Succinct Determinisation of Counting Automata via Sphere Construction

Lenka Turoňová [1]

Margus Veanes [2], Lukáš Holík [1], Ondřej Lengál [1], Olli Saarikivi [2], Tomáš Vojnar [1]

[1] FIT, Brno University of Technology; [2] Microsoft Research, Redmond, USA

Motivation

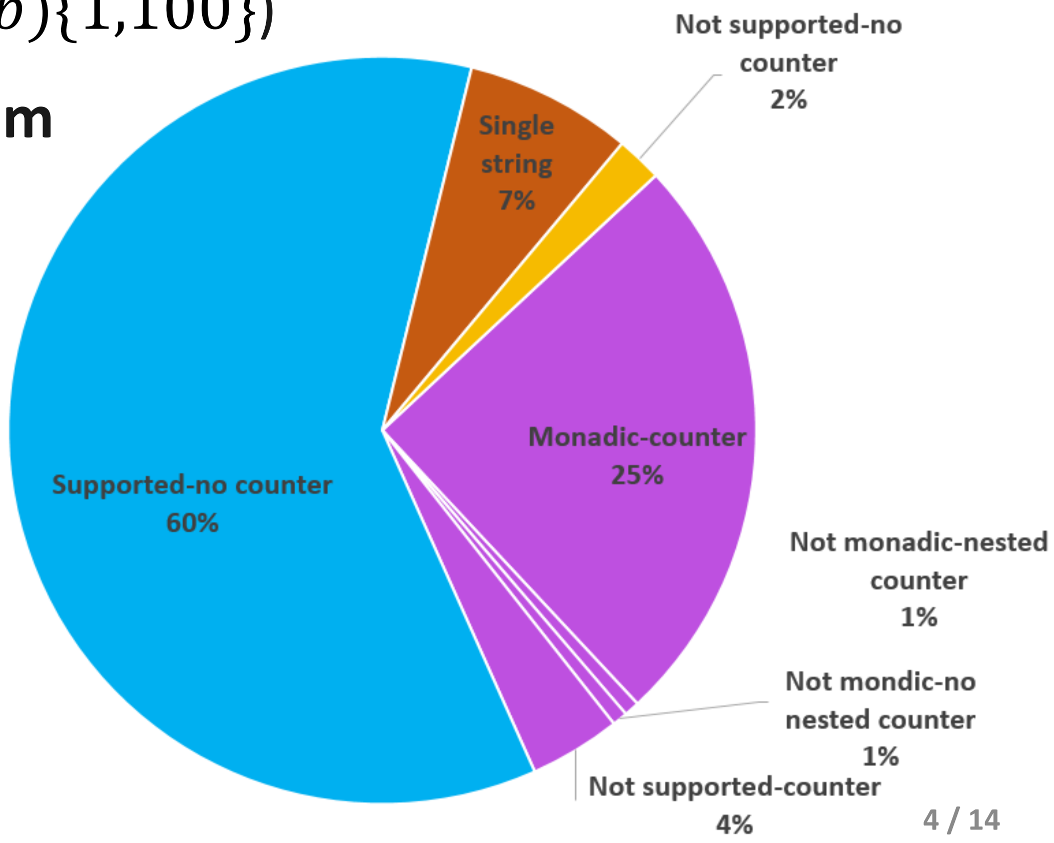
- **pattern matching**
  - recognising URIs, markup code, pieces of Java code, or SQL queries;
  - finding attacks in network traffic;
  - in real-life XML schemas (bounds 10 million);
  - identifying credential leaks in source code and configuration files;
  - 30-40 % of Java, JavaScript, and Python software;
- **efficiency** of matching engines – impact on the usability of SW applications
- unpredictability of their performance may lead to **catastrophic consequences** (e.g. *a global outage of Cloudflare services*)


