Parameterized Model Counting for String and Numeric Constraints

Abdulbaki Aydin[1], **William Eiers[2]**, Lucas Bang[3], Tegan Brennan[2], Miroslav Gavrilov[2], Tevfik Bultan[2], Fang Yu[4]

[1] Microsoft

[2] University of California Santa Barbara

[3] Harvey Mudd College

[4] National Chengchi University, Taiwan

Quantitative program analysis

Given a program, quantitative program analysis can determine:

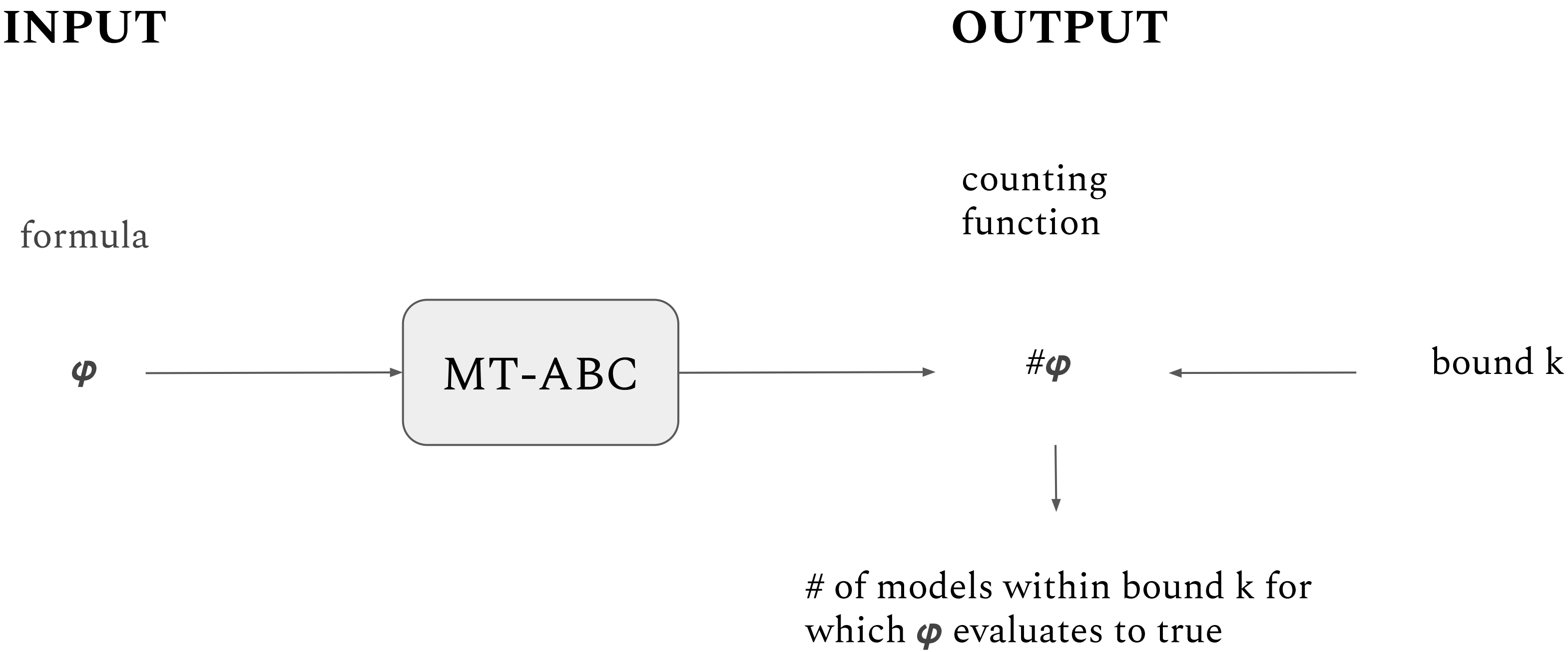
- Probability of program behaviors
- Number of inputs that cause an error
- Amount of information leakage

Quantitative program analysis requires model counting:

- Counting the number of satisfying solutions (models) for a given constraint

Our tool MT-ABC is the ***most expressive*** *model-counting constraint solver!*

MT-ABC: Model counting constraint solver



MT-ABC: Expressive constraint language

- Language agnostic, supports SMT2Lib format
- Supports string and numeric constraints and their combinations

$$
\begin{aligned}
\varphi &\longrightarrow \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \varphi_{\mathbb{Z}} \mid \varphi_{\mathbb{S}} \mid \top \mid \bot \\
\varphi_{\mathbb{Z}} &\longrightarrow \beta = \beta \mid \beta < \beta \mid \beta > \beta \\
\varphi_{\mathbb{S}} &\longrightarrow \gamma = \gamma \mid \gamma < \gamma \mid \gamma > \gamma \mid \mathrm{match}(\gamma, \rho) \mid \mathrm{contains}(\gamma, \gamma) \mid \mathrm{begins}(\gamma, \gamma) \mid \mathrm{ends}(\gamma, \gamma) \\
\beta &\longrightarrow v_i \mid n \mid \beta + \beta \mid \beta - \beta \mid \beta \times n \\
&\qquad \mid \mathrm{length}(\gamma) \mid \mathrm{toint}(\gamma) \mid \mathrm{indexof}(\gamma, \gamma) \mid \mathrm{lastindexof}(\gamma, \gamma) \\
\gamma &\longrightarrow v_s \mid \rho \mid \gamma \cdot \gamma \mid \mathrm{reverse}(\gamma) \mid \mathrm{tostring}(\beta) \mid \mathrm{charat}(\gamma, \beta) \mid \mathrm{toupper}(\gamma) \mid \mathrm{tolower}(\gamma) \\
&\quad \mid \mathrm{substring}(\gamma, \beta, \beta) \mid \mathrm{replacefirst}(\gamma, \gamma, \gamma) \mid \mathrm{replacelast}(\gamma, \gamma, \gamma) \mid \mathrm{replaceall}(\gamma, \gamma, \gamma) \\
\rho &\longrightarrow \varepsilon \mid s \mid \rho \cdot \rho \mid \rho \mid \rho \mid \rho^*
\end{aligned}
$$

MT-ABC in a nutshell

Automata-based constraint solving

Why?

