Solving Extended Regular Constraints Symbolically

Margus Veanes
Microsoft Research, Redmond

Initial Motivation

- **Test table** and **test data** generation for SQL queries
  - Given a query **q**, what is an interesting set of tables and parameters such that q satisfies a given test condition φ?
  - Queries often involve LIKE-patterns (special **regexes**), e.g.:

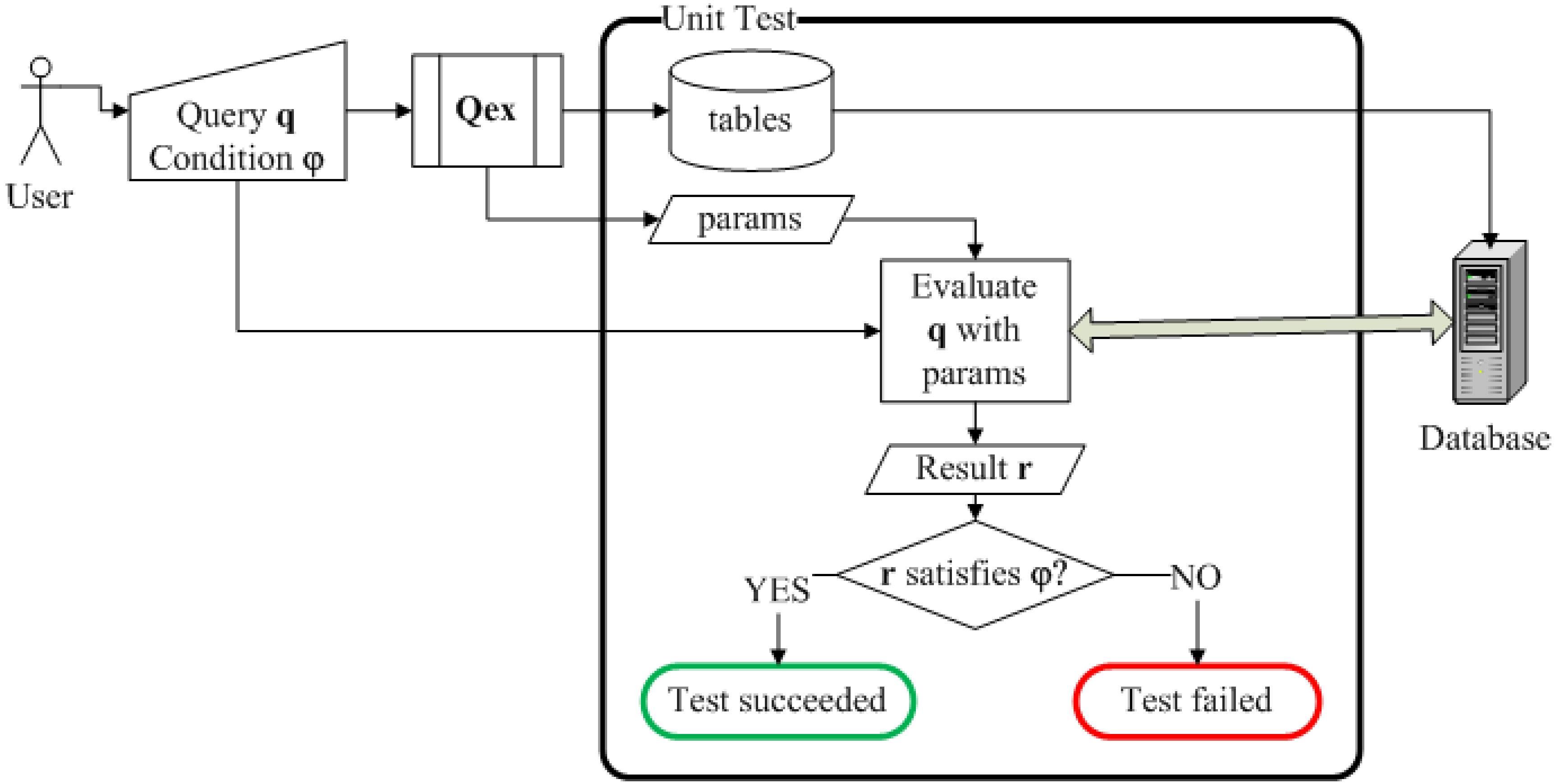
q:
```
SELECT * FROM T
WHERE C LIKE "Mar%" AND NOT C LIKE "%gus" AND LEN(C) < D + E
```