PATTERN MATCHING
LIGHTYEAR
PATTERN MATCHING
EVERYWHERE

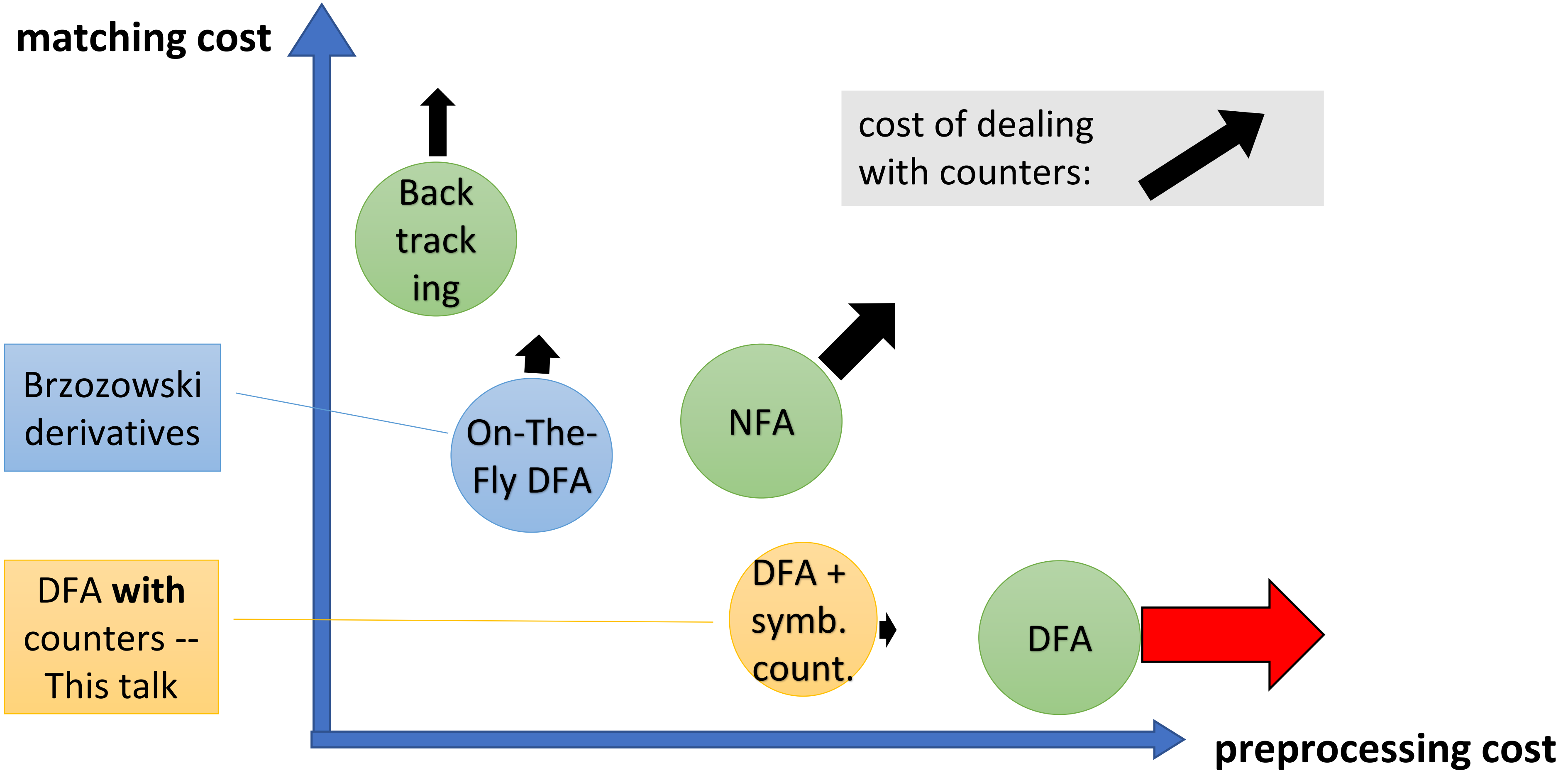
Regular Expressions with Counters

- **counting operator** is used in *extended regular expressions* – a number of repetitions of subexpressions (e.g. $(ab)\{1{,}100\}$)
- regexes from **a Microsoft product team**

