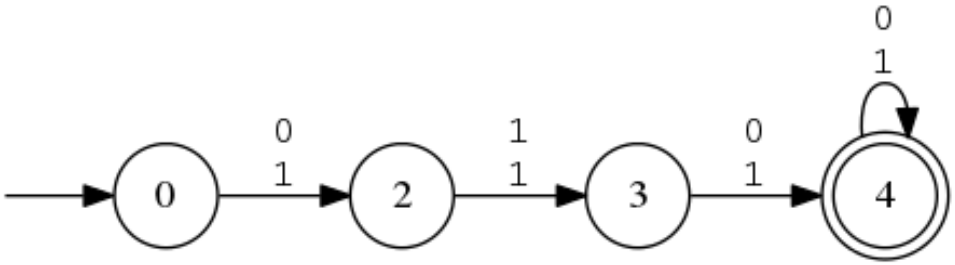


- Using automata construction techniques described in:

C. Bartzis and Tevfik 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

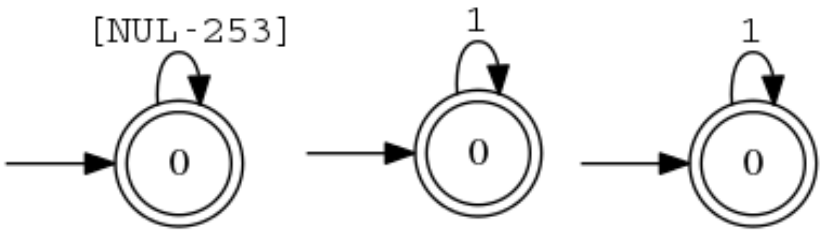
Mixing String – LIA Constraints

$$
\begin{array}{lll}
C & \longrightarrow & bterm \\
\\
bterm & \longrightarrow & \texttt{v} \mid \texttt{true} \mid \texttt{false} \\
 & \mid & \neg bterm \mid bterm \wedge bterm \mid bterm \vee bterm \mid (bterm) \\
 & \mid & sterm = sterm \\
 & \mid & \texttt{match}(sterm, sterm) \\
 & \mid & \texttt{contains}(sterm, sterm) \\
 & \mid & \texttt{begins}(sterm, sterm) \\
 & \mid & \texttt{ends}(sterm, sterm) \\
 & \mid & iterm = iterm \mid iterm < iterm \mid iterm > iterm \\
\\
iterm & \longrightarrow & \texttt{v} \mid \texttt{n} \\
 & \mid & iterm + iterm \mid iterm - iterm \mid iterm \times \texttt{n} \mid (iterm) \\
 & \mid & \texttt{length}(sterm) \mid \texttt{toint}(sterm) \\
 & \mid & \texttt{indexof}(sterm, sterm) \\
 & \mid & \texttt{lastindexof}(sterm, sterm) \\
\\
sterm & \longrightarrow & \texttt{v} \mid \varepsilon \mid \texttt{s} \\
 & \mid & sterm.sterm \mid sterm|sterm \mid sterm^* \mid (sterm) \\
 & \mid & \texttt{charat}(sterm, iterm) \mid \texttt{tostring}(iterm) \\
 & \mid & \texttt{toupper}(sterm) \mid \texttt{tolower}(sterm) \\
 & \mid & \texttt{substring}(sterm, iterm, iterm) \\
 & \mid & \texttt{replacefirst}(sterm, sterm, sterm) \\
 & \mid & \texttt{replacelast}(sterm, sterm, sterm) \\
 & \mid & \texttt{replaceall}(sterm, sterm, sterm)
\end{array}
$$

Mixing String – LIA Constraints

$$length(s) = x \wedge x = 2y$$

- We can try to solve using string constraint solving
- Integer variables are represented with unary automaton
  - $A_s = L(\Sigma_s^*), A_x = L(\lambda^*), A_y = L(\lambda^*), A_{2y} = L(\lambda^*)$
  - $A_s \rightarrow A_{length};\ L(\Sigma_s^*) \rightarrow L(\lambda^*)$
  - $L(\lambda^*) = L(\lambda^*) \wedge L(\lambda^*) = L(\lambda^*)$