C	D	E
?	?	?
?	?	?
?	?	?

Eval(q)

Ø

Table generation: usage scenario



General Approach

- Encode the query and the condition as a formula **F** using rich **background theories T**
  - bags, tuples, arithmetic, …
- Represent the problem with ***Satisfiability Modulo T:***
  - *Does F have a model M s.t. M is a model of T*?
- Use SMT solving for generating M
  - ***power-user of Z3***
- Extract the test data from M

Paper presented at *ICFEM'09* last week
coauthors: *Pavel Grigorenko, Nikolai Tillmann, Peli de Halleux*

How far can we push **T**?

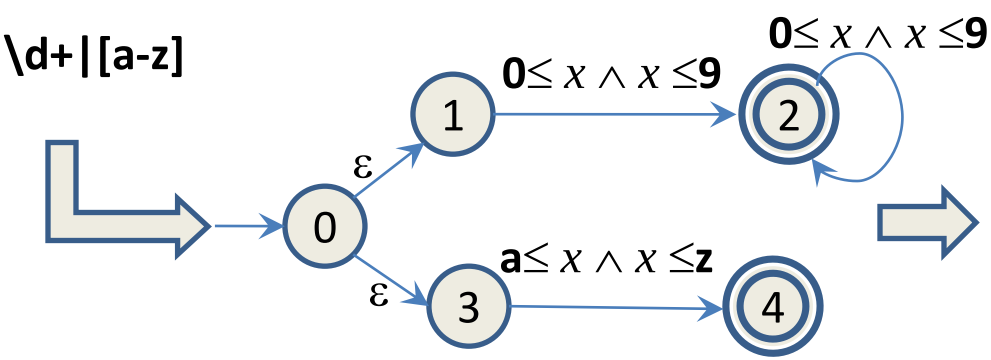
- How can we support constructs like **LIKE**?

```
SELECT * FROM T  WHERE C LIKE "%Bob%" AND …
```

  - Must be *extensible* with other theories
    - recall first example that used *linear arithmetic* and *string-length constraints*
  - Requires a combination of techniques from:
    - Language theory
    - SMT (SAT) solving
  - Lead to an approach for solving *regular constraints* symbolically.   --- > **Rex**

Rex: Symbolic **R**egex **ex**plorer

- Constraints involving regular expressions are converted into specialized theories
  - Can be combined with other constraints on strings, e.g. length constraints, and other theories



$$\forall w\, n\, (Acc(w,n) \Leftrightarrow (Acc_1(w,n) \vee Acc_3(w,n)))$$

$$\forall w\, n\, (Acc_1(w,n) \Leftrightarrow (w \neq \texttt{""} \wedge n > 0 \wedge \psi[hd(w)] \wedge Acc_2(tl(w), n-1)))$$

$$\forall w\, n\, (Acc_2(w,n) \Leftrightarrow ((w \neq \texttt{""} \wedge n > 0 \wedge \psi[hd(w)] \wedge Acc_2(tl(w), n-1)) \vee (w = \texttt{""} \wedge n = 0)))$$

$$\forall w\, n\, (Acc_3(w,n) \Leftrightarrow (w \neq \texttt{""} \wedge n > 0 \wedge \varphi[hd(w)] \wedge Acc_4(tl(w), n-1)))$$

$$\forall w\, n\, (Acc_4(w,n) \Leftrightarrow (w = \texttt{""} \wedge n = 0))$$

- Besides **Qex**, immediate application in **Pex** (Parametrized unit-testing framework for C#)
  - Regular constraints occur in C# programs
  - Rex is going to be integrated into Pex