- 152 regular expressions
- 31 % regexes with counters



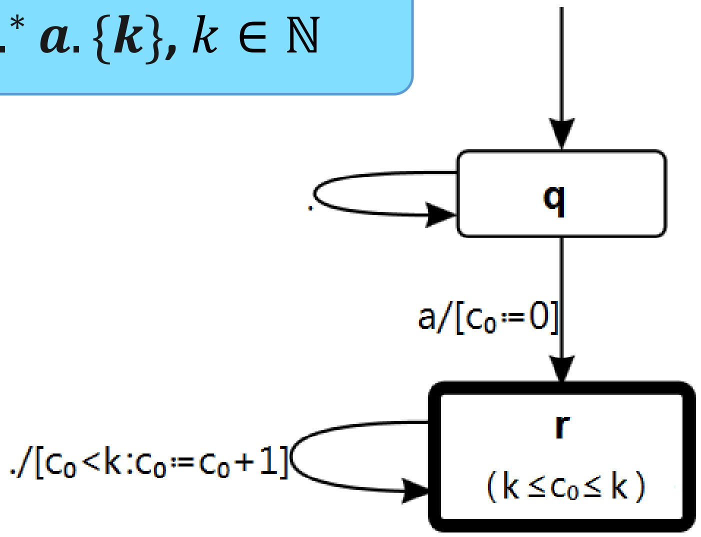
Wide Spectrum of Regex Matching Techniques



Naive Determinisation

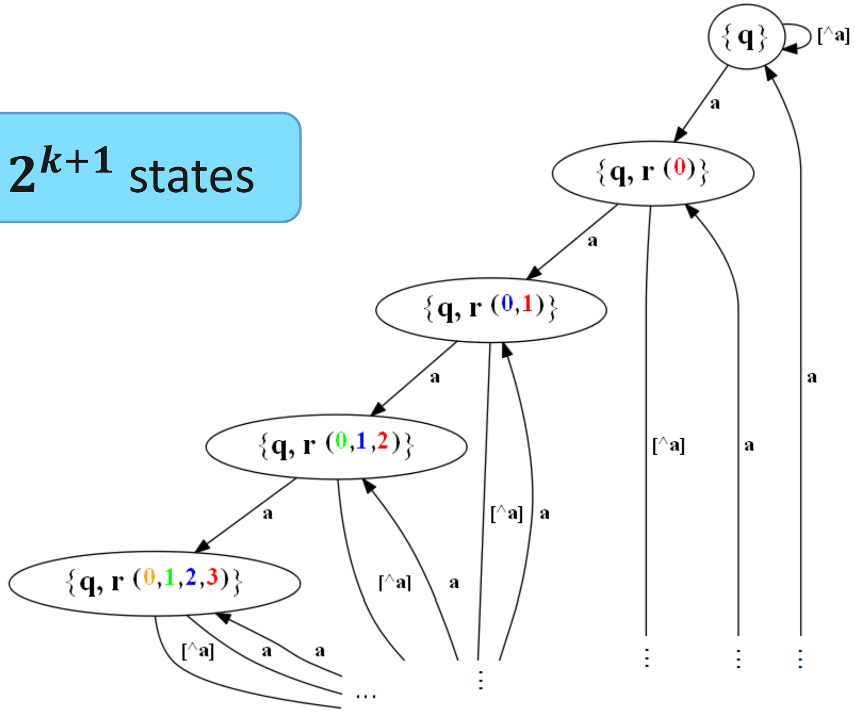
- classical subset construction in which concrete values of counters become a part of the control states