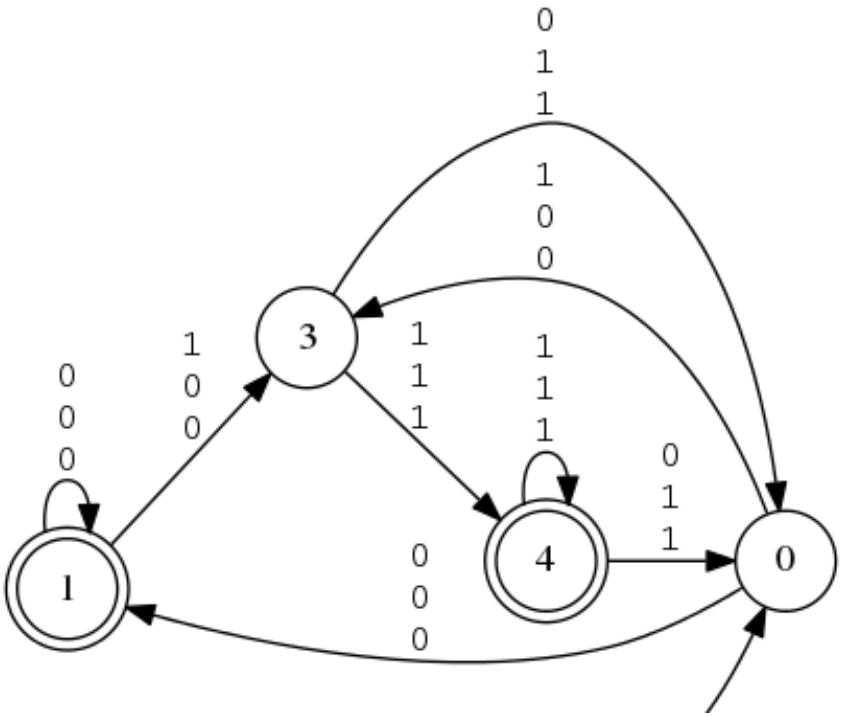


  - $s = 'abc', x = 1, y = 2$

Mixing String – LIA Constraints

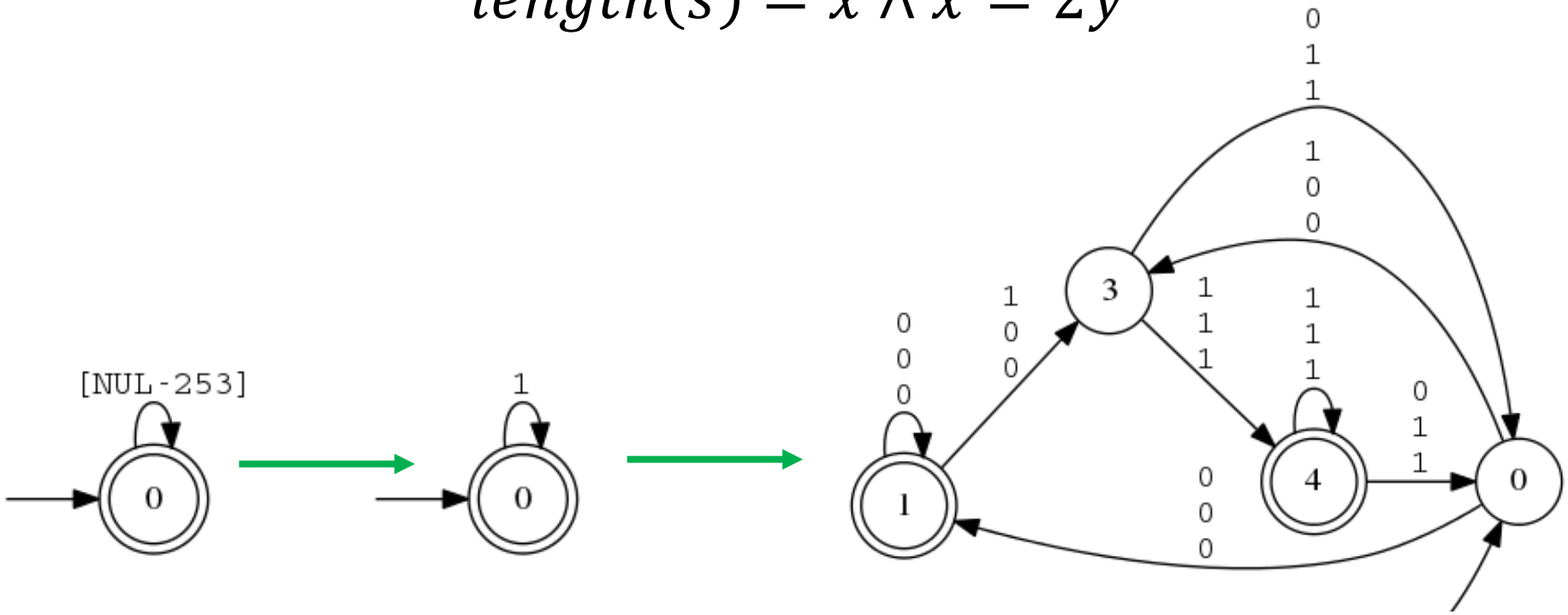
$$length(s) = x \wedge x = 2y$$

- We can do better by using multi-track binary integer automaton
  - There will be three integer variables: $length(s), x, y$