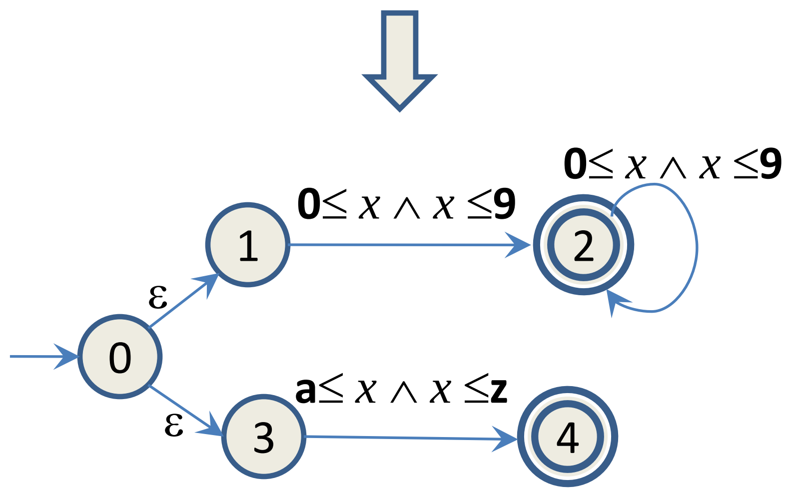
The idea behind **Rex**

- A *regex* r is translated into a *finite symbolic automaton* (FSA) A_r
- A_r is translated into a theory $Th(A_r)$ with a binary relation symbol Acc called a *symbolic language acceptor* such that
  $\{s \in L(r) \mid len(s)=k\} = \{w^M \mid M \vDash Acc(w,k)\}$
- $Th(A_r)$ is added into the background theory T

From regex to FSA

- Example:

Regex: **\d+|[a-z]**



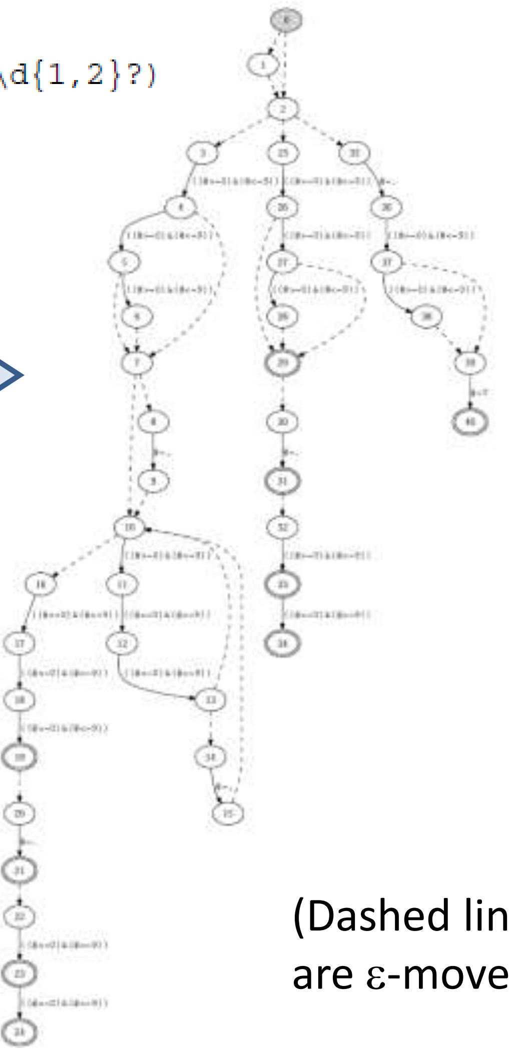
Note: a move $(p,\ \varphi[x],\ q)$ encodes the *set* of transitions

$$\{(p,\ x^M,\ q) \mid M \vDash \varphi[x]\}$$

Larger example of FSA(r)

```
$?(\d{1,3},?(\d{3},?)*\d{3}(\.\d{0,2})?|\d{1,3}(\.\d{0,2})?|\.\d{1,2}?)
```

