Automatic Repair of Regular Expressions

Rong Pan, Qinheping Hu, Gaowei Xu, Loris D'Antoni
OOPSLA 2019

Presented by Ayaka Yorihiro
PLDG 05/19/2021

Motivating Example

MasterCard Numbers are 16 digits numbers that start with 51-55...

User: [51|52|53|54|55]{2}[0-9]{14}

...this is actually equivalent to [|12345]{2}[0-9]{14}

ex) 1500000000000000 ❌

ex) |500000000000000 ❌

How can we automate the recovery process?

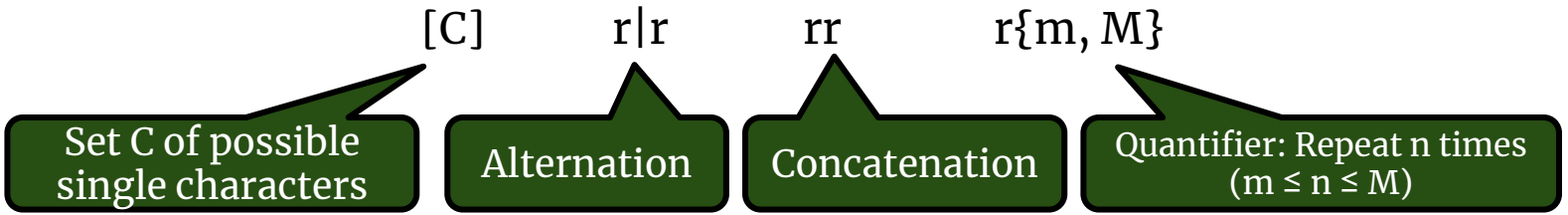
Contributions

Key Contributions:

- RFixer: First *sound and complete tool* for producing minimal repairs of Regular Expressions from examples
- Template-based approach for repair
- Two SMT Encodings for solving templates
  - Automata-based
  - Regular Expression operator-based

Background and Terminology

A **Regular Expression** r is formed using operators:



Note: [0-9] = [0123456789]

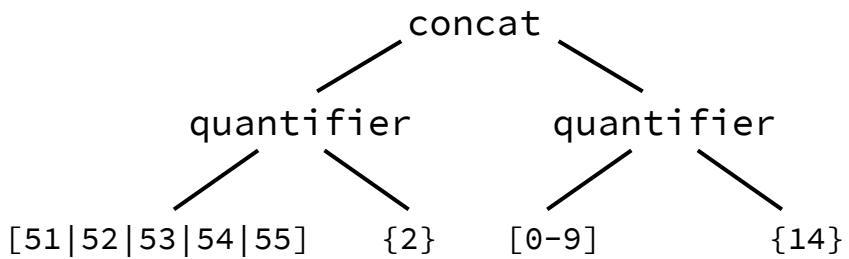
Note: r{n} is shorthand for r{n, n}

Note: r* (Kleene star) is represented by r{0, ∞}

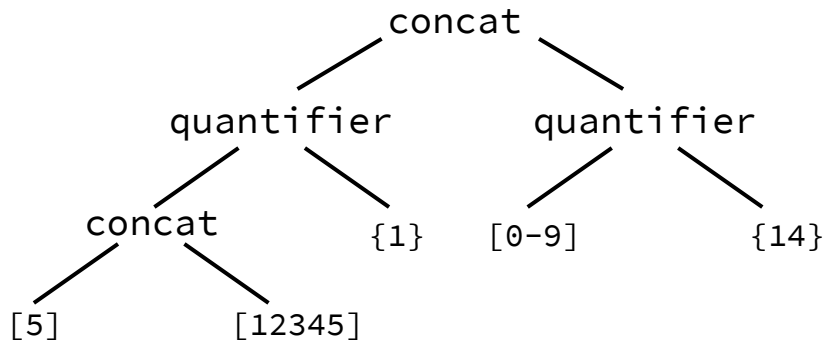
ASTs of Regular Expressions

A Regular Expression has a corresponding AST...

ex) [51|52|53|54|55]{2}[0-9]{14}



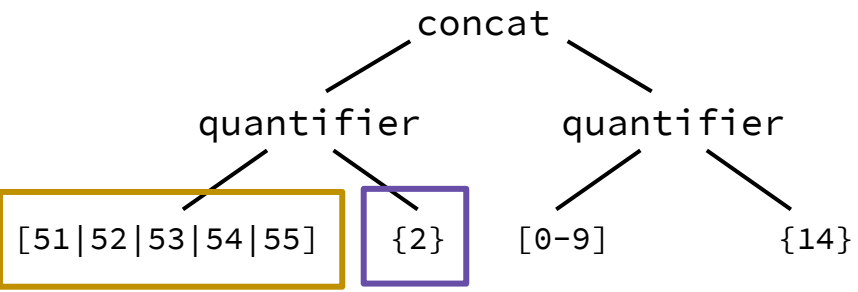
ex) (5[12345]){1}[0-9]{14}



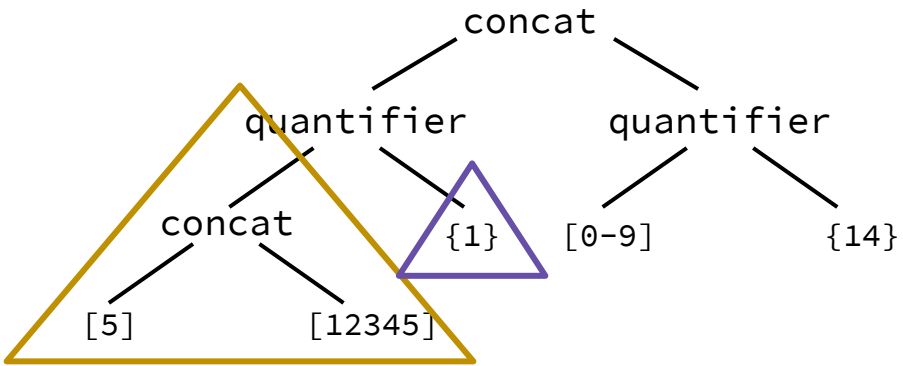
Distance between Regular Expressions

Distance = (# Nodes in original AST to be replaced) + (# Nodes replaced)

ex) [51|52|53|54|55]{2}[0-9]{14}



ex) (5[12345]){1}[0-9]{14}



Distance = (1 + 1) + (3 + 1) = 6

Templates

A **Template** is an expression in the grammar

$$t ::= [C] \mid t|t \mid tt \mid t\{m, M\} \mid t\{\triangleright, \triangleleft\} \mid \circ$$

$\circ$ (hole) - always leaf in expressions, can be replaced with any Regular expression

$\{\triangleright, \triangleleft\}$ (quantifier holes) - can be replaced with any value $\{m, M\}$

CON(t) : set of concrete regular expressions induced by filling holes in template t

Simple Template Completions

The set of *simple completions* of a template *t* ($CON_c(t)$) are completions of *t* where ○ holes are replaced with sets of characters [C]

- ex) For template t = ○{▷, ◁}([0-9]{14})...
  - [5][1]{1, 1}([0-9]{14}) ❌
  - [5]{1, 1}([0-9]{14}) ✅

Checking whether there exists a Regular Expression $r \in CON_C(t)$ that is consistent with positive and negative expressions is an NP-Complete problem.

→ Use the power of SMT solvers!!