Mixing String – LIA Constraints

$$length(s) = x \wedge x = 2y$$

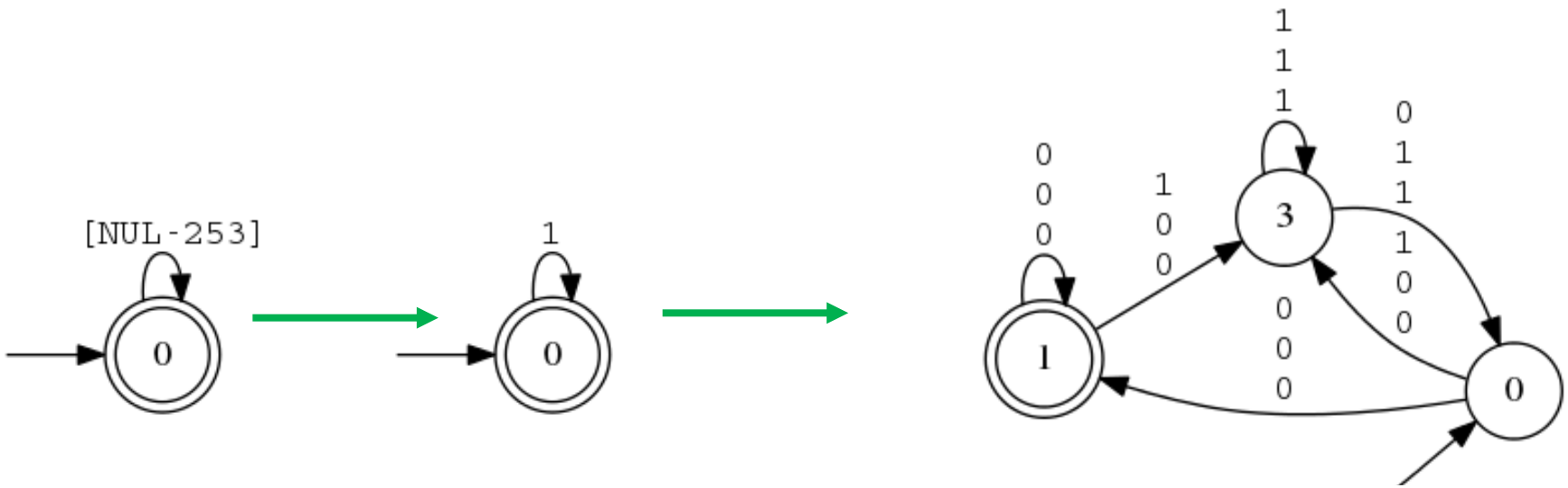


- Using automata techniques described in:

F. Yu, T. Bultan, O. H. Ibarra. Symbolic String Verification: Combining String Analysis and Size Analysis. *TACAS'09*

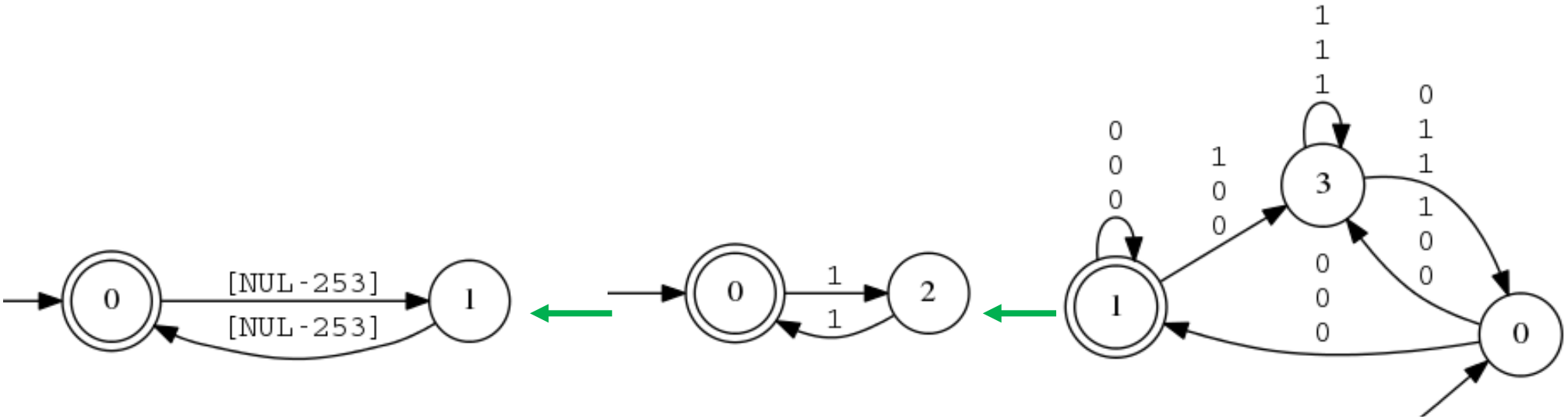
Mixing String – LIA Constraints

$$length(s) = x \wedge x = 2y$$



Mixing String – LIA Constraints

$$length(s) = x \wedge x = 2y$$



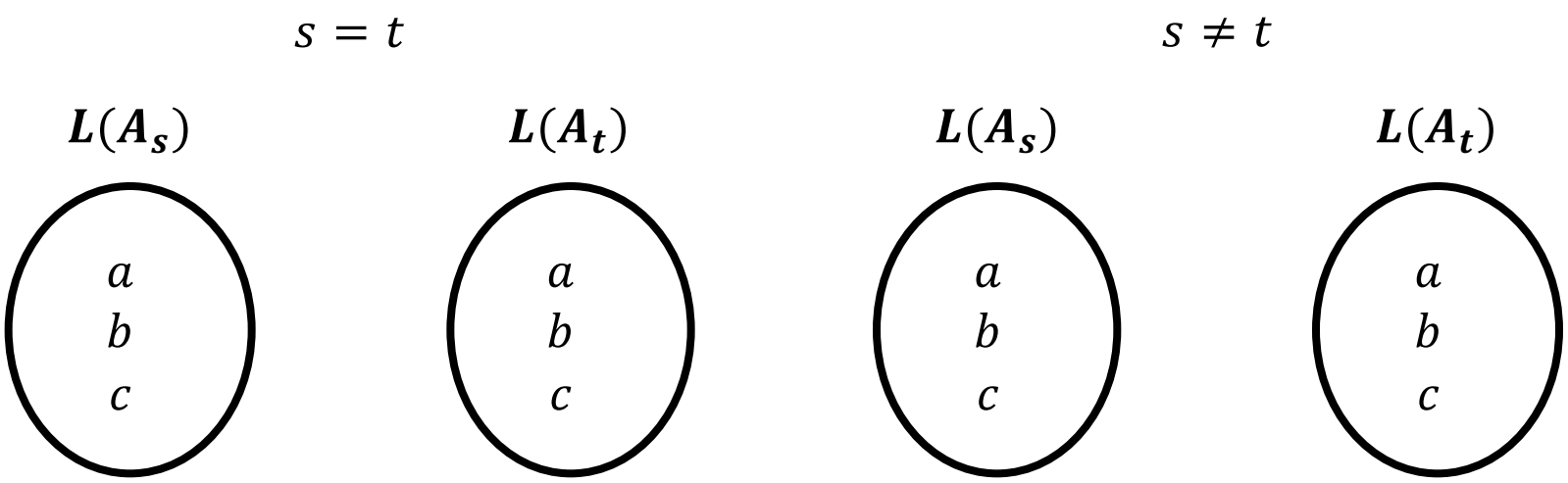
Relational String Constraints

$$
\begin{array}{lll}
C & \longrightarrow & bterm \\
\\
bterm & \longrightarrow & \texttt{v} \mid \texttt{true} \mid \texttt{false} \\
 & \mid & \neg bterm \mid bterm \wedge bterm \mid bterm \vee bterm \mid (bterm) \\
 & \mid & sterm = sterm \\