- **Note**: The FSAs are typically *sparse* graphs



(Dashed lines are ε-moves)

Background Universe

- The background universe is *multi-sorted*
  - Basic sorts:
    - Integers, rational numbers, bit-vectors, Booleans ($\mathbb{B}$),
  - Algebraic datatypes:
    - Lists: $\mathbb{L}\langle\sigma\rangle$ where σ is a sort
      - Constructors: nil: $\mathbb{L}\langle\sigma\rangle$, $cons$: $\sigma \times \mathbb{L}\langle\sigma\rangle \rightarrow \mathbb{L}\langle\sigma\rangle$
      - Accessors: $hd : \mathbb{L}\langle\sigma\rangle \rightarrow \sigma,\ : tl : \mathbb{L}\langle\sigma\rangle \rightarrow \mathbb{L}\langle\sigma\rangle$
    - Unary natural numbers: $\mathbb{N}$ (successor arithmetic)
    - Trees …
- There are *built-in* functions: =, <, +, ….
- The signature can be expanded with *fresh uninterpreted* function symbols

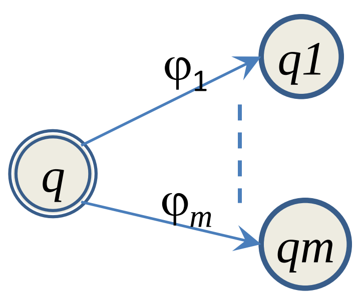
From FSA to Axioms

- Axioms in $Th(A)$ use lists to represent strings
  - *Characters* $\mathbb{C}$ are k-bit-vectors (k=16 for Unicode)
  - *Strings* are lists of characters $\mathbb{L}\langle\mathbb{C}\rangle$
- Given FSA A, for each state q of A, declare fresh:
  - $Acc_q : \mathbb{L}\langle\mathbb{C}\rangle \times \mathbb{N} \to \mathbb{B}$, let Acc be Acc_{q0} were $q0$ is the initial state
  - Define the axioms (if q is a final state, similarly for other states)
    - $\forall\ s\ (Acc_q(s, 0) \Leftrightarrow s = nil)$
    - $\forall\ s\ n\ \big(Acc_q(s, succ(n)) \Leftrightarrow\ s \neq nil \wedge$

$$((\varphi_1[hd(s)] \wedge Acc_{q1}(tl(s),n)) \vee \cdots \vee (\varphi_m[hd(s)] \wedge Acc_{qm}(tl(s),n)))\big)$$

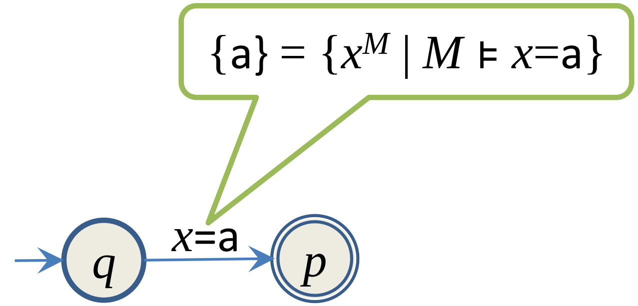
Note: ε-moves add additional disjuncts in *rhs* where $succ(n)$ is not decremented

for the moves:



Step-by-step example ($Th(A)$ construction)