MT-ABC in a nutshell

Automata-based constraint solving

Basic idea:

Automata can represent sets of strings

Represent satisfying solutions for constraints as strings

Construct an automaton that accepts satisfying solutions for a given constraint

This reduces the model counting problem to path counting

Given some bound, count the number of paths in a graph

Automata-based constraint solving

Generate automaton that accepts satisfying solutions for the constraint

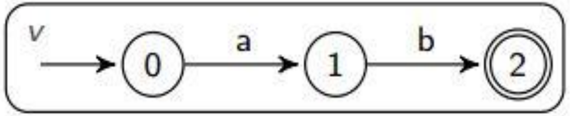
MT-ABC can handle both
string and **integer** constraints

- Constraints over only string variables (e.g., v = “abcd”)
- Constraints over only integer variables (e.g., i = 2×j)
- Constraints over both string and integer variables (e.g., length(v) = i)

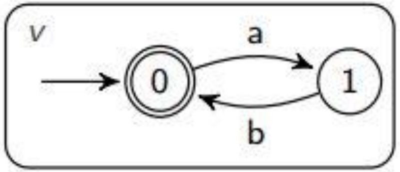
Automata-based constraint solving: Strings, ¬

Basic string constraints are directly mapped to automata

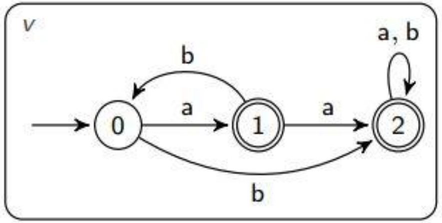
`v = “ab”`



`match(v, (ab)*)`



`¬match(v, (ab)*)`

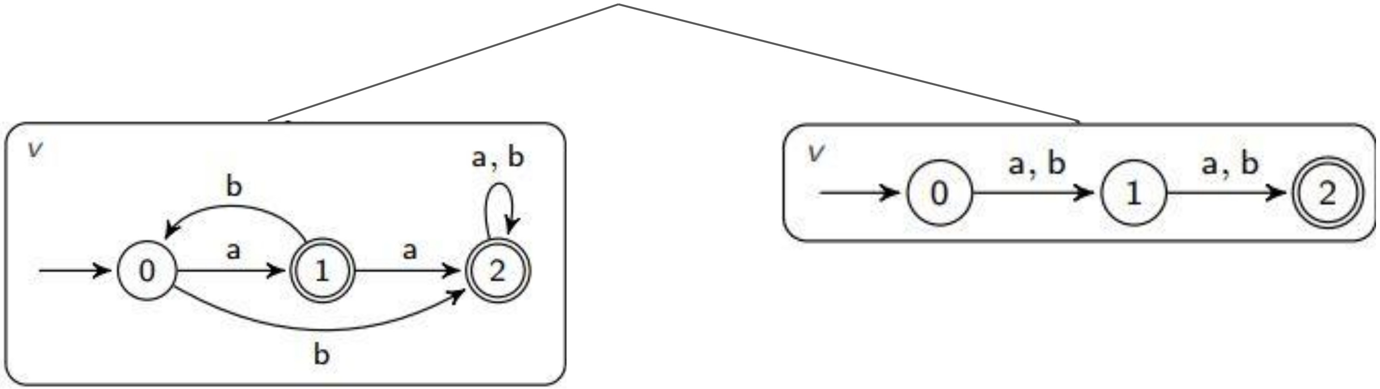


automata
complement

Automata-based constraint solving: Strings, ¬, ∧, ∨

More complex constraints are solved by creating automata for subformulae then combining their results

```
¬match(v, (ab)*) ∧ length(v) = 2
```

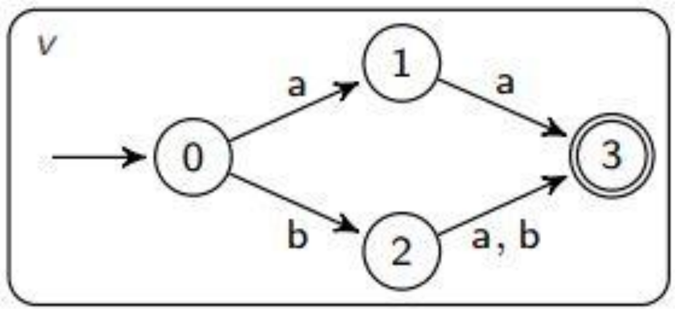


automata
product

Automata-based constraint solving: Strings, ¬, ∧, ∨

More complex constraints are solved by creating automata for subformulae then combining their results

```
¬match(v, (ab)*) ∧ length(v) = 2
```

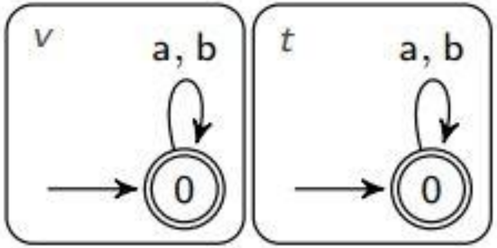


automata
product

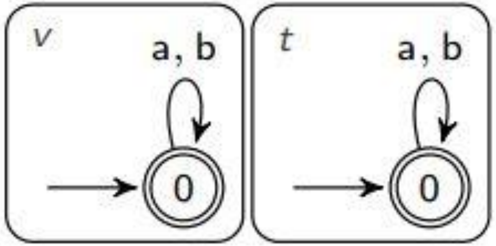
Automata-based constraint solving: Multi-variable

For multi-variable constraints, generate an automaton for each variable

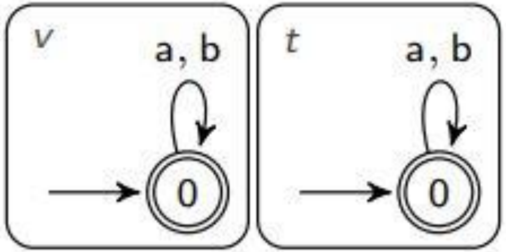
`v = t`



`v ≠ t`



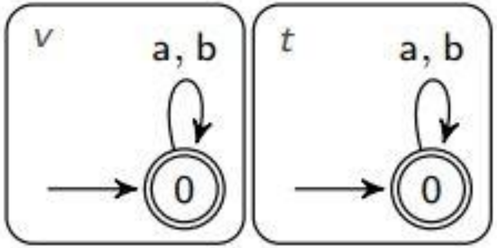
`v = t ∧ v ≠ t`



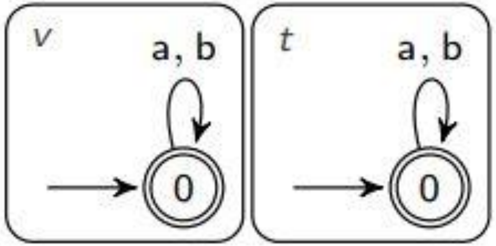
Automata-based constraint solving: Multi-variable

For multi-variable constraints, generate an automaton for each variable

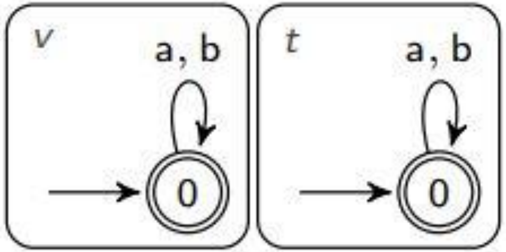
`v = t`



`v ≠ t`



`v = t ∧ v ≠ t`



Not Satisfiable!