 & \mid & \texttt{match}(sterm, sterm) \\
 & \mid & \texttt{contains}(sterm, sterm) \\
 & \mid & \texttt{begins}(sterm, sterm) \\
 & \mid & \texttt{ends}(sterm, sterm) \\
 & \mid & iterm = iterm \mid iterm < iterm \mid iterm > iterm \\
\\
iterm & \longrightarrow & \texttt{v} \mid \texttt{n} \\
 & \mid & iterm + iterm \mid iterm - iterm \mid iterm \times \texttt{n} \mid (iterm) \\
 & \mid & \texttt{length}(sterm) \mid \texttt{toint}(sterm) \\
 & \mid & \texttt{indexof}(sterm, sterm) \\
 & \mid & \texttt{lastindexof}(sterm, sterm) \\
\\
sterm & \longrightarrow & \texttt{v} \mid \varepsilon \mid \texttt{s} \\
 & \mid & sterm.sterm \mid sterm|sterm \mid sterm^* \mid (sterm) \\
 & \mid & \texttt{charat}(sterm, iterm) \mid \texttt{tostring}(iterm) \\
 & \mid & \texttt{toupper}(sterm) \mid \texttt{tolower}(sterm) \\
 & \mid & \texttt{substring}(sterm, iterm, iterm) \\
 & \mid & \texttt{replacefirst}(sterm, sterm, sterm) \\
 & \mid & \texttt{replacelast}(sterm, sterm, sterm) \\
 & \mid & \texttt{replaceall}(sterm, sterm, sterm)
\end{array}
$$

Relational String Constraint Example

$$s = t \wedge s \neq t$$

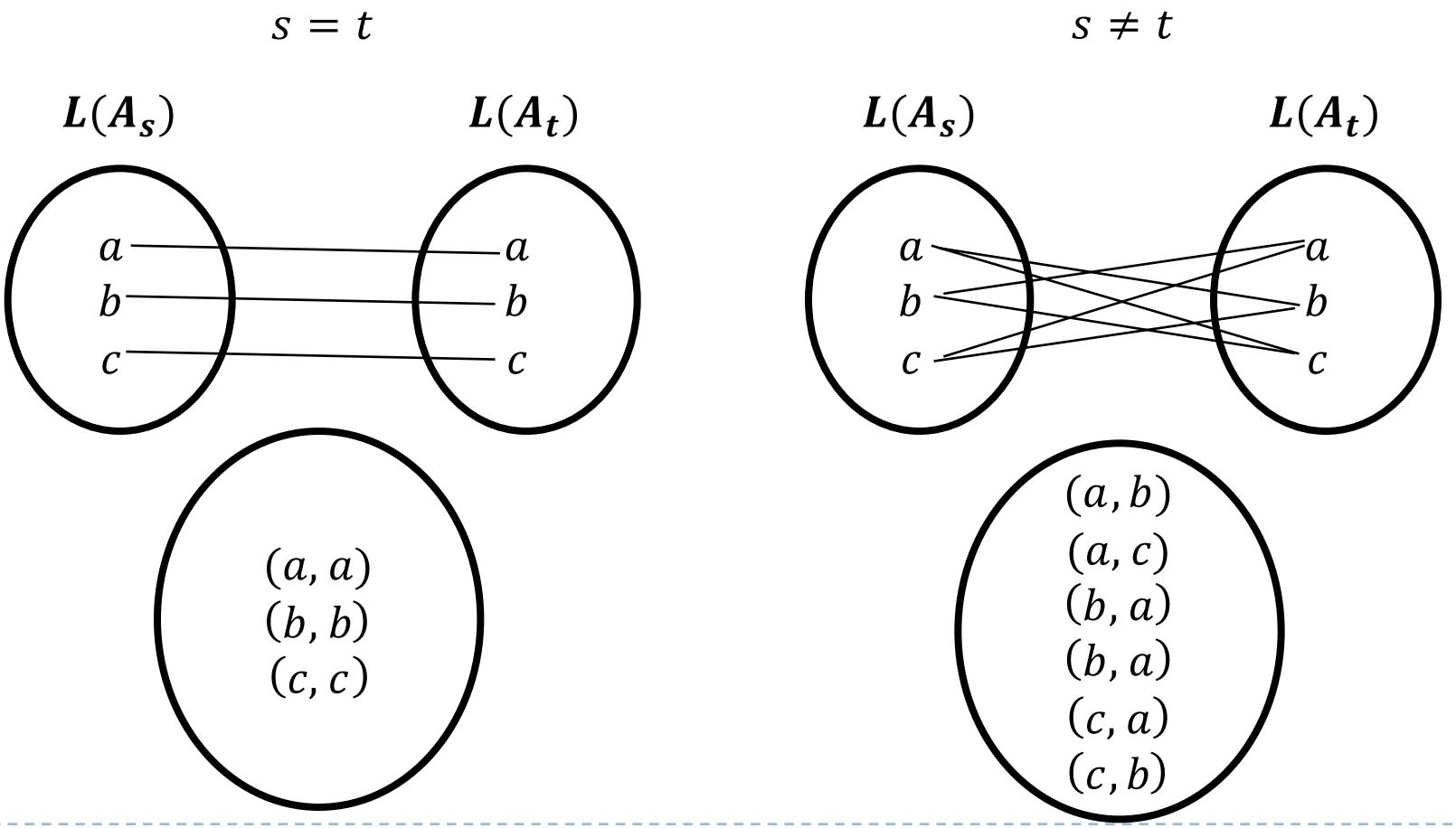
- Single track string automata results in SAT (w/ optimizations)
  - Assume $\Sigma = \{a, b, c\}$