Formalizing the Repair Problem

The *repair-from-examples problem* is as follows:

Input:

- Regular expression r to be repaired
- Finite Set of Positive examples $P \subseteq \Sigma^*$
- Finite Set of Negative examples $N \subseteq \Sigma^*$ (Note: $P \cap N = \varnothing$)

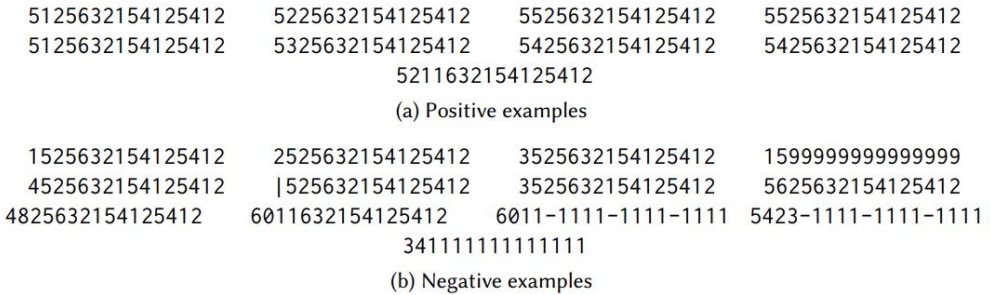
Output:

- A regular expression r' such that…
  - r' accepts all positive examples (i.e. $P \subseteq L(r')$)
  - r' rejects all negative examples (i.e. $N \cap L(r') = \varnothing$)
  - r' has minimal distance from r compared to all other repairs

Example Repair Problem

Input:

- Regular Expression
  [51|52|53|54|55]{2}[0-9]{14}
- Positive Examples
- Negative Examples



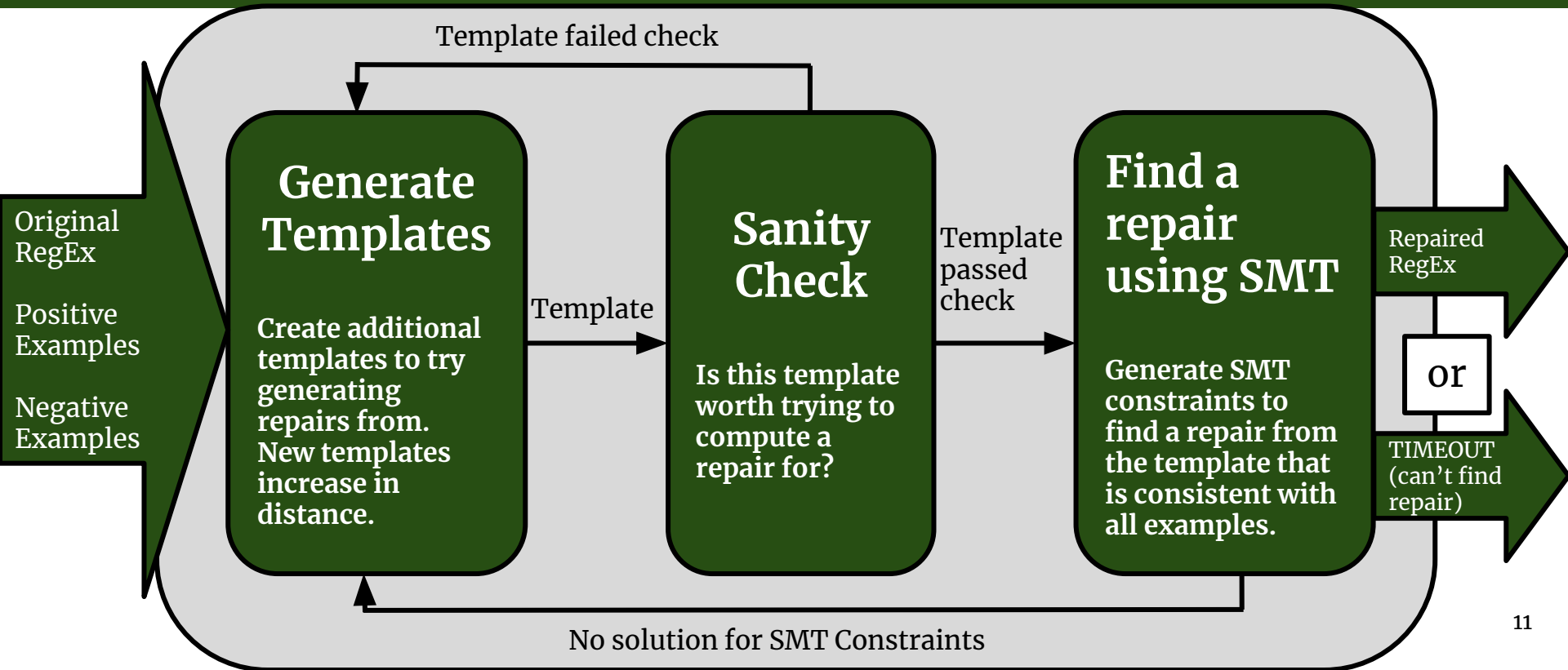
(a) Positive examples

(b) Negative examples

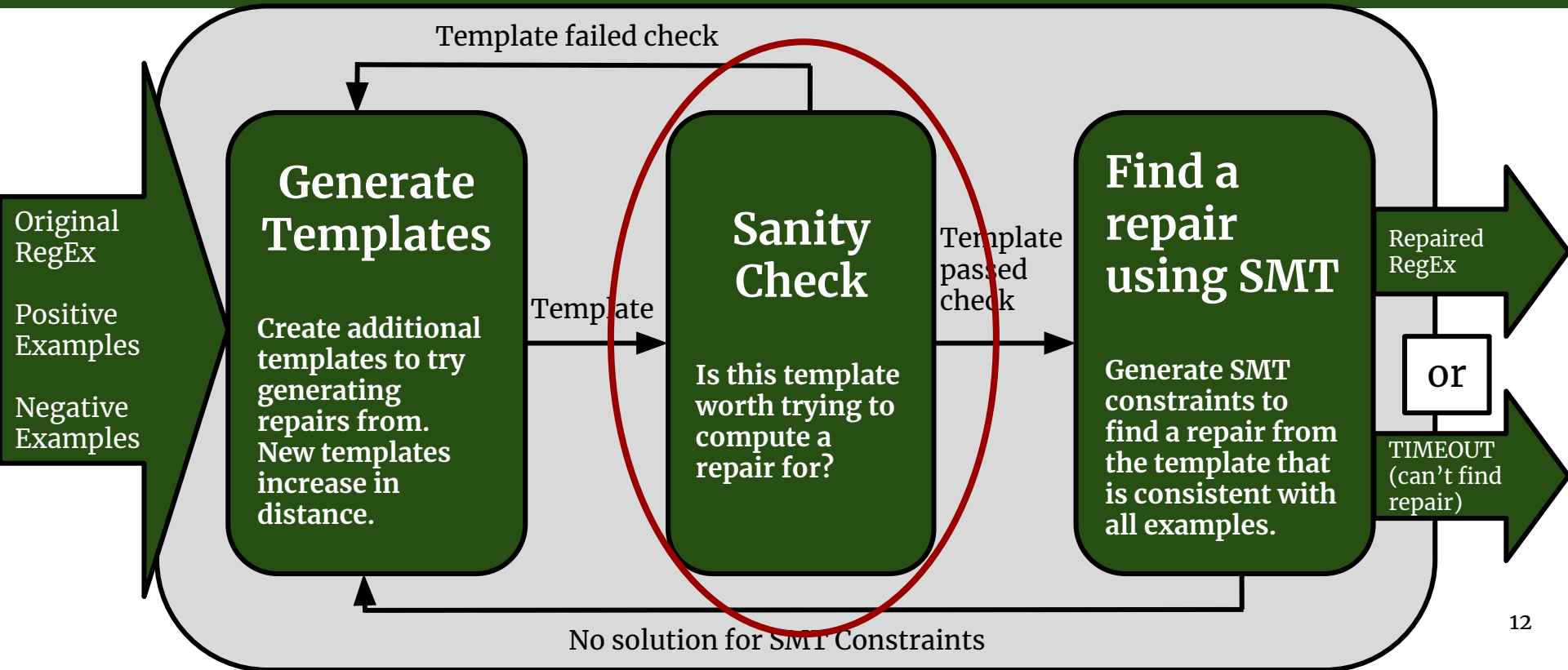
Output:

- Regular Expression
  (5[12345]){1}[0-9]{14}

RFixer Pipeline



RFixer Pipeline



Sanity Check ($t_{\perp}$ and $t_{\top}$)

- **Goal:** Filter out templates that are *guaranteed to not result into a solution*
- Regular Expressions use monotonic operators
  - $\forall r \in CON(t), L(t_{\perp}) \subseteq L(r) \subseteq L(t_{\top})$
- If $t_{\perp}$ does not reject all negative examples, then there does not exist a completion r that can reject all negative examples
- If $t_{\top}$ does not accept all positive examples, then there does not exist a completion r that can accept all positive examples

Sanity Check ($t_{\perp}$ and $t_{\top}$) [2]