Automata-based constraint solving: Multi-variable

Traditional string automata cannot precisely capture relational constraints

Generated automata significantly over-approximate # of satisfying solutions

Can we do better?

YES!

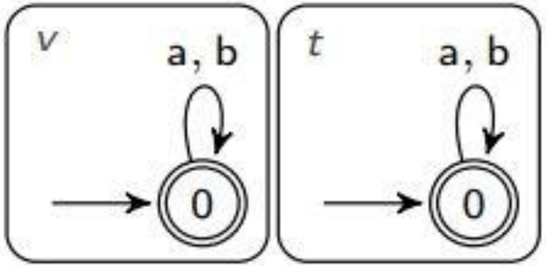
Enter Multi-track Automata...

Multi-track automata

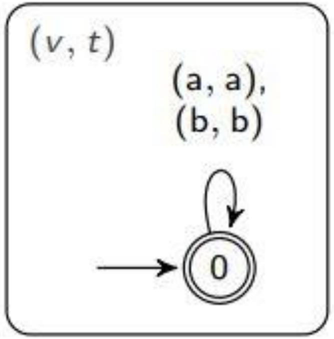
Multi-track automaton = DFA accepting tuples of strings

Each track represents the values of a single variable

`v = t`



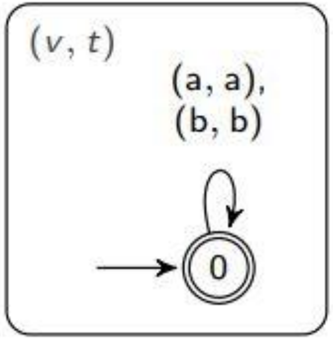
`v = t`



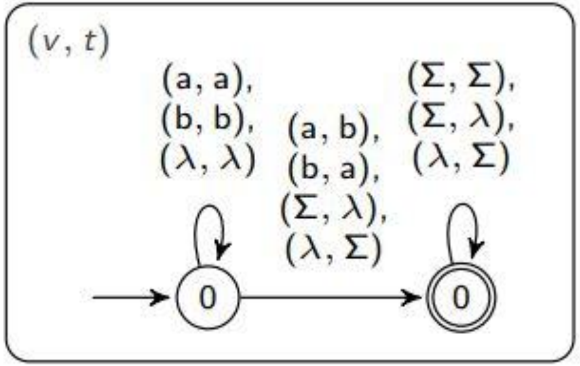
Preserves relations between variables!

Multi-track automata

`v = t`



`v ≠ t`

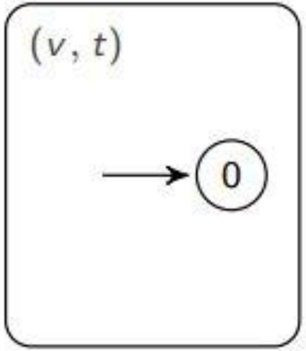


Padding symbol $\lambda \notin \Sigma$ used to align tracks of different length

- Appears at the end

automata product

`v = t ∧ v ≠ t`



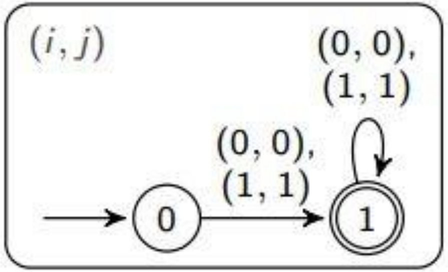
Correctly encodes unsatisfiability!

Multi-track automata

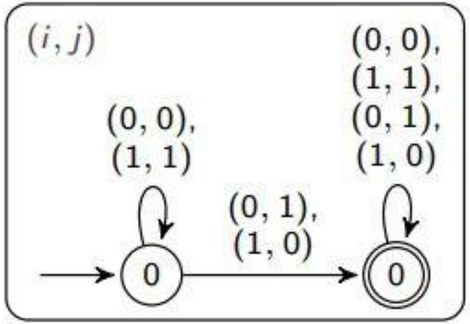
Multi-track automata can also solve numeric constraints

- Each track represents a single numeric variable
- Encoded as binary integers in 2's complement form