$.^{*}\, a.\{k\},\ k \in \mathbb{N}$



NCA

2^{k+1} states

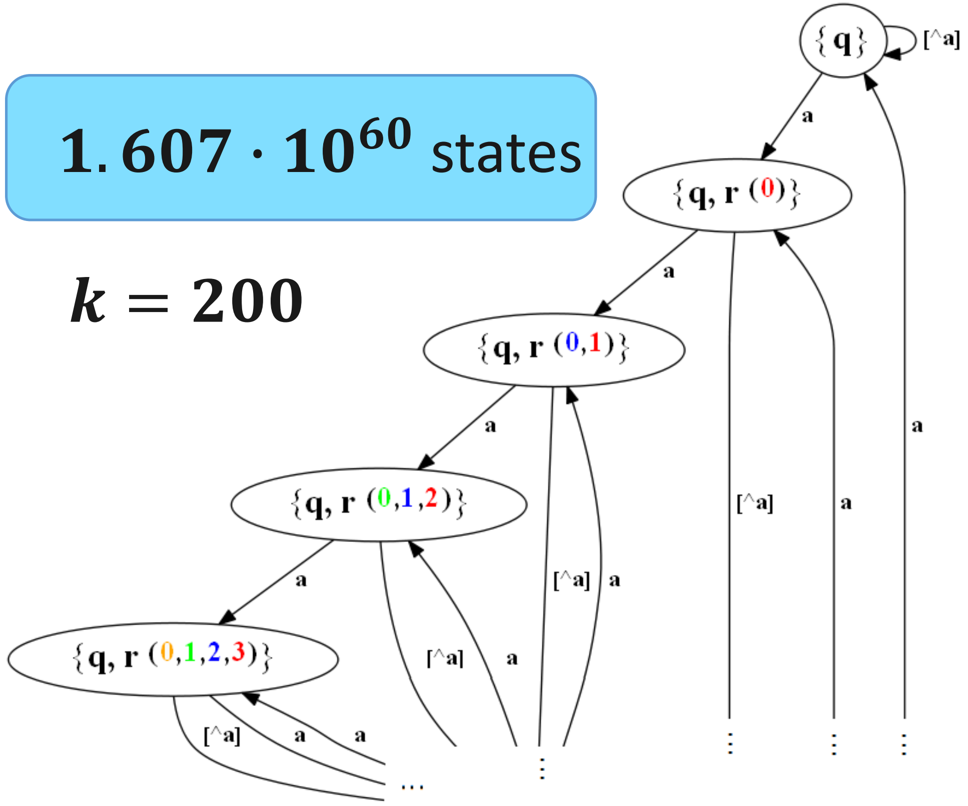


DFA

Naive vs Proposed Determinisation

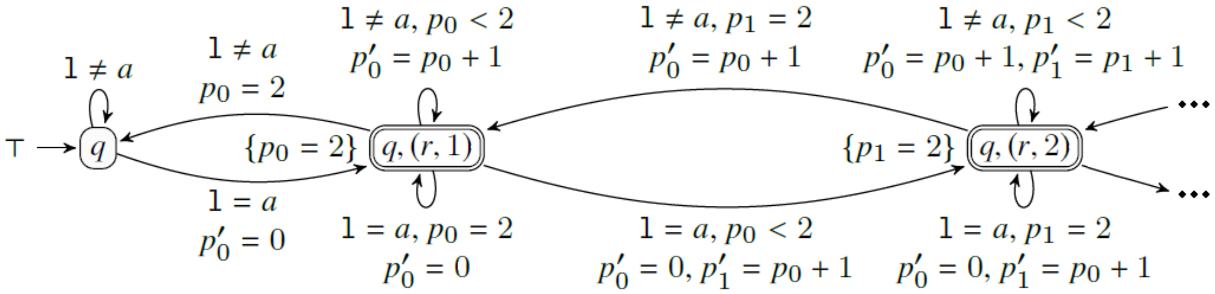
$1.607 \cdot 10^{60}$ states

$k = 200$



DFA

$k + 1$ states



DCA

Counting Automata

$$\boldsymbol{A} = (\boldsymbol{Q}, \boldsymbol{C}, \boldsymbol{I}, \boldsymbol{F}, \Delta)$$