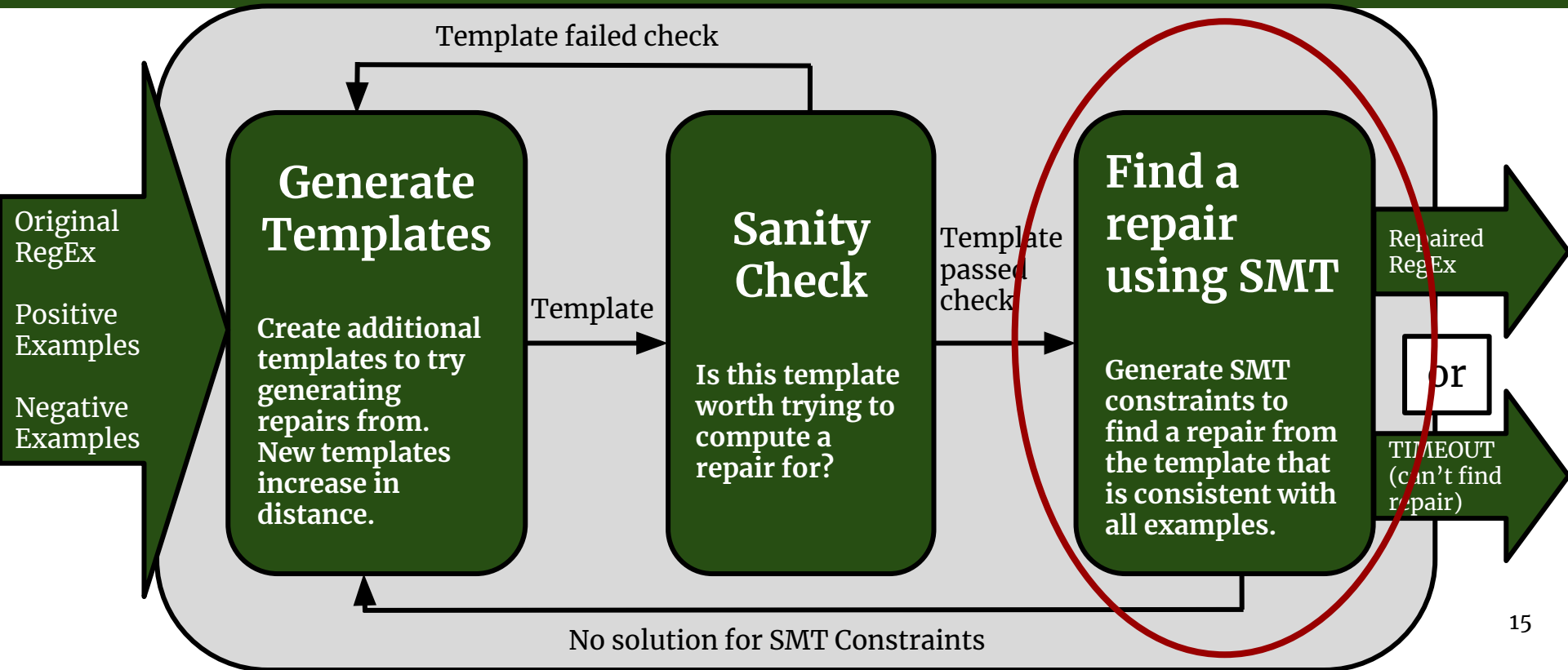
$t_{\perp}$: fill all ○ with ∅, and all {▷, ◁} with {1, 0}

- ex) For template t = ○{▷, ◁}([0-9]{14}), $t_{\perp}$ = ∅{1, 0}([0-9]{14}) = ∅

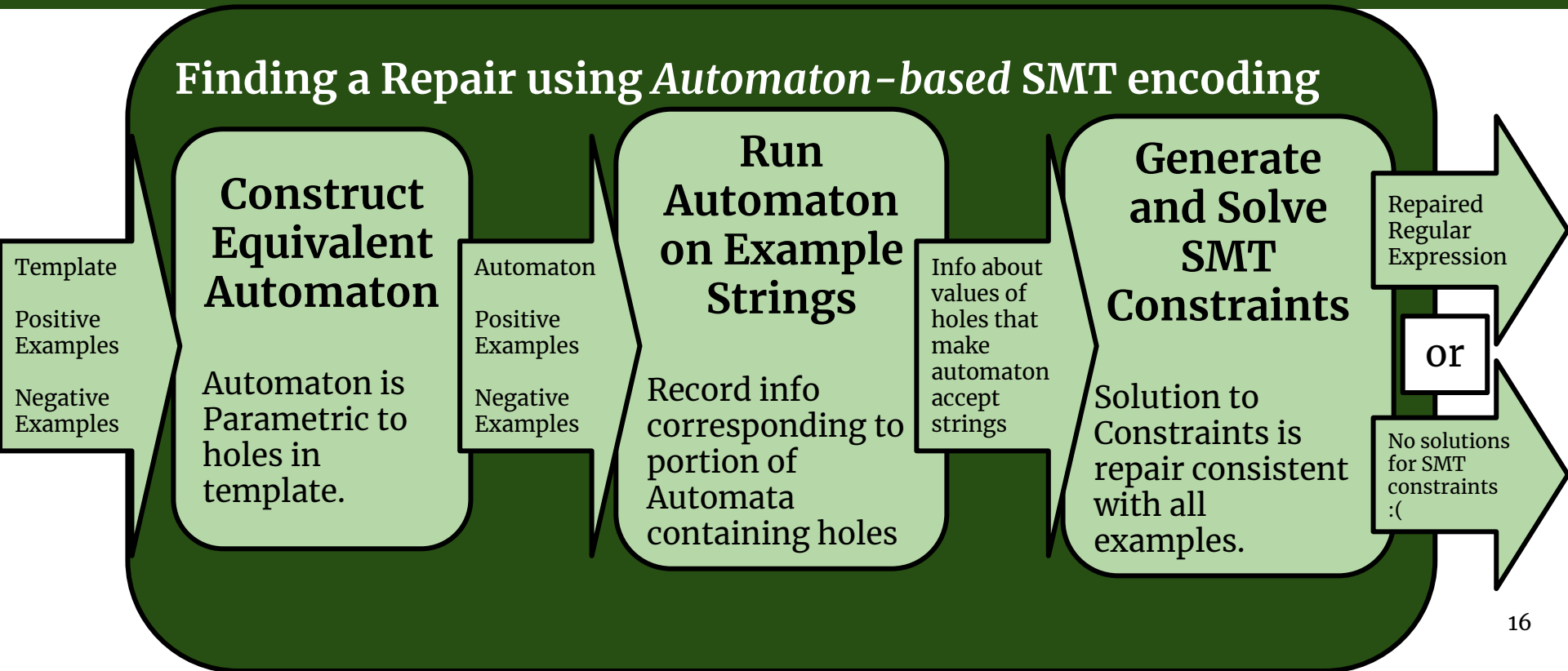
$t_{\top}$: fill all ○ with [Σ]*, and all {▷, ◁} with {0, ∞}

- ex) For template t = ○{▷, ◁}([0-9]{14}), $t_{\top}$ =[Σ]*{0, ∞}([0-9]{14})

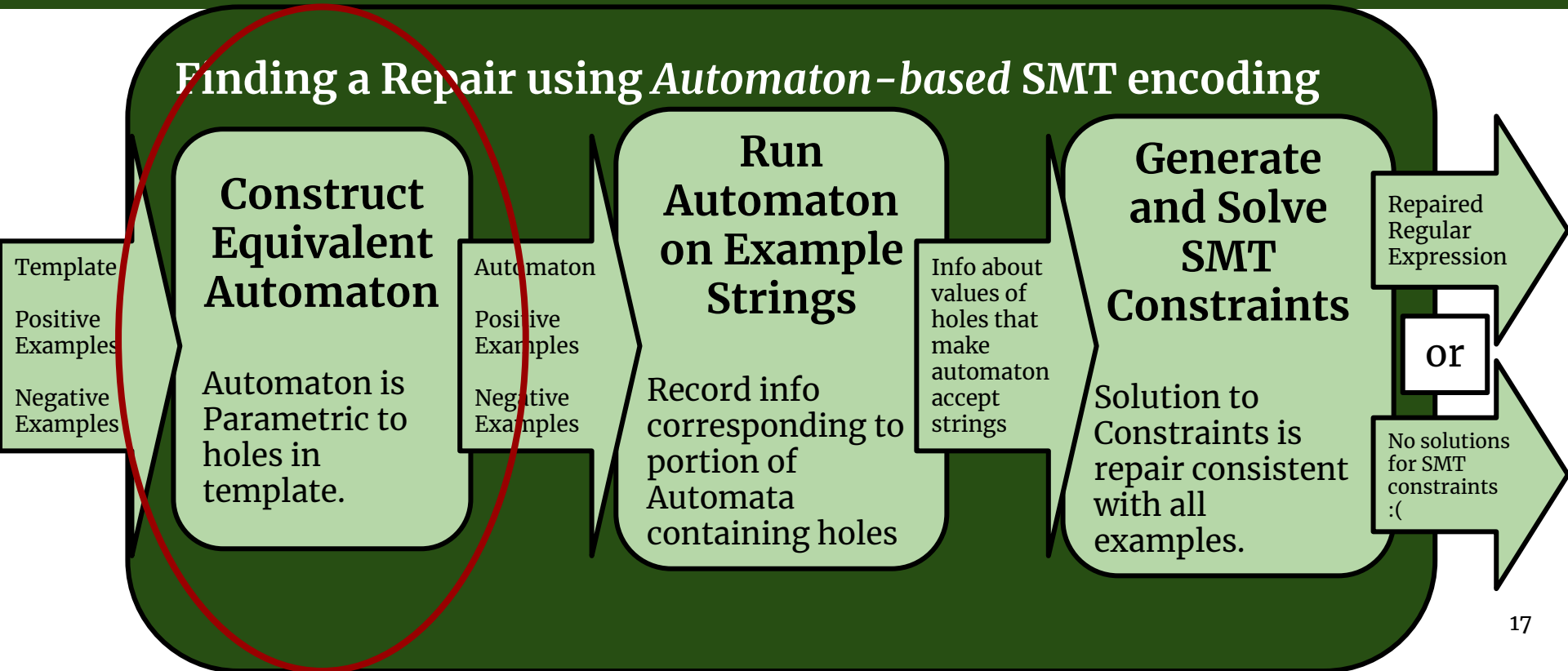
RFixer Pipeline



Zooming in… ver. 1 (Automaton-based)