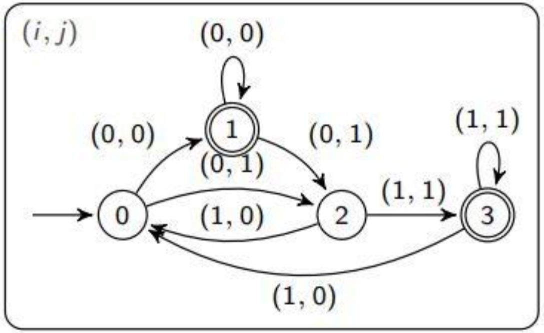
`i = j`



`i ≠ j`



`i = 2×j`

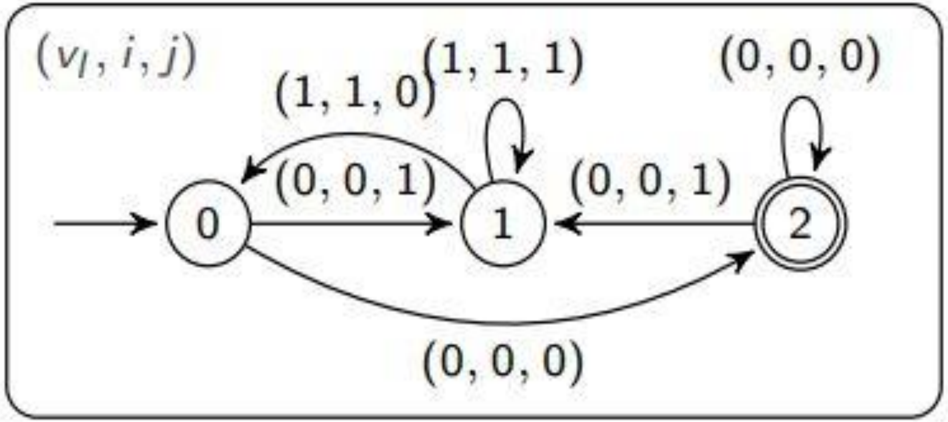


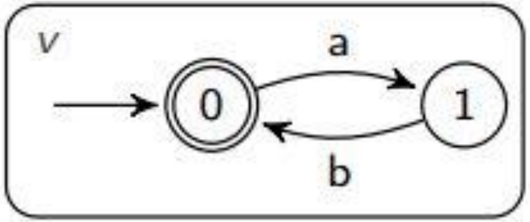
Constraint Solving: Algorithm

1. Push negations down to atomic constraints
2. Solve atomic string ($\varphi_{\mathbb{S}}$) and integer ($\varphi_{\mathbb{Z}}$) constraints
   - Initially all variables are unconstrained
3. Solve mixed constraints
4. Handle disjunctions using automata product
5. Handle conjunctions using automata product
6. If there is an over-approximation under a conjunction, solve atomic constraints that cause over-approximation again
   - This time initialize variables with the latest computed values

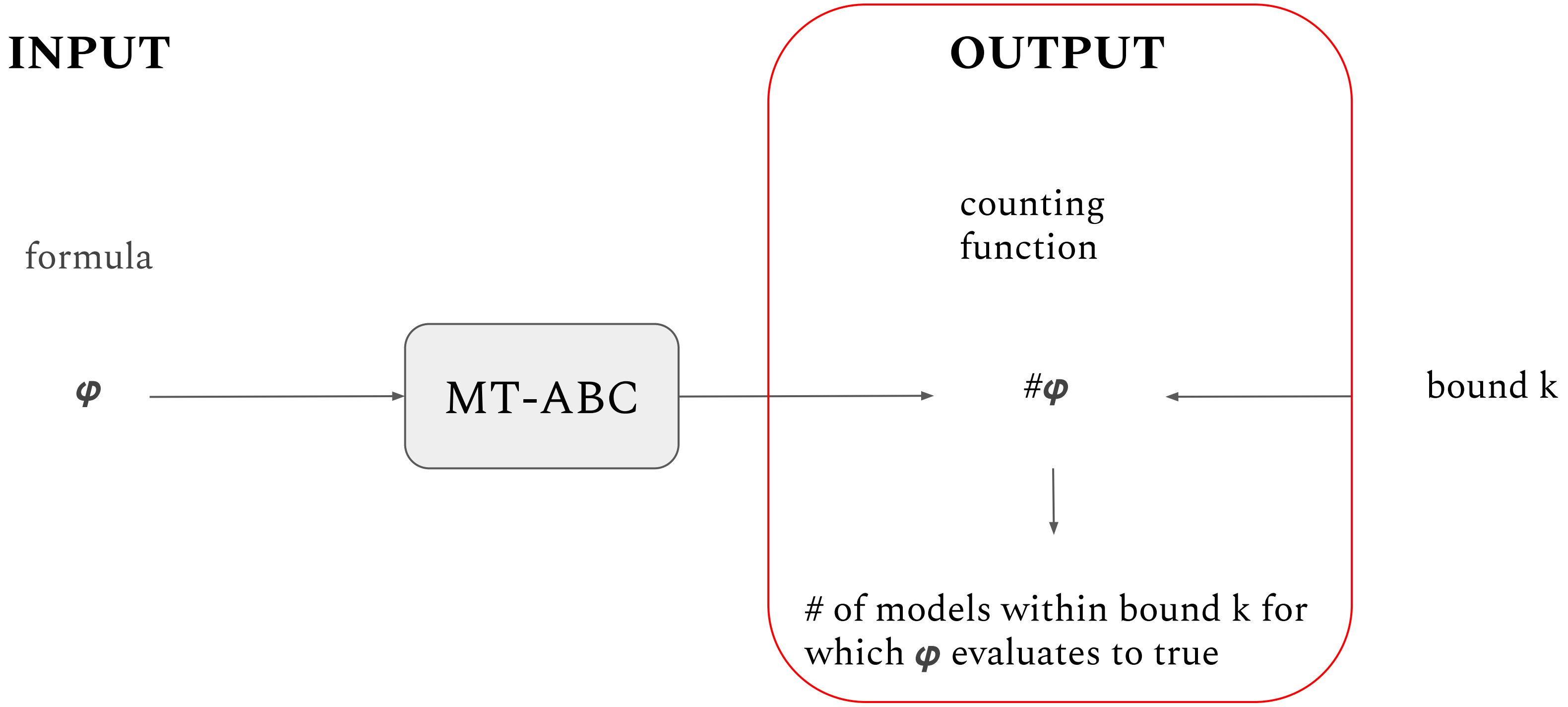
Constraint Solving: Example

```
i = 2×j ∧ length(v) = i
```





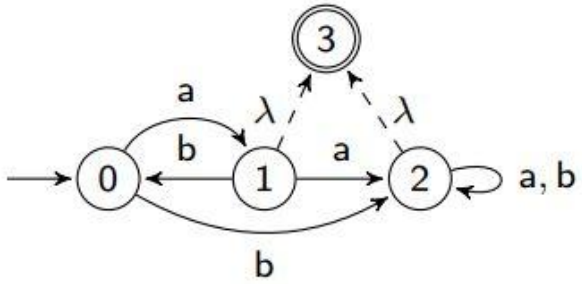
MT-ABC: Model counting constraint solver



Automata-based model counting

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

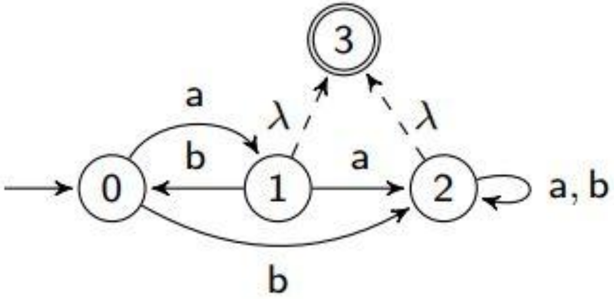
$$\varphi \equiv \neg\mathrm{match}(v, (\mathsf{ab})^*)$$