- Given regex r: "a", i.e. $L(r)=\{a\}$
- Construct automaton A
- Define $Th(A)$:
  - $\forall s\ Acc_q(s,0) \Leftrightarrow false$ (q is not final)
  - $\forall s\ n\ Acc_q(s,succ(n)) \Leftrightarrow hd(s)=a \wedge Acc_p(tl(s),n)$
  - $\forall s\ Acc_p(s,0) \Leftrightarrow s=nil$ (p is final)
  - $\forall s\ n\ Acc_p(s,succ(n)) \Leftrightarrow false$ (p has no outgoing moves)



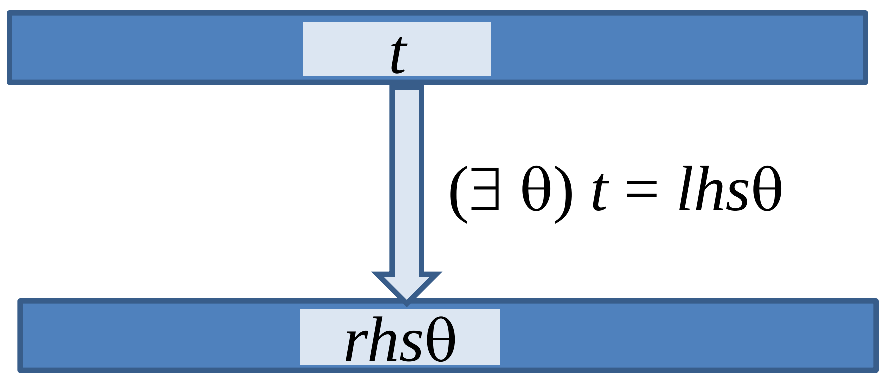
In general, axioms may also be nonequational and are *triggered* by associated *patterns*

E-matching in Z3

- Equational axioms have the form
  $\forall\, \boldsymbol{x}\ (lhs[\boldsymbol{x}] = rhs[\boldsymbol{x}])$
  (Note that '$=$' is same as '$\Leftrightarrow$' when $lhs, rhs{:}\mathbb{B}$)
- There is a *current goal* that is a *quantifier free ground* formula, axioms are used to *rewrite* the goal during model generation by matching axioms (from left to right):

current goal =

new goal =



Step-by-step example (E-matching)

- Assuming $Th(A_r)$ as defined earlier for r=“a”
- Declare $w{:}\mathbb{L}\langle\mathbb{C}\rangle$ as an *uninterpreted constant*
- Consider the goal $Acc_q(w, succ(0))$
- E-matching:

$Acc_q(w, succ(0))$

($\exists\ \theta=\{s\mapsto w,\ n\mapsto 0\}$) $Acc_q(w, succ(0)) = Acc_q(s, succ(n))\theta$
Thus: replace $Acc_q(w, succ(0))$ with $hd(s)=\text{a} \wedge Acc_p(tl(s),n)\theta$

$hd(w)=\text{a} \wedge Acc_p(tl(w),0)$

($\exists\ \theta=\{s\mapsto tl(w)\}$) $Acc_p(tl(w),0) = Acc_p(s,0)\theta$
Thus: replace $Acc_p(tl(w),0)$ with $(s=nil)\theta$