$$q \xrightarrow{\{g,f\}} r$$

- $\boldsymbol{Q}$ – a finite set of control states,
- $\boldsymbol{C}$ – a set of counters,
- $\boldsymbol{I}, \boldsymbol{F}$ – an initial / final formula,
- $\boldsymbol{\Delta}$ – a set of transition formulae: $(s = q) \wedge g \wedge f \wedge (s' = r)$,
  - $\boldsymbol{s}$: a unique state variable,
  - $\boldsymbol{g}$: a guard formula on input: $(l = a, a \in \Sigma)$ and counters $(c \leq k, c \geq k, k \in \mathbb{N})$,
  - $\boldsymbol{f}$: counter assignment $c' \coloneqq (d + k)/k,$ where $k \in \mathbb{N}$ and d is a counter
- **bounded counters**: $\exists \boldsymbol{max}_c : \forall\, c \in C : c \leq \boldsymbol{max}_c$

Proposed Determinisation – Basic Notions

- **outcome** φ of $w \in \Sigma^*$ – a formula over **state variables** and **counters** representing a set of configurations reachable by reading the word w

$$s = q \land (c = 2 \lor c = 3) \qquad s = q \land (c = 1 \lor c = 2)$$

- **sphere** Ψ
  - created from an outcome by replacing **concrete values** of the counters with **parameters** (a number of parameters up to $\boldsymbol{max_c} + 1$)
  - a value of the counters is **computed at runtime**

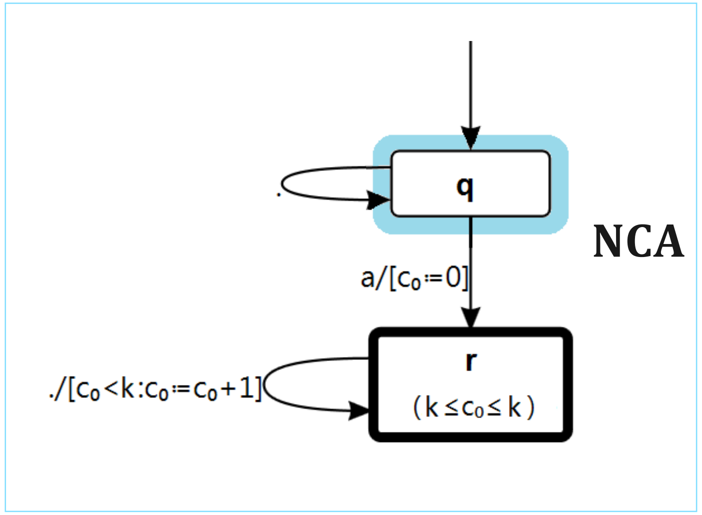
$$\psi \stackrel{\text{def}}{=} s = q \land (c = p_1 \lor c = p_2)$$

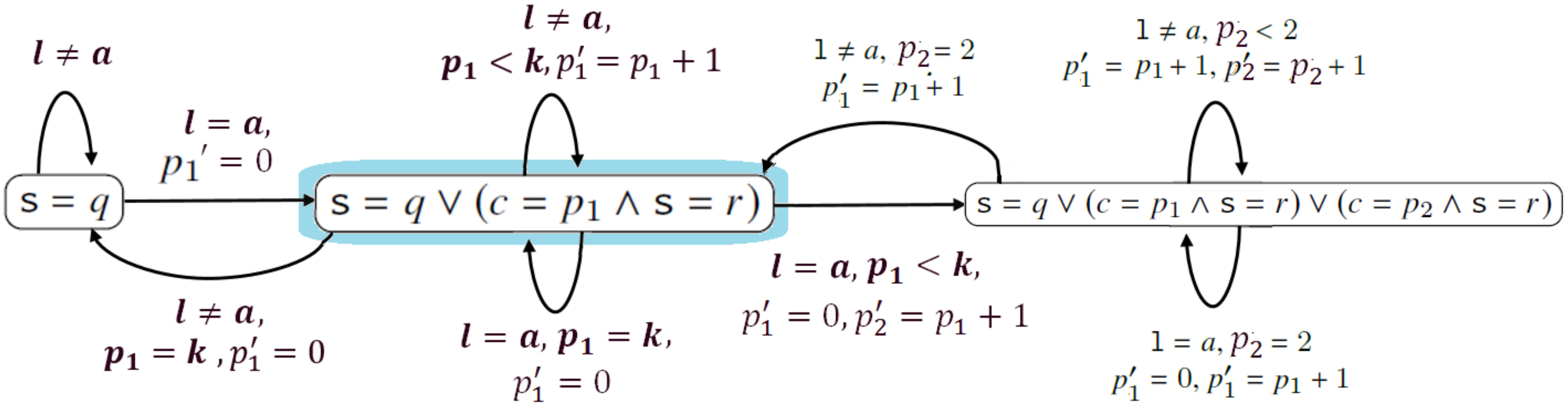
Determinisation: Example (Initialisation)