- We generate a function f(k)
  - Given a length bound k, it will count the number of accepting paths with length k

Parameterized Model Counting

$$\varphi \equiv \neg\text{match}(v, (\text{ab})^*)$$



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

$f(0) = 0$ $\quad$ $f(1) = 2$ $\quad$ $f(2) = 3$ $\quad$ $f(3) = 8$

Experimental evaluation

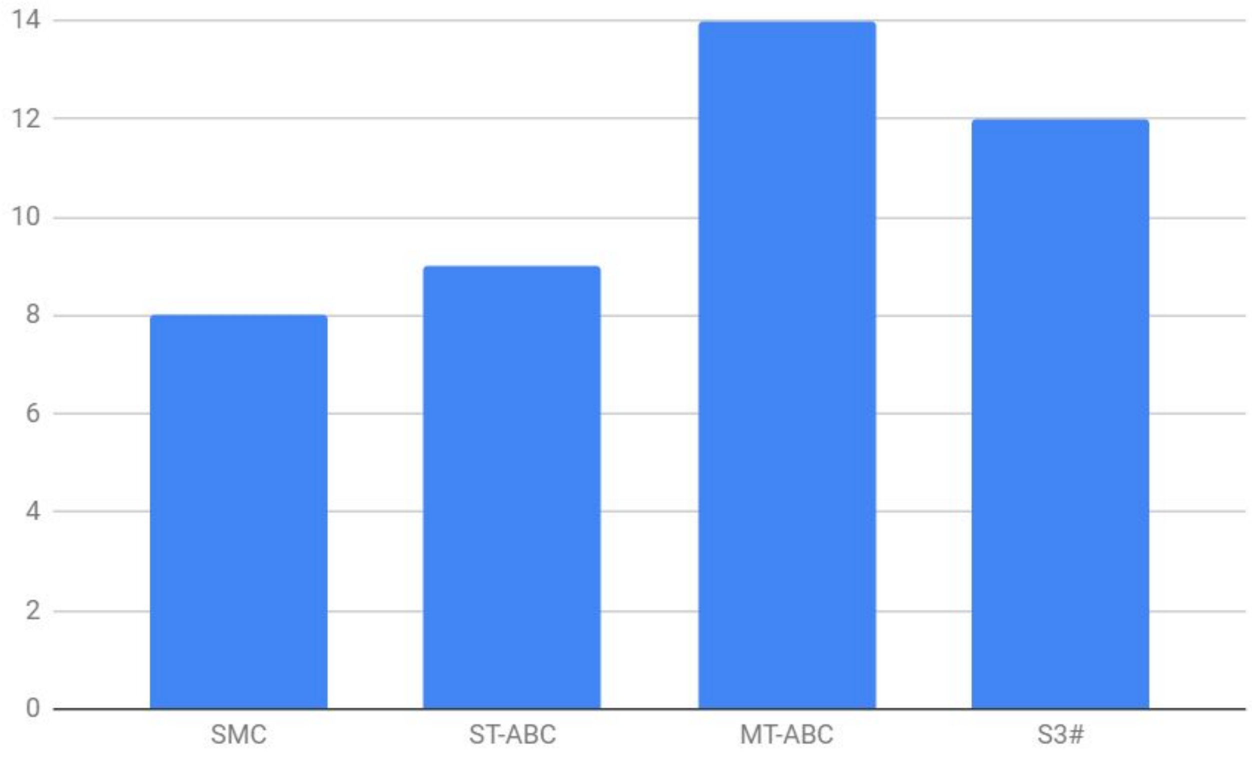
Compared MT-ABC with existing model counters on a variety of benchmarks

- S3#
  - String constraints, mixed constraints
- SMC
  - String constraints
- ST-ABC
  - String constraints
- LattE
  - Integer constraints
- SMTApproxMC
  - Integer constraints