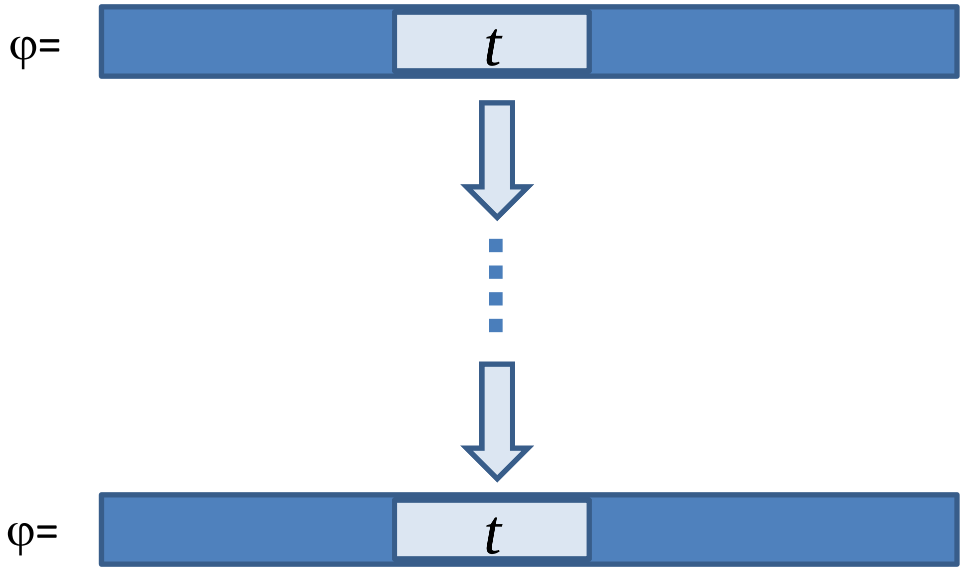
$hd(w)=\text{a} \wedge tl(w)=nil$

- Now $hd(w)=\text{a} \wedge tl(w)=nil$ has a model M using the built-in list theory, namely $w^M = cons(\text{a},nil)$

General problems with recursive axioms

- *Wrong*: might not capture the intended semantics
- May cause *nontermination* of E-matching

(For FSAs both problems arise with ε-loops)



Acceptors for Symbolic PDAs

- Allows to deal with CFGs, possible applications:
  - Pex data generation for XML
  - SQL injection vulnerability checking
- Can be combined with regular acceptors
- Given SPDA A, Each q-acceptor predicate is declared as
  - $Acc_q : \mathbb{L}\langle\mathbb{C}\rangle \times \mathbb{L}\langle\mathbb{Z}\rangle \times \mathbb{N} \rightarrow \mathbb{B}$, where $\mathbb{L}\langle\mathbb{Z}\rangle$ is a *stack* where $\mathbb{Z}$ is a sort for stack symbols (e.g. integers)
- The axioms $Th(A)$ are defined similarly to FSAs where $\forall\ s\ n\ (Acc(s,n) \Leftrightarrow Acc_{q0}(s,cons(z0,nil),n))$ ($z0$ is the stack start symbol and $q0$ the initial state)

Conditional correctness of $Th(A)$

- **Theorem***: Let A be an FSA without ε-loops. $Th(A) \wedge Acc^A(s,k)$ is sat. ⇔ $s \in L(A)$ and $len(s)=k$.

  * M. Veanes, P. de Halleux, and N. Tillmann, "Rex: Symbolic regular expression explorer," Microsoft Research, Tech. Rep. MSR-TR-2009-137, October 2009.

- A similar statement can be proved when A is an SPDA.

Equivalent forms of $Th(A)$

- There are straightforward generalizations* of classical algorithms of (N)FAs to FSAs, such as:
  1. Epsilon elimination
  2. Determinization
  3. Minimization
  4. Product construction

- What is the effect of the algorithms on $Th(A)$? **Not obvious**.
  – (1) eliminates ε-loops but increases complexity of conditions, that may increase overall complexity of $Th(A)$
  – For (2) and (3) performance is highly unpredictable
  – (4) seems to be useful: since $Acc^A(s,k) \wedge Acc^B(s,k) \Leftrightarrow Acc^{A\times B}(s,k)$ and $Th(A\times B)$ may be considerably simpler than $Th(A) \cup Th(B)$

- **?**: computational complexity of (2) and (3) for FSAs
  – **Note**: (2) and (3) use sat. checking of single-variable bit-vector formulas

Member generation experiments

SAMPLE REGEXES.