Relational String Constraint Example

$$s = t \wedge s \neq t$$

- Multi-track string automata can keep relations

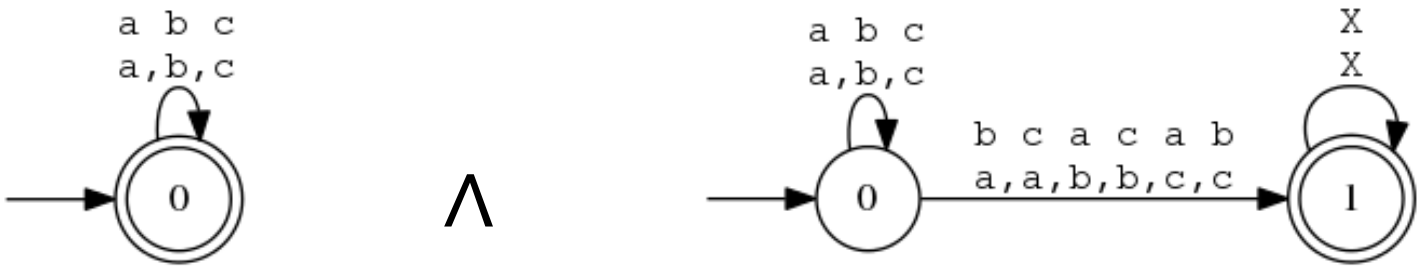


Relational String Constraint Example

$$s = t \wedge s \neq t$$

- Each track represents values of one string variable

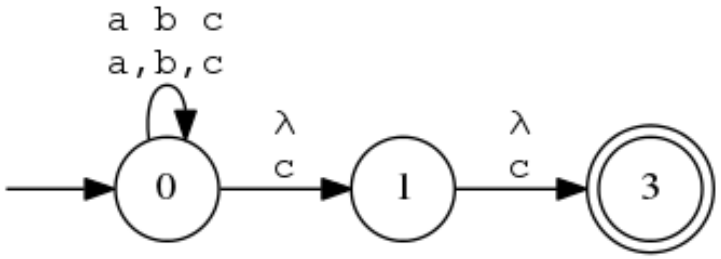
$s = t$ $\wedge$ $s \neq t$



Word Equations

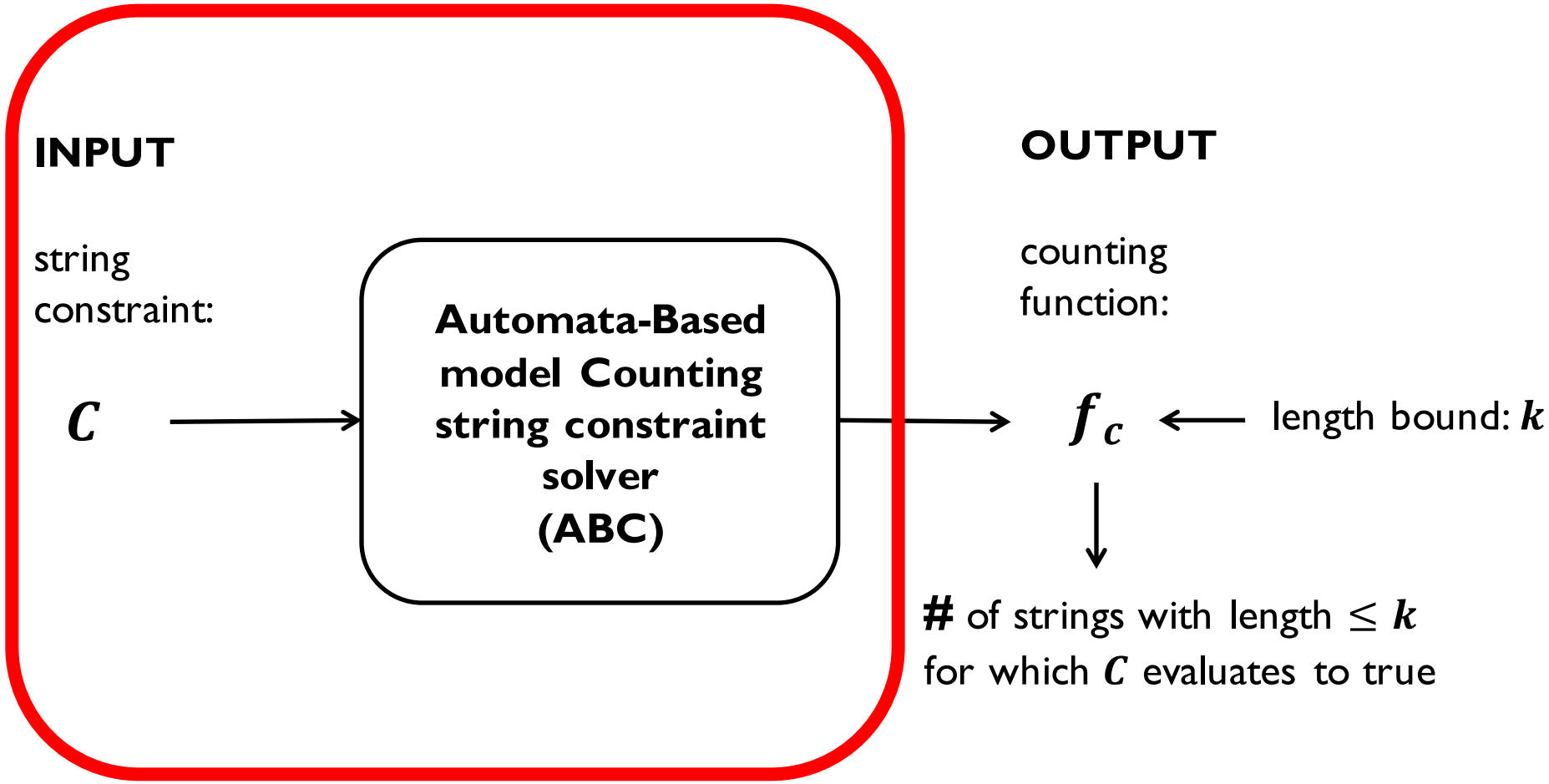
- Let X (the first track), Y (the second track), be two string variables
- λ is the padding symbol