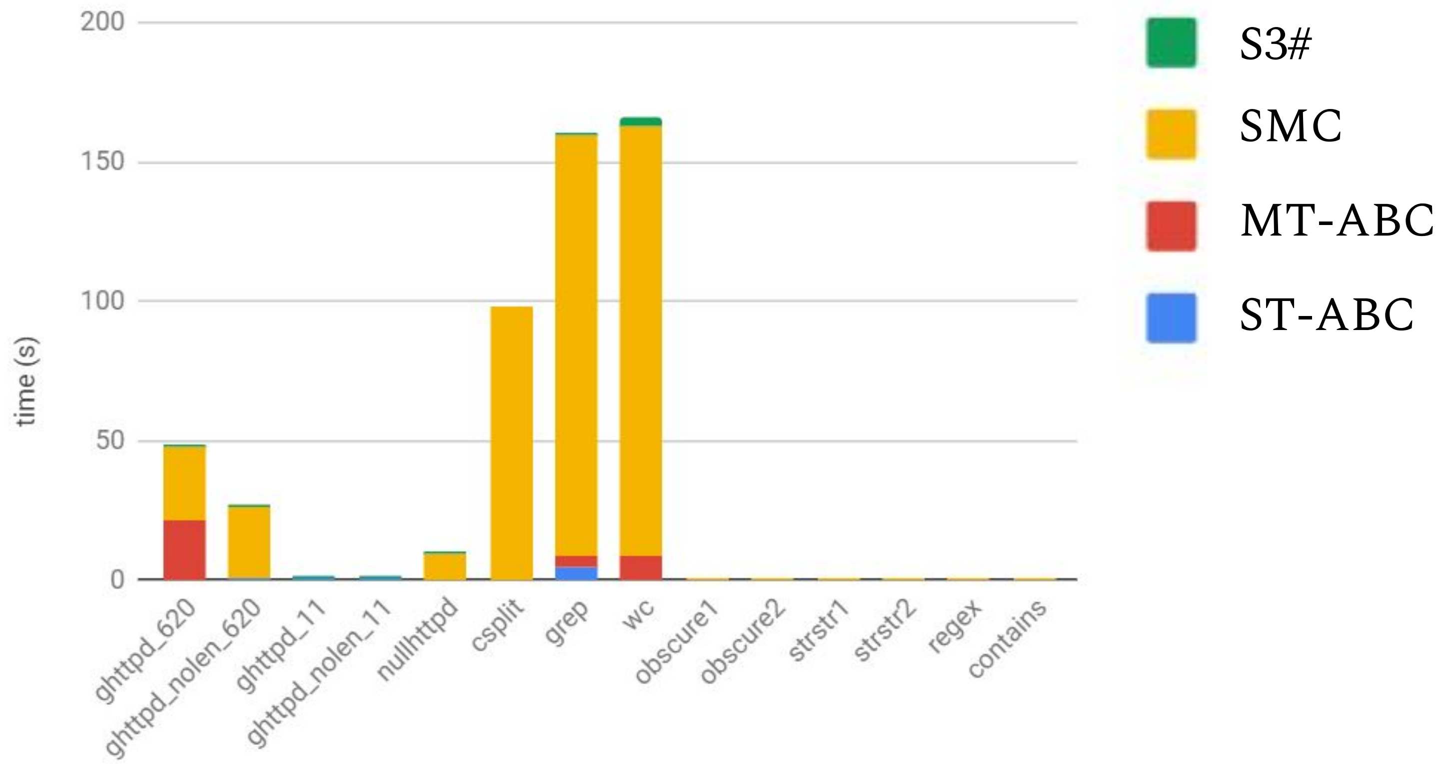
S3# security benchmark

- String constraint benchmark introduced by authors of S3# to evaluate their tool
  - 14 constraints taken from various security contexts
  - Comparison with SMC, ST-ABC
- We extend the comparison with results from MT-ABC

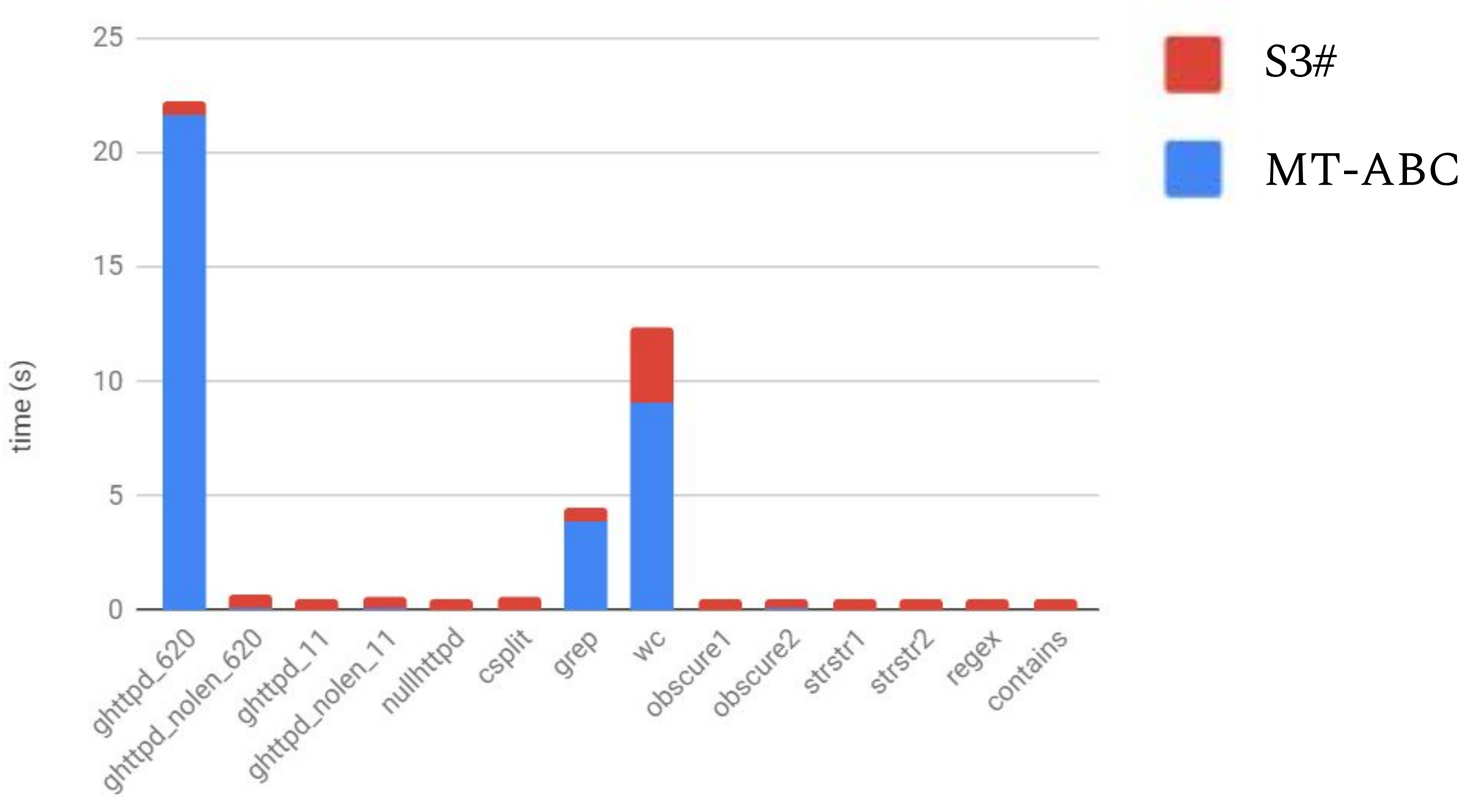
S3# security benchmark: # of precise results