<table>
<tr><td>#1</td><td>\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*([,;]\s*\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*)*</td></tr>
<tr><td>#2</td><td>$?(\d{1,3},?(\d{3},?)*\d{3}(\.\d{0,2})?|\d{1,3}(\.\d{0,2})?|\.\d{1,2}?)</td></tr>
<tr><td>#3</td><td>([A-Z]{2}|[a-z]{2} \d{2} [A-Z]{1,2}|[a-z]{1,2} \d{1,4})?([A-Z]{3}|[a-z]{3} \d{1,4})?</td></tr>
<tr><td>#4</td><td>[A-Za-z0-9](([\.\-]?[a-zA-Z0-9]+)*)@([A-Za-z0-9]+)(([\.\-]?[a-zA-Z0-9]+)*)\. ([A-Za-z][A-Za-z]+)</td></tr>
<tr><td>#5</td><td>(\w|-)+@((\w|-)+\.)+(\w|-)+</td></tr>
<tr><td>#6</td><td>[+-]?([0-9]*\.?[0-9]+|[0-9]+\.?[0-9]*)([eE][+-]?[0-9]+)?</td></tr>
<tr><td>#7</td><td>((\w|\d|\-|\.)+)@{1}(((\w|\d|\-){1,67})|((\w|\d|\-)+\.(\w|\d|\-){1,67}))\.((([a-z]|[A-Z]|\d){2,4})(\.([a-z]
|[AZ]|\d){2})?)</td></tr>
<tr><td>#8</td><td>(([A-Za-z0-9]+ +)|([A-Za-z0-9]+\-+)|([A-Za-z0-9]+\.+)|([A-Za-z0-9]+\++))*[A-Za-z0-9]+@((\w+\-+)|(\w+\.))*\w
{1,63}\.[a-zA-Z]{2,6}</td></tr>
<tr><td>#9</td><td>(([a-zA-Z0-9 \-\.]+)@([a-zA-Z0-9 \-\.]+)\.([a-zA-Z]{2,5}){1,25})+([;.](([a-zA-Z0-9 \-\.]+)@([a-zA-Z0-9 \-\.
]+)\.([a-zA-Z]{2,5}){1,25})+)*</td></tr>
<tr><td>#10</td><td>((\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*)\s*[,]{0,1}\s*)+</td></tr>
</table>

EVALUATION RESULTS FOR SAMPLE REGEXES.

	ϵFSA(r)		FSA(r)		DFSA(r)		mDFSA(r)	
r	*size*	t ms	*size*	t ms	*size*	t ms	*size*	t ms
#1	91	100	73	40	81	70	20	140
#2	90	10	64	10	71	30	29	40
#3	83	10	70	10	104	30	69	100
#4	45	40	35	60	53	70	26	70
#5	98	100	71	10	74	30	15	40
#6	31	0	12	0	16	10	10	10
#7	2728	840	920	1800				
#8	281	40	269	60	380	170	296	870
#9	1944	280	2128	260				
#10	112	30	104	30				