Zooming in... ver. 1 (Automaton-based)



Corresponding Automaton

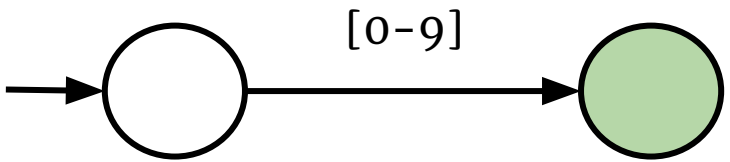
A *nondeterministic finite automaton with counters K and holes* (NFAC) is a NFA with the following properties:

- Between two states there is **at most one transition**
- Transition labels are one of the following:
  - [C]: allows any symbol in the set C
  - $\circ_i$: allows any symbol
  - ε
  - k++: ε-transition that increases value of counter *k* by 1
  - $m \leq k \leq M$ / $k \leftarrow 0$: ε-transition triggered if value of *k* is between *m* and *M*, and sets *k* to 0
  - $\triangleright \leq k \leq \triangleleft$ / $k \leftarrow 0$: ε-transition that sets *k* to 0

Corresponding Automaton Example

Let’s build an Automaton corresponding to [0-9]{14} = [0-9]{14, 14}!

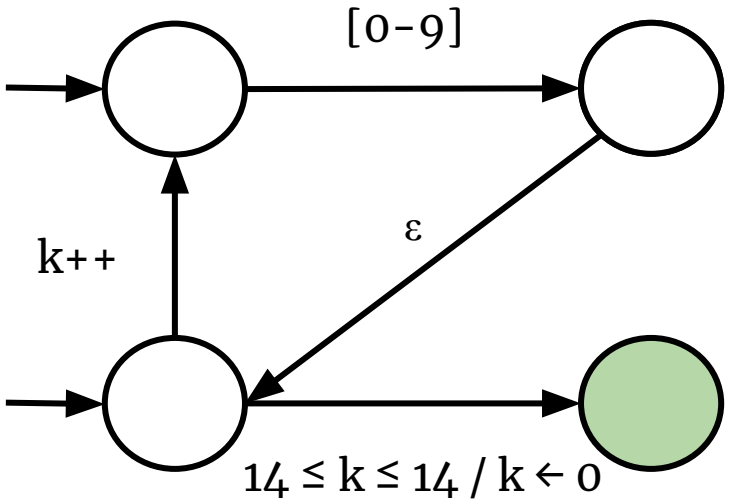
First, we can build an Automaton for [0-9]...



Corresponding Automaton Example

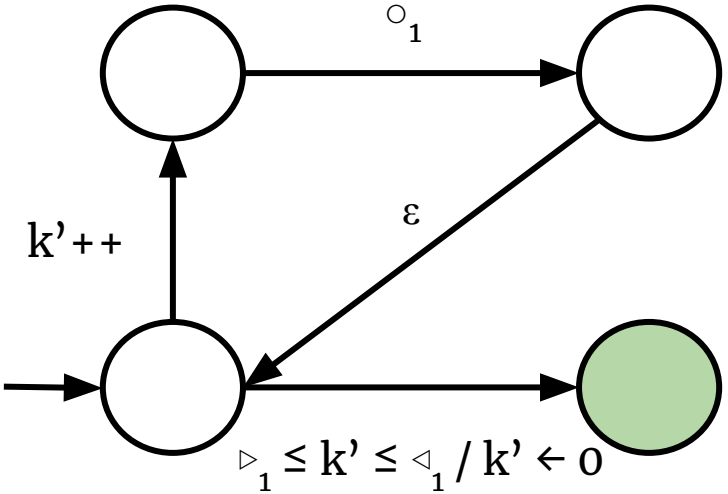
Let’s build an Automaton corresponding to [0-9]{14} = [0-9]{14, 14}!

Using this, we can build an Automaton for [0-9]{14, 14}...



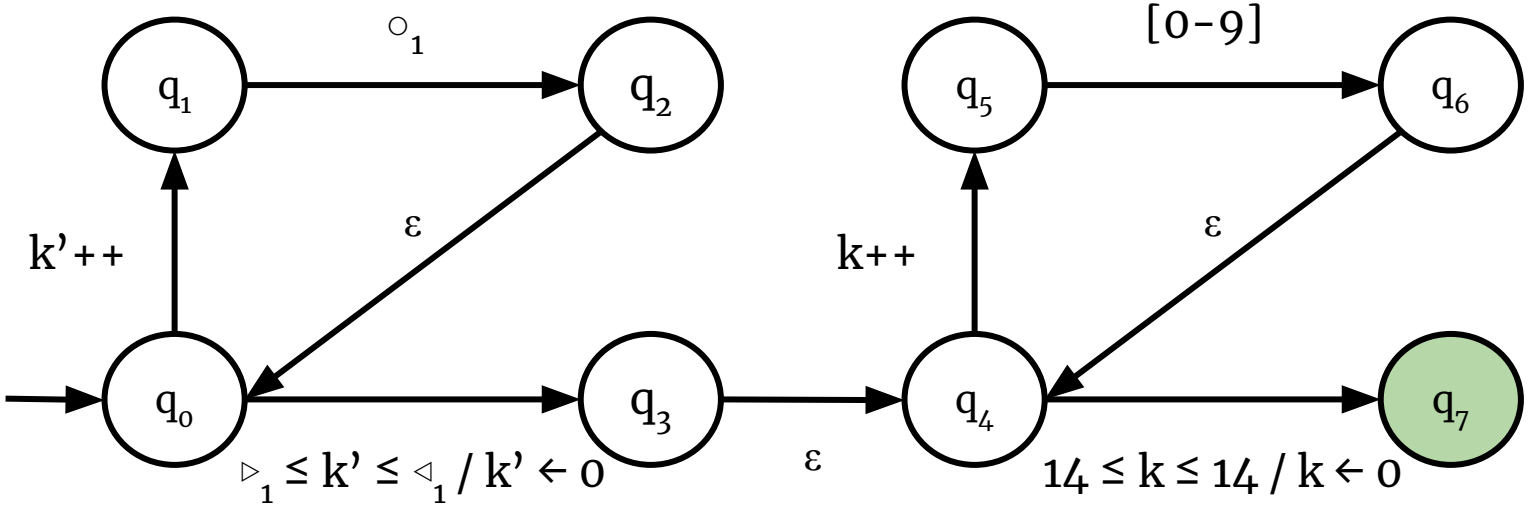
Corresponding Automaton Example

Similarly, we can construct an Automaton corresponding to $\circ_1\{\triangleright_1, \triangleleft_1\}$...