$$I^d = (s = \Psi_I)$$

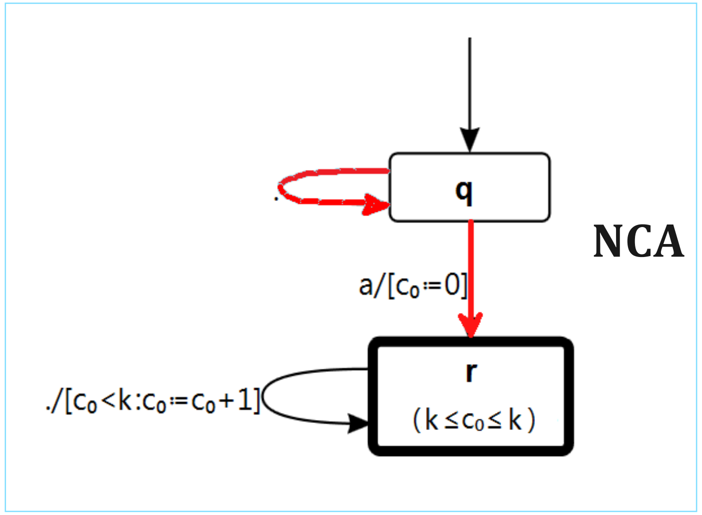
$$\Psi_I = \boxed{s = q}$$

$$.^* \, a.\{k\}, k = 2$$





Determinisation – Example (DCA transitions)

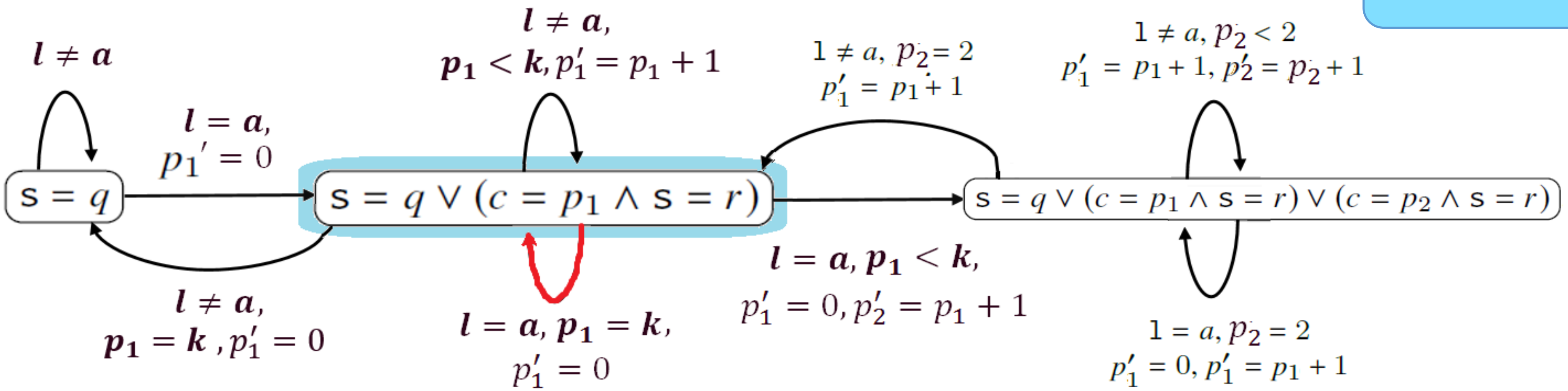


- **factorisation of guards of transitions leaving a set of NCA states:**

$$g_1: \quad (l \neq a) \wedge (p_1 < k)$$
$$g_2: \quad (l = a) \wedge (p_1 < k)$$
$$g_3: \quad (l \neq a) \wedge (p_1 = k)$$
$$g_4: \quad (l = a) \wedge (p_1 = k)$$