S3# security benchmark: Execution time



S3# security benchmark: Execution time

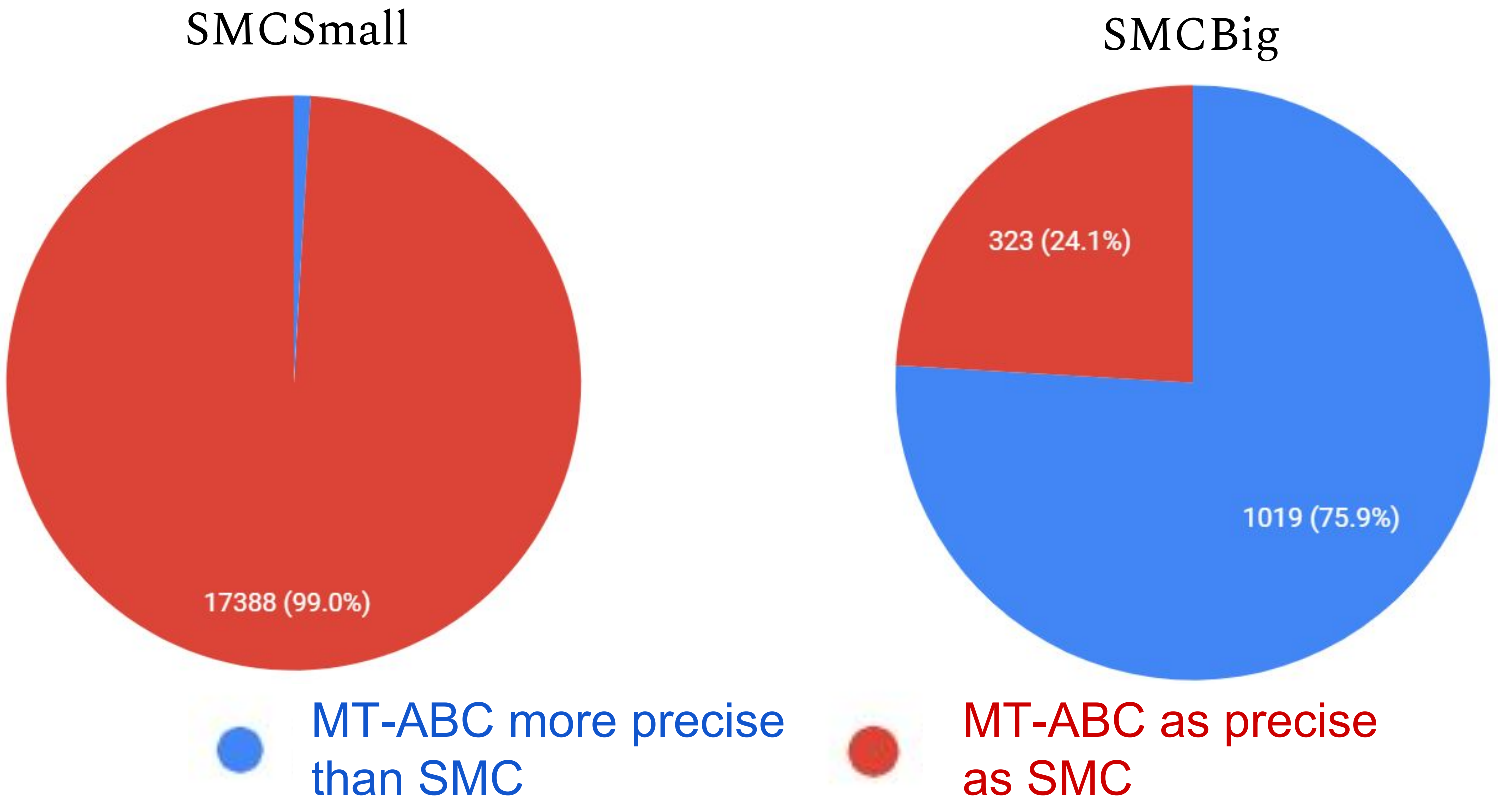


Kaluza benchmark

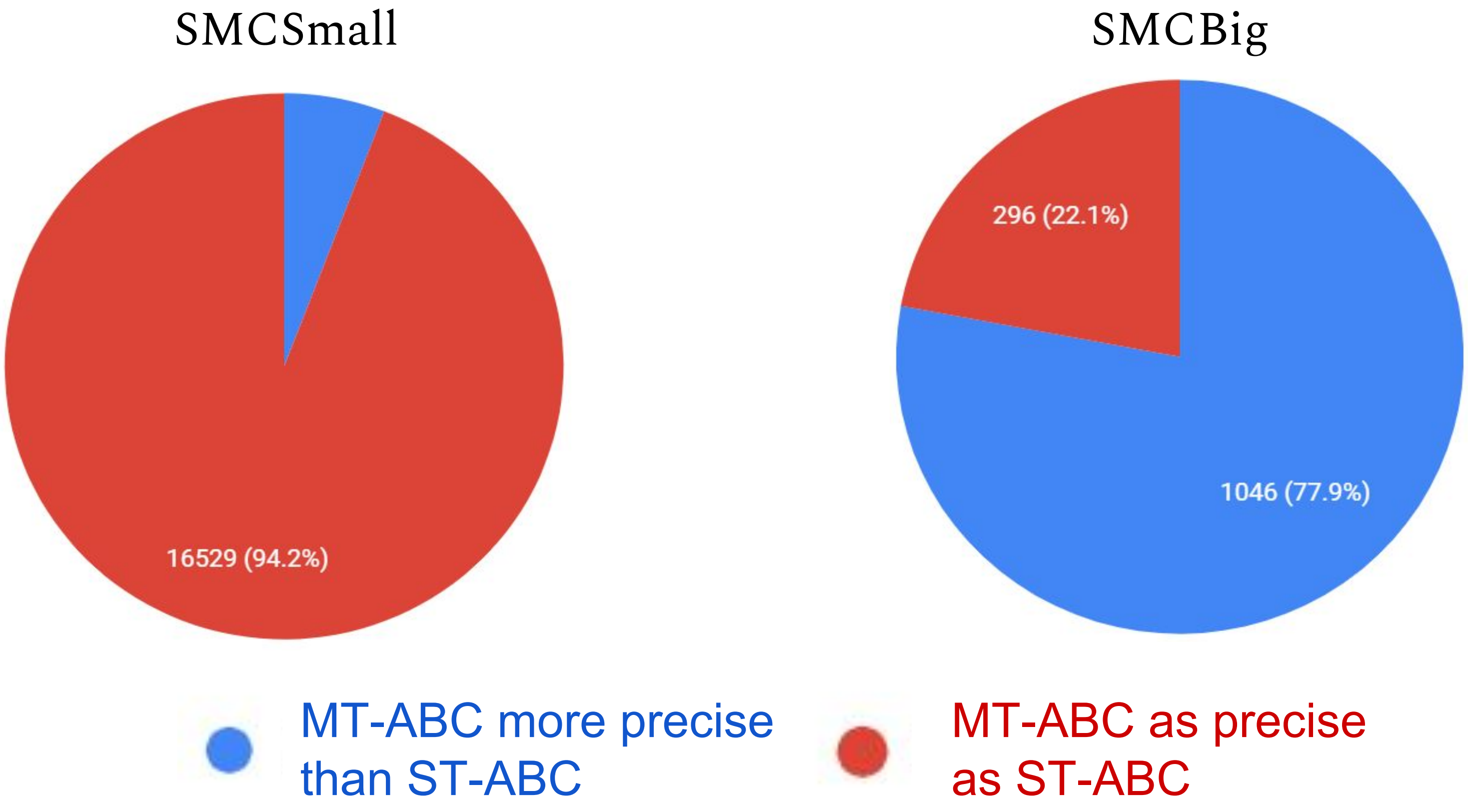
- Kaluza benchmark generated via symbolic execution of JavaScript programs
- Simplified and partitioned into two benchmarks by SMC authors
  - SMCSmall (17544 constraints), SMCBig (1342 constraints)
  - Removed disjunctions and replaced integer variables with constants