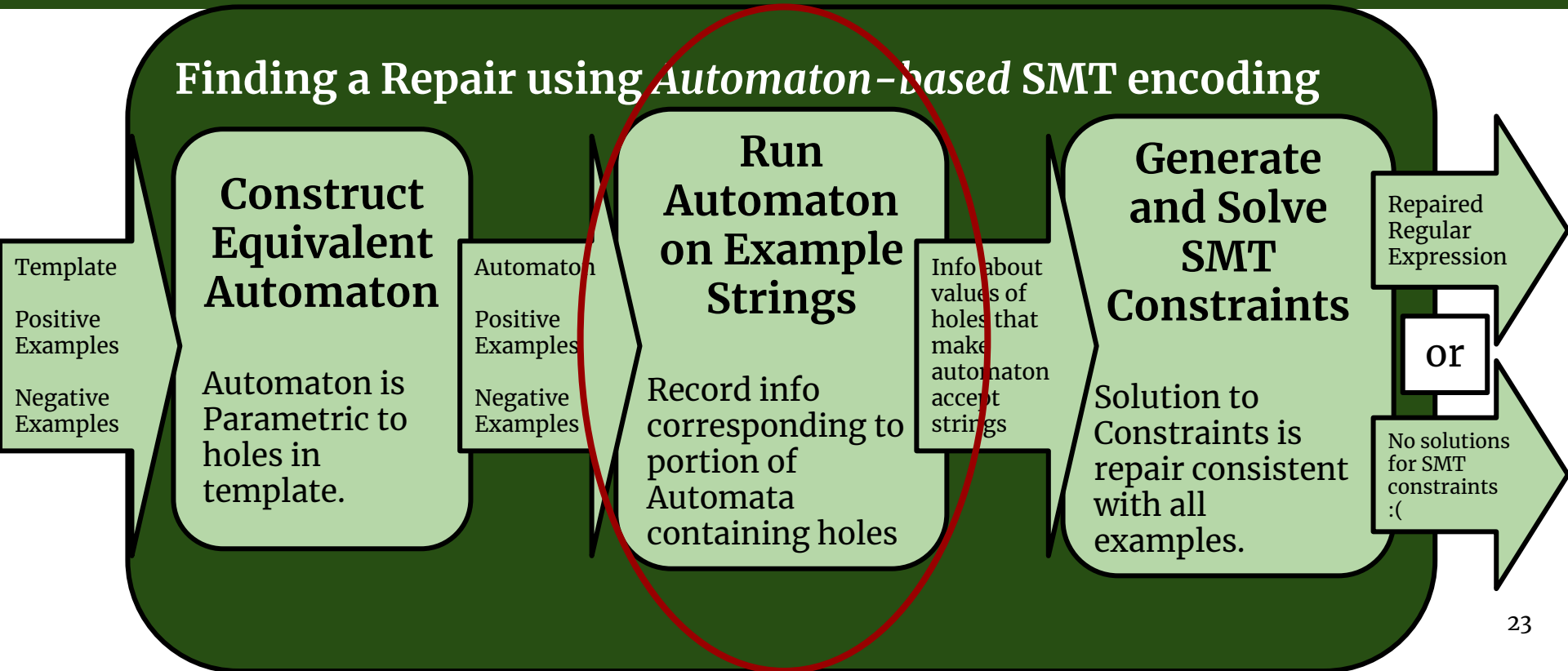


Corresponding Automaton Example

Putting these together,
we can construct an Automaton corresponding to $\circ_1\{\triangleright_1, \triangleleft_1\}[0\text{-}9]\{14\}$...



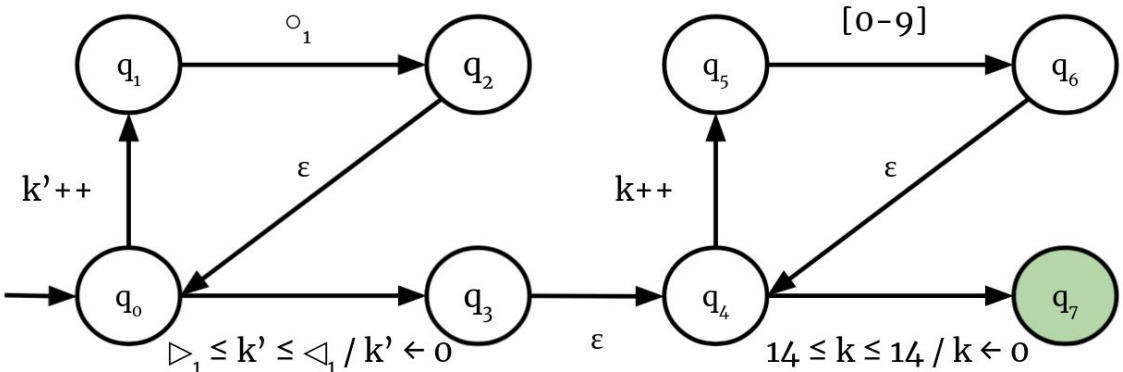
Zooming in… ver. 1 (Automaton-based)



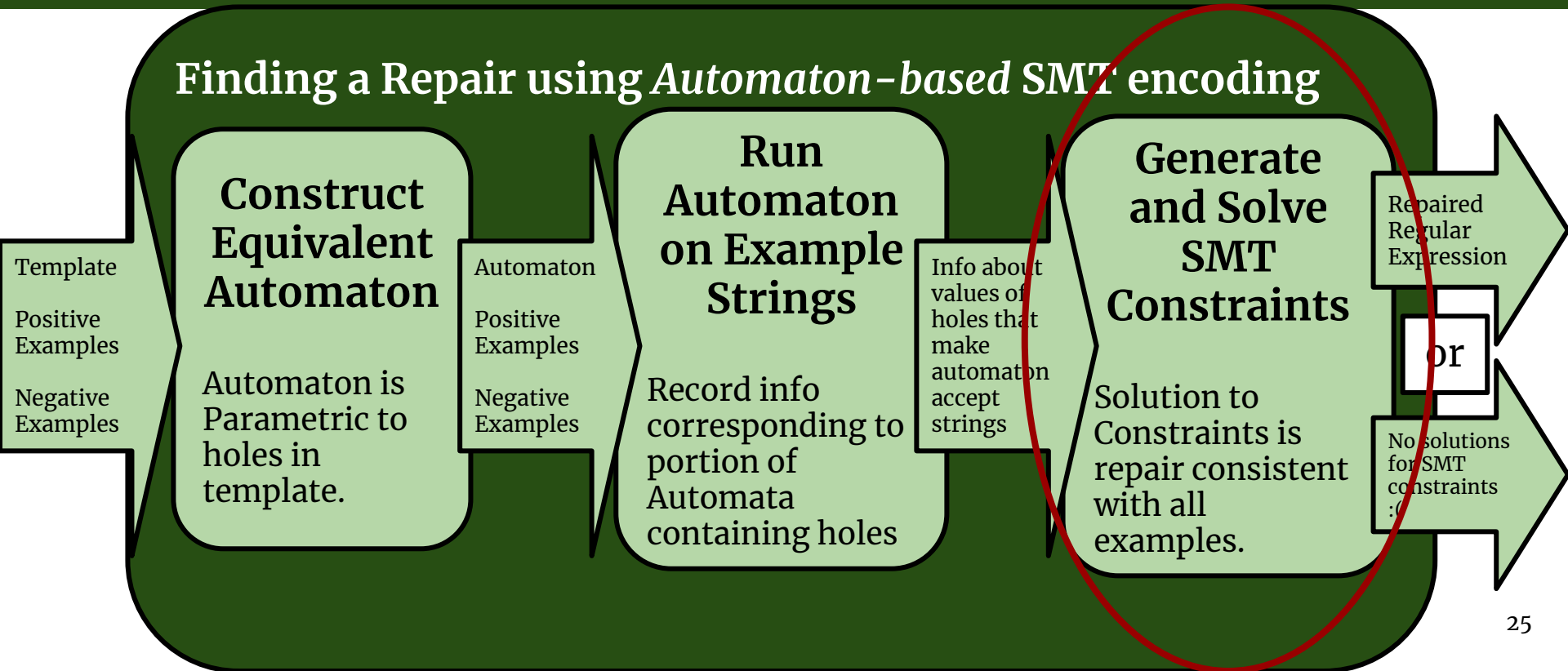
An Accepting run on a NFAC Intuition

ex) How would the string “5125632154125412” get accepted by our example NFAC?