$$X = Y.cc$$

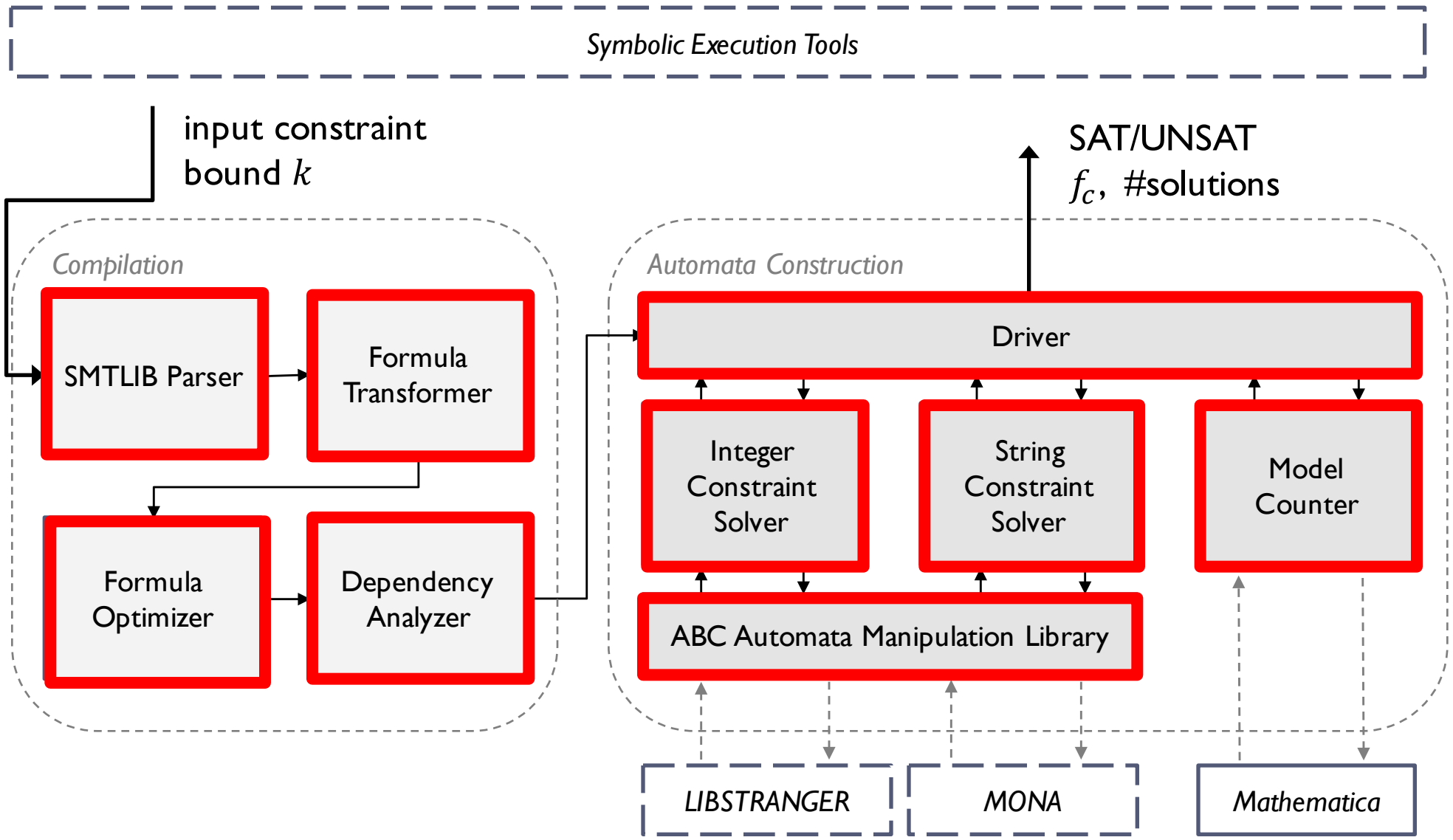


- $X = Y.C,\ X = C.Y,\ C = X.Y$ can be precisely represented with multi-track automaton
- $X = Y.Z$ is non-linear, can be represented with over- or under-approximation

Model Counting String Constraints Solver



ABC Tool


Symbolic Execution Tools
input constraint
bound k
SAT/UNSAT
f_c, #solutions
Compilation
SMTLIB Parser
Formula Transformer
Formula Optimizer
Dependency Analyzer
Automata Construction
Driver
Integer Constraint Solver
String Constraint Solver
Model Counter
ABC Automata Manipulation Library
LIBSTRANGER
MONA
Mathematica

Experimental Evaluation

- We conducted experiments on 3 benchmark sets
  - A java benchmark with wide range of string operations
  - Kaluza Small & Kaluza Big benchmarks for satisfiability check

	Frequency of Operations Per 1000 Formulas				
	$\in$	.	=	LENGTH	REPLACEALL, *BEGINS* *SUBSTRING*, *ENDS* *CONTAINS*, *INDEXOF*
Kausler	0.42	386.10	129.39	382.54	716.5
Kaluza Small	30.29	93.89	224.87	46.84	0
Kaluza Big	38.12	129.53	164.64	60.46	0

Satisfiability Check Comparison

- Compared with CVC4
  - Used SMT-lib format of Kaluza benchmarks from CVC4

	ABC-CVC4 sat-sat	ABC-CVC4 unsat-unsat	ABC-CVC4 sat-unsat	ABC-CVC4 unsat-sat	ABC-CVC4 sat-timeout
sat/small	19728	3	0	0	0
sat/big	1587	0	0	0	0
unsat/small	8139	3013	74	0	0
unsat/big	3419	5904	2385	0	2359

  - Constraint solver performance for Kaluza benchmarks

	ABC Avg. Time (seconds)	CVC4 Avg. Time (seconds)
big	0.44	0.01
small	0.01	0.015

Kauser's Benchmark

- Extracted from 7 real-world server-side Java applications
  - Constraints were generated by extracting program path constraints through dynamic symbolic execution
  - CVC4 1.4 stable version was not able to handle Kauser's benchmark

# of satisfiable path constraints	Avg. # of BDD Nodes (each 16 bytes)	Avg. Running Time (seconds)
66236	69.51	0.0015

- Automata representation we use encodes the transition relation as a binary decision diagram

Model Counting String Constraints Solver