- Given a query variable, count the number of solutions with length <= 50
  - Evaluated efficiency and precision of MT-ABC with ST-ABC and SMC

Simplified Kaluza benchmark: MT-ABC vs SMC



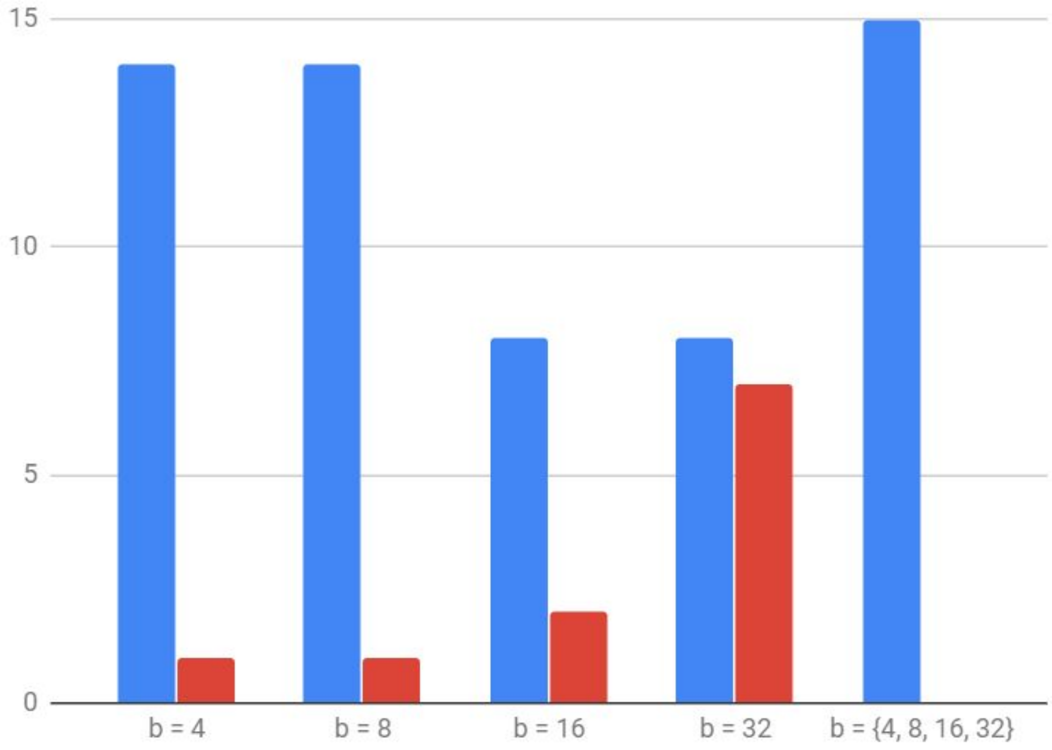
Simplified Kaluza benchmark: MT-ABC vs ST-ABC



Integer constraint benchmark

- Compared efficiency of MT-ABC with LattE for model counting linear arithmetic constraints
  - Both tools can precisely model count linear arithmetic constraints
  - Focus on timing comparison between both
- Evaluated each tool on benchmark for varying bit length bounds

Integer constraint benchmark: Execution time



MT-ABC faster execution time

LattE faster execution time

Mixed constraint benchmark

Compare MT-ABC with S3# in the context of mixed string and integer constraints

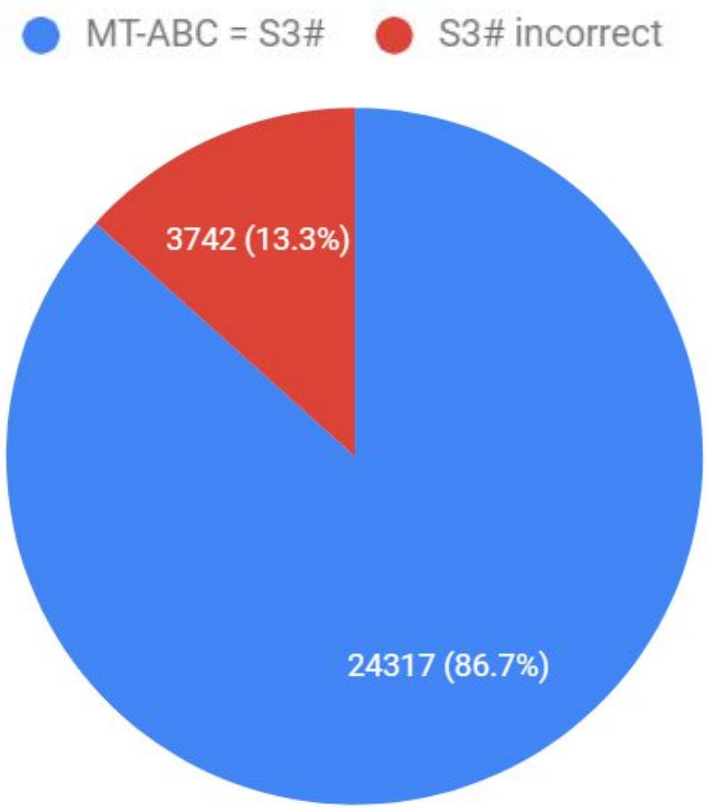
- Only known model counter claiming to handle this constraint combination

Evaluated using the Kaluza benchmark (unmodified)

- Features mixed string and integer constraints
- Used by S3# authors to prove their claim

Mixed constraint benchmark

- MT-ABC, S3# agree on count for many of the constraints
  - S3# gave same lower/upper bounds
- S3# counts incorrect for the rest
  - Manually confirmed MT-ABC correct
  - S3# lower/upper bounds **incorrect**



Conclusion

- String, numeric and mixed constraints can be mapped to automata
- Automata representation for constraints reduces model counting problem to path counting in graphs
- MT-ABC performs as well as domain specific string and integer model counters
- MT-ABC is the only model counter that can handle mixed string and numeric constraints

Thanks!