$$.^{*}\, \boldsymbol{a}.\{\boldsymbol{k}\}, k = 2$$

- **update:** updates of NCA transitions satisfying the guard



Monadic Regular Expressions

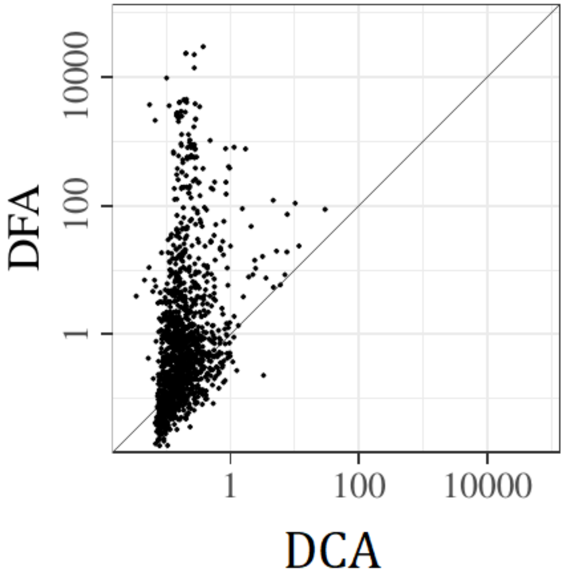
- regular expressions extended with counting limited to **character classes**
- optimization: parameter values **kept in a queue**, testing **only on maximum value**

monadic	**non-monadic**
$.* \, a.\{2\}$,	$(Karel)\{2\}$,
$[A-Za-z0-9\_]\{4,10\}$	$(fixup\|squash)\{2\}$
$http\backslash://www.[A-Za-z0-9\_]+$	$(?:\backslash\backslash s*\$NL)\{2\}$

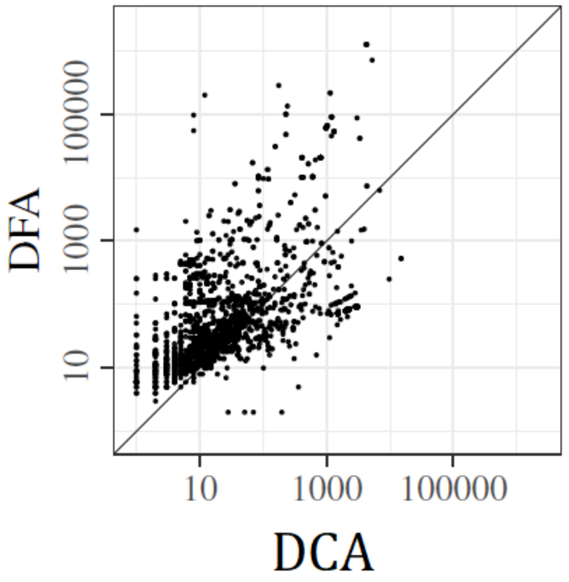
- a much **lower worst-case complexity** – number of states depends on max_c only **polynomially**

Experimentation

- extension of the **Microsoft's Automata library** with a prototype support for counters
- **2 362 regexes**: network intrusion detection systems (*Snort, Bro*), Microsoft's security leak scanning system, log analyses engine, …



Ratio of times of conversion of regexes to DFA and DCA



Ratio of the number of states of DFA and DCA

Future Work

- optimised representations of counters for specific but frequent cases
- efficient algorithm beyond a subclass of monadic regular expressions
- integration to a matching loop, match generator
- lazy vs. eager loops
- minimisation of CAs
- Boolean operations (product, complement, …)

Thank you for attention...

Monadic Regular Expressions

- 152 regular expressions
- 99 % monadic regexes