- We know that the last 14 characters are digits
- Then, we can deduce the following:
  - $\circ_1$ needs to admit “5” and “1”
  - k’ == 2 by the time we reach the check. Thus, $\triangleright_1 \leq 2 \leq \triangleleft_1$ needs to be true



Zooming in... ver. 1 (Automaton-based)

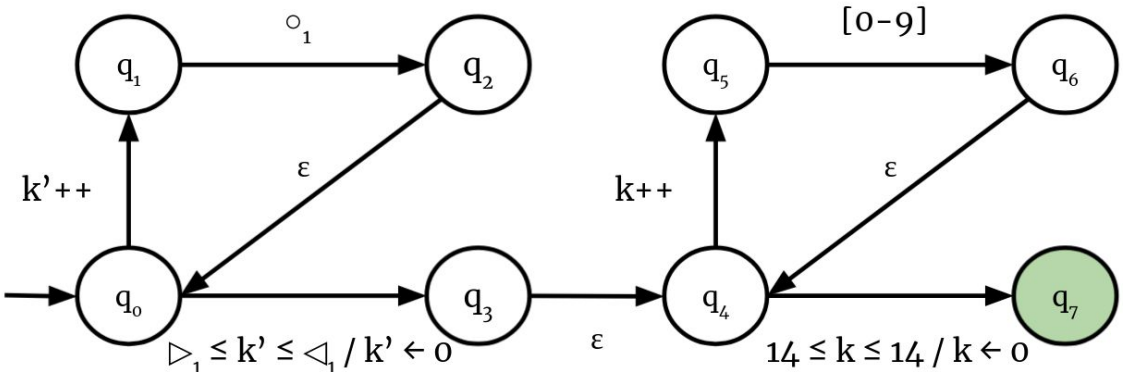


Building Constraints Intuition

From earlier, we learned that for “5125632154125412” to be accepted,

- ○$_1$ needs to admit “5” and “1”
- k’ == 2 by the time we reach the check. Thus, ▷$_1$ ≤ 2 ≤ ◁$_1$ needs to be true

Since “5125632154125412” is a positive example, we want the above to be true.



Constraints Example

How can we formalize this?

$\circ_1$ needs to admit “5” and “1”

- $\varphi_{\circ 1} = x^5_{\circ 1} \wedge x^1_{\circ 1}$
- Boolean variable $x^5_{\circ}$ says that $\circ$ can be completed with 5 if true in the solution

k’ == 2 by the time we reach the check. Thus, $\triangleright_1 \leq 2 \leq \triangleleft_1$ needs to be true

- $\varphi_{\triangleright 1, \triangleleft 1} = [\, x_{\triangleright 1} \leq 2 \,] \wedge [\, 2 \leq x_{\triangleleft 1} \,]$
- $x_{\triangleright 1}$, $x_{\triangleleft 1}$ are positive integer values that will be set by the solution as valuations of the holes

SMT Constraint Building – Run ρ

The constraint for a run ρ deals with all of the holes!

$$\Phi_\rho \equiv \bigwedge_{\circ \in H_\circ} \varphi_\circ \wedge \bigwedge_{(\triangleright, \triangleleft) \in H_{\triangleright\triangleleft}} \varphi_{\triangleright, \triangleleft}$$

SMT Constraint Builing – Consistency w Examples

A string s is accepted if it has at least one successful run

$$\Phi_s \equiv \bigvee_{\rho \in \text{RUNS}_A(s)} \Phi_\rho$$

All runs of the Automaton A on string s

The solution should be consistent with all positive and negative examples

$$\Phi^A_{P,N} \equiv \bigwedge_{s \in P} \Phi_s \wedge \bigwedge_{s \in N} \neg \Phi_s$$

All positive examples are accepted

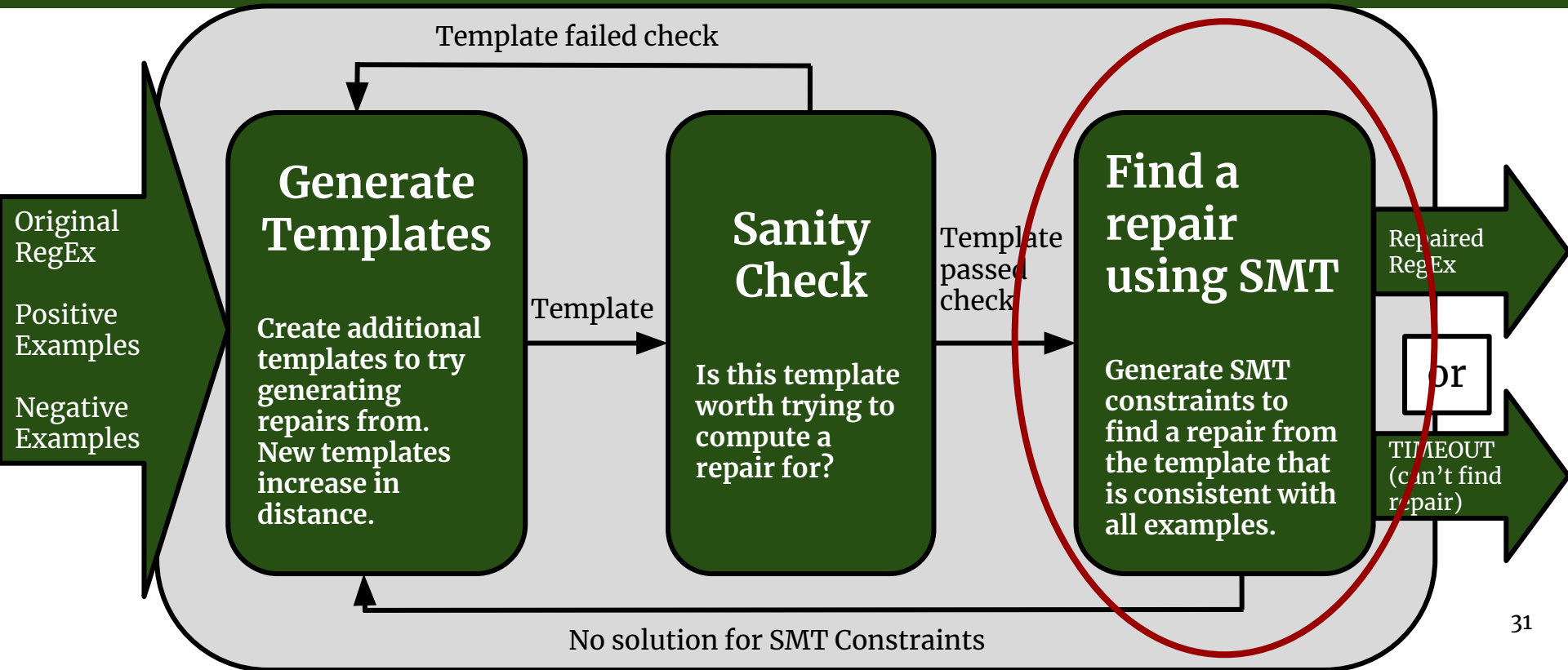
All negative examples are rejected

From Solution to Regular Expression

Given a solution τ to the constraints, the repaired Regular Expression can be constructed...

- Hole $\circ_i$ will be filled with $[C_i]$ s.t. $C_i = \{ a \mid \tau(x^a{}_{\circ i}) = \text{true} \}$
- Holes $\triangleright_i$, $\triangleleft_i$ will be respectively filled with $x_{\triangleright i}$ and $x_{\triangleleft i}$

RFixer Pipeline



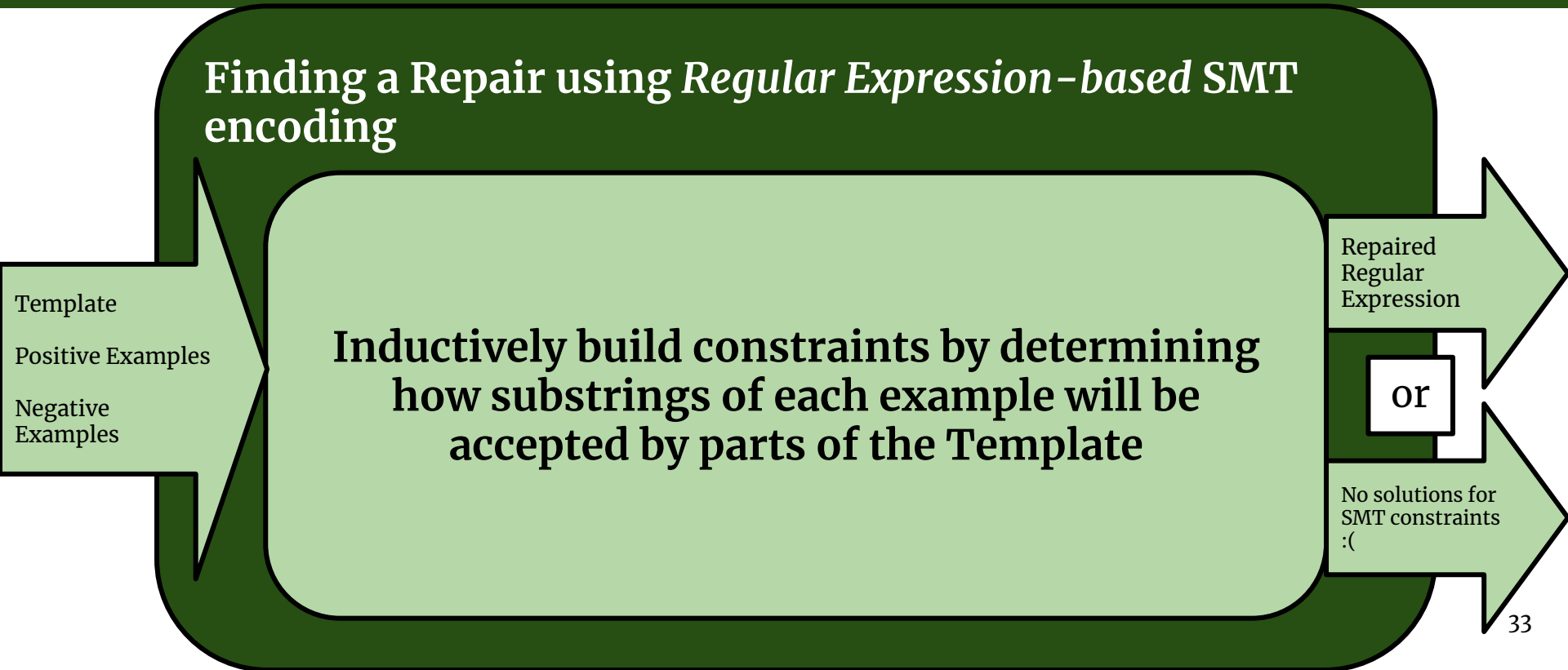
Regular Expression-based SMT Encoding

Problem: Automaton-based SMT Encoding can be impractical for expressions for which strings can have many accepting runs :(

Regular Expression-based SMT Encoding:

- ❏ Directly use the semantics of Regular Expressions to decide whether a string is accepted!

Zooming in... ver. 2 (RegEx-based)



Regular Expression-based SMT Encoding

Boolean variable t(s[i, j]):

Given a template t, a valuation τ assigning values to all holes, a string $s = a_o \dots a_n$, and positions in the string $i, j \in n$...

$$\mathrm{t(s[i, j])} = \top \qquad \textit{iff.} \qquad a_i \dots a_j \in L(t_\tau)$$

i.e. Does the expression obtained by replacing the holes with corresponding variable values accept the substring of s that starts at index i and ends at index j?

The string *s* is accepted by *t* if and only if t(s[0, n]) = ⊤!

Defining $t(s[i, j])$... Base Cases

Intuition: Break both the string and the template into atomic cases, then check

Base Cases:

- $\varnothing(s[i, j]) = \bot$
- $\varepsilon(s[i, j]) = \bot$
- $[C](s[i, j]) = \top$ iff. $i = j$ and $a_i \in C$
  - ex) [0-9]("123"[0, 0]) = $\top$
  - ex) [0-9]("123"[0, 1]) = $\bot$
  - ex) [0-9]("abc"[0, 0]) = $\bot$
- $\circ(s[i, j]) = x^{ai}_{\circ} \wedge i = j$
  - The string a_i is in the language of $\circ$ iff. the character a_i is included in the solution for the hole

Defining $t(s[i, j])$... Alternation & Concatenation

Alternation: if $t = t_1 \mid t_2$ ...

$$t(s[i,j]) = t_1(s[i,j]) \vee t_2(s[i,j])$$

Concatenation: if $t = t_1\ t_2$ ...

$$t(s[i,j]) = [\bigvee_{i \le k < j} t_1(s[i,k]) \wedge t_2(s[k+1,j])] \vee [\text{EPS}_{t_1} \wedge t_2(s[i,j])] \vee [t_1(s[i,j]) \wedge \text{EPS}_{t_2}]$$

Explore all possible nonzero length partitions of s

t_1 accepts ε and t_2 accepts s

t_1 accepts s and t_2 accepts ε

Defining *t(s[i, j])*... Quantifiers

If $t = t'\{m, M\}$ then...

Define variable $t'\{c\}(s[i, j])$: is s[i, j] accepted by c repeats of t'?

- $(t'\{1\})(s[i, j]) = t'(s[i, j])$

- For every $2 \le c \le min(n, M)$:

$$(t'\{c\})(s[i,j]) = \bigvee_{i \le k < j} (t'\{c-1\})(s[i,k]) \wedge t'(s[k+1,j])$$

Would *s[i, k]* be accepted by *c-1* repeats of *t'*?

Would *s[k+1, j]* be accepted by *t'*?

Defining $t(s[i, j])$... Concrete Quantifiers

If $t = t'\{m, M\}$ then...

- If t' does not accept ε

$$t(s[i,j]) = \bigvee_{min(n,m) \le c \le min(n,M)} (t'\{c\})(s[i,j])$$

Need to check that for some value c between m and M,
s[i, j] is accepted by t' repeated c times

- If t' accepts ε

$$t(s[i,j]) = \bigvee_{1 \le c \le min(n,M)} (t'\{c\})(s[i,j])$$

Minimum Requirement becomes trivial

Defining $t(s[i, j])$... Quantifier Holes, cont.

If $t = t'\{\triangleright, \triangleleft\}$ then...

- If t' does not accept ε

$$t(s[i,j]) = \bigvee_{1 \le c \le j-i+1} x_{\triangleright} \le c \le x_{\triangleleft} \wedge (t'\{c\})(s[i,j])$$

Set constraint on $x_{\triangleright}$ and $x_{\triangleleft}$

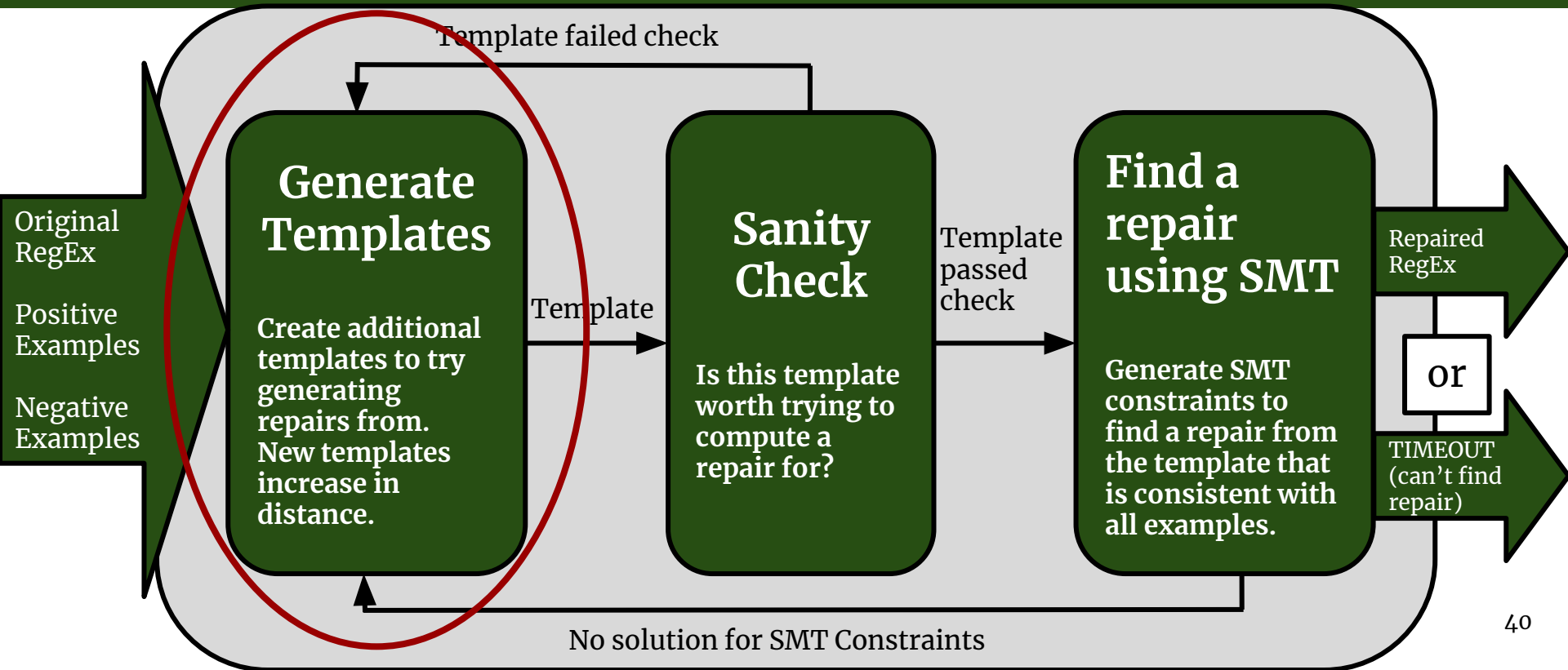
...such that s[i, j] is accepted by t' repeated c times for some value c between $x_{\triangleright}$ and $x_{\triangleleft}$

- If t' accepts ε

$$t(s[i,j]) = \bigvee_{1 \le c \le j-i+1} c \le x_{\triangleleft} \wedge (t'\{c\})(s[i,j])$$

No constraint on $x_{\triangleright}$

RFixer Pipeline



Template Generation

If we can't find a simple completion for template *t*, use *t* to generate more templates!

Three operations:

- Add a new hole
  - Replace a character class [C] with ○
  - Replace a quantifier {m, M} with {▷, ◁}
- Expand an existing hole
  - Replace ○ with $○_1○_2$, $○_1|○_2$ or $○_1\{▷_1, ◁_1\}$
- Reduce an existing hole ○ by replacing its parent node with a hole

Template Generation Example

Initial Templates for *r* = [51|52|53|54|55]{2}[0-9]{14} = $[C_1]\{2\}[C_2]\{14\}$ are created by **Adding Holes**...

$\circ\{2\}[C_2]\{14\}$ $[C_1]\{\triangleright, \triangleleft\}[C_2]\{14\}$ $[C_1]\{2\}\circ\{14\}$ $[C_1]\{2\}[C_2]\{\triangleright, \triangleleft\}$

Expanding the hole in $\circ\{2\}[C_2]\{14\}$ gives us new templates...

$\circ\circ\{2\}[C_2]\{14\}$ $(\circ|\circ)\{2\}[C_2]\{14\}$ $\circ\{\triangleright, \triangleleft\}[C_2]\{14\}$

Eventually, RFixer reaches template *t* = $\circ_1\circ_2\{\triangleright, \triangleleft\}[C_2]\{14\}$, and finds a simple completion [5][12345]{1, 1}$[C_2]\{14\}$!!

Generalizing Solutions using MaxSMT

- Minimal Distance isn't the only criterion for a good repair!

ex) Template ○{▷, ◁}, and some set of positive and negative examples

Repair #1 = [0-9]{0, ∞} = $[0\text{-}9]^*$

Repair #2 = [0236]{2, 8}

RFixer will see these repairs as equal...

Use MaxSMT Optimization Objectives (SMT formulas with a minimization objective over a set of weights) to produce "more general" repairs!

※ We assume that we have already found an SMT Solution by this point

Generalizing Solutions using MaxSMT [2]

Encodings for character classes that represent common sets of characters

ex) To add constraint for hole ○ and general character class [0-9], add...

- The variable $x^{[0\text{-}9]}_{\circ}$
- The constraint $x^{[0\text{-}9]}_{\circ} \rightarrow x^{0}_{\circ} \wedge x^{1}_{\circ} \wedge \ldots \wedge x^{9}_{\circ}$
- Add weights such that *the general character class will be favored over multiple single characters*
  - Weight of character class [0-9] = 3
  - Weight of a single digit $d \in$ [0-9] = 2

Similar mechanism for [a-z], [A-Z], and quantifiers (?, *, +)

Evaluation Details

- RFixer implemented in Java using Z3 ver 4.6.2 as SMT solver

- RFixer-a: Automaton-based SMT Encoding
- RFixer-r: Regular Expression-based SMT Encoding
- Both run with and without the MaxSMT optimizations

Evaluation - Benchmarks

Benchmark Sources

- *Automata Tutor:* Website for helping students learn automata theory
  - Anonymized dataset of Regular Expression Assignments and student-submitted solutions
- *RegExLib:* Website collecting Regular Expressions from developers
  - Users can point out imprecise Regular Expressions and provide counterexamples
- *Rebele et al. [2018]:* Repairs Regular Expressions to accept Positive Examples
  - Use heuristics to greedily modify the input Regular Expression
  - Benchmarks: *Training Set* and *Test Set*

Evaluation - Effectiveness of RFixer

RFixer-a/r successfully repaired...

- *Automata Tutor*: 1588/2104 (75%) expressions - avg. time 9.3s
- *RegExLib*: 23/25 (92%) expressions - avg. time 7.3s
- *Rebele et al. [2018]*: 35/50 (70%) expressions - avg. time 2.4s

RFixer couldn't solve benchmarks that

- required very large fixes
- involved complex nested quantifiers (SMT encodings and $t_{\top}$ test slow)

Evaluation - Quality of Repairs

Rebele et al. [2018]:

- A repair is created using the *training set*, and evaluated on the *test set*
- F1 score: 0-1 number computed as avg. of precision and recall

Average: RFixer's repairs improved F1 scores by 0.16 (113% improvement)

F1 scores only decreased for 4 repairs

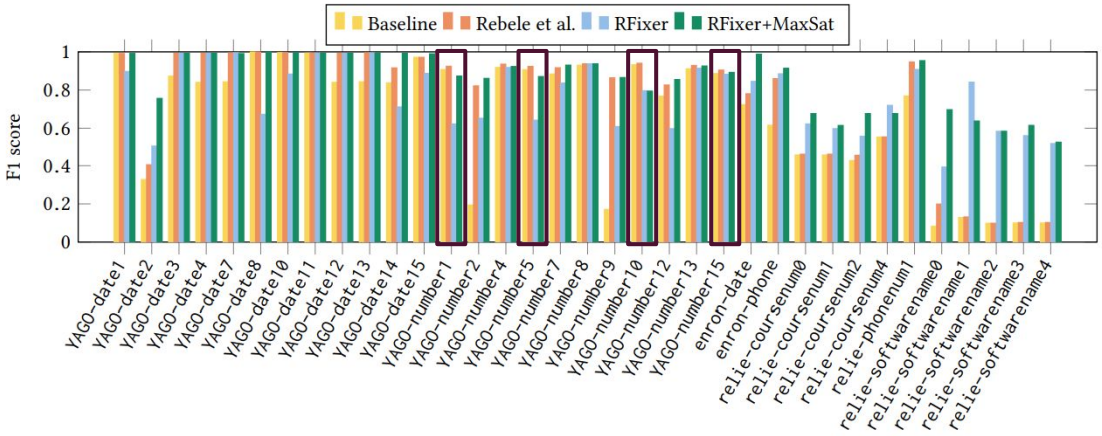
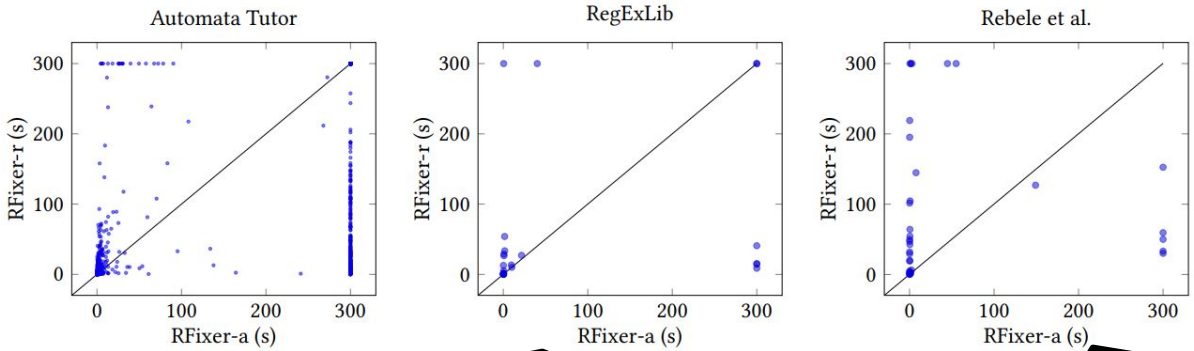


Fig. 5. F1 scores for the benchmarks from Rebele et al. [2018] for which RFixer terminated.

Evaluation - SMT Encodings Performance

Automaton-based (RFixer-a) vs. Regular Expression-based (RFixer-r)...



Average: RFixer-r is 2.0× slower
RFixer-a times out on 252 benchmarks
RFixer-r times out on 20 benchmarks

Average: RFixer-r is 4.5× slower
RFixer-a times out on 4 benchmarks
RFixer-r times out on 2 benchmarks

Average: RFixer-r is 1.75× slower
RFixer-a times out on 5 benchmarks
RFixer-r times out on 5 benchmarks

Conclusion

- RFixer: The first tool for repairing Regular Expressions from Positive and Negative Examples
  - Create templates that include character class holes ○ and quanitifier holes {▷, ◁}
  - Automaton-based SMT Encoding (RFixer-a)
  - Regular Expression-based SMT Encoding (RFixer-r)
  - MaxSMT Optimization to prioritize general repairs

- RFixer is promising in terms of performance and effectiveness of repairs, especially when compared against existing tools
- In general, RFixer-a is faster than RFixer-r

Thank you!!