Experiments with product

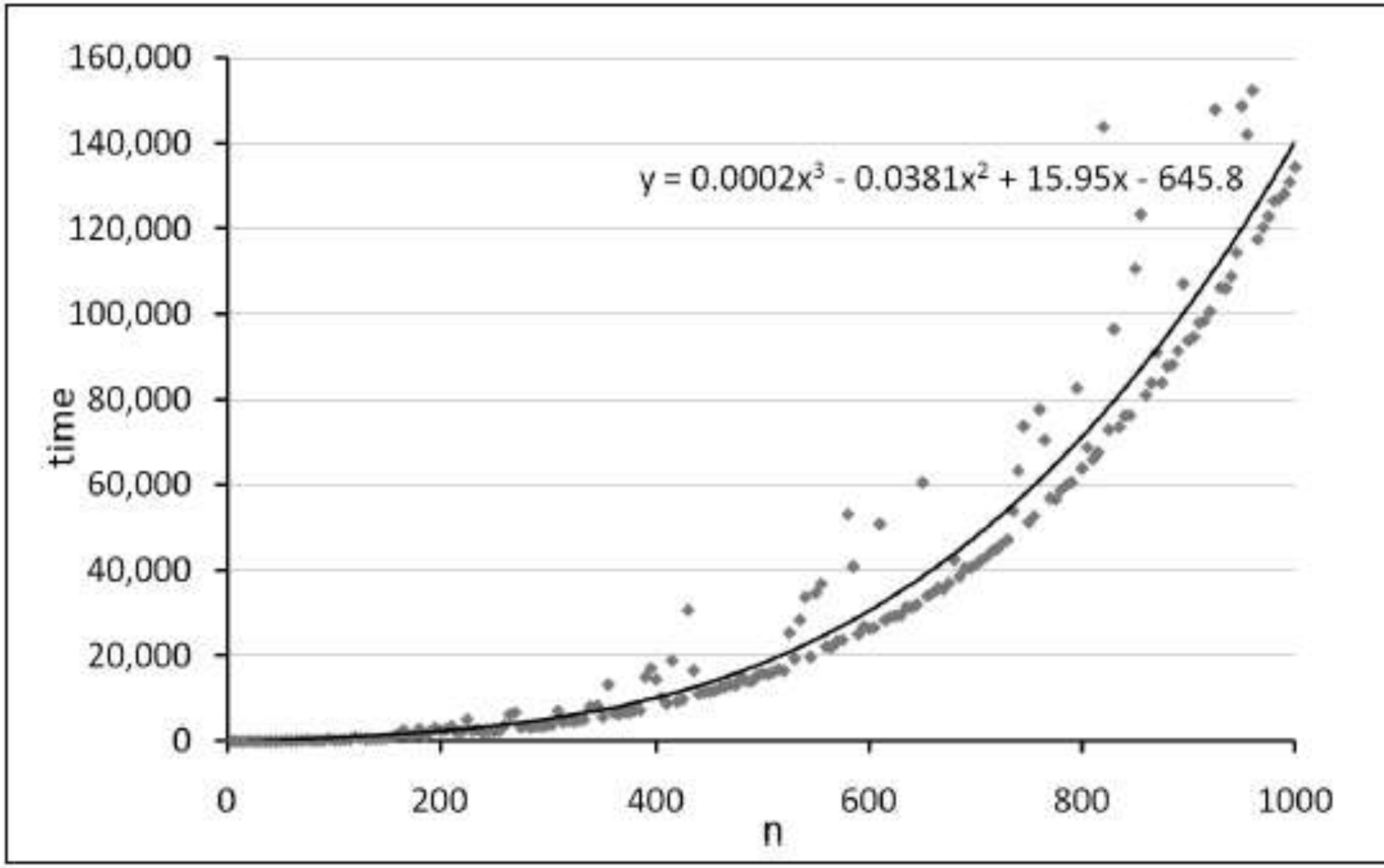


Figure 5. Member generation times (ms) for the intersection of the regexes `[a-c]*a[a-c]{n+1}` and `[a-c]*b[a-c]{n}` for n up to 1000.

Product construction of FSAs

- Given A and B construct C, $L(C) = L(A) \cap L(B)$

(i) Initially $S = (\langle q_{0A}, q_{0B} \rangle)$, $V = \{\langle q_{0A}, q_{0B} \rangle\}$, $T = \emptyset$.

(ii) If S is empty go to (iv) else pop $\langle q_1, q_2 \rangle$ from S.

(iii) Iterate for each $t_1 \in \Delta_A(q_1)$ and $t_2 \in \Delta_B(q_2)$, let $\varphi = Cond(t_1) \wedge Cond(t_2)$, let $p_1 = Target(t_1)$, and let $p_2 = Target(t_2)$. *If φ is satisfiable* then

- add $(\langle q_1, q_2 \rangle, \varphi, \langle p_1, p_2 \rangle)$ to T;
- if $\langle p_1, p_2 \rangle$ is not in V then add $\langle p_1, p_2 \rangle$ to V and push $\langle p_1, p_2 \rangle$ to S.

Proceed to (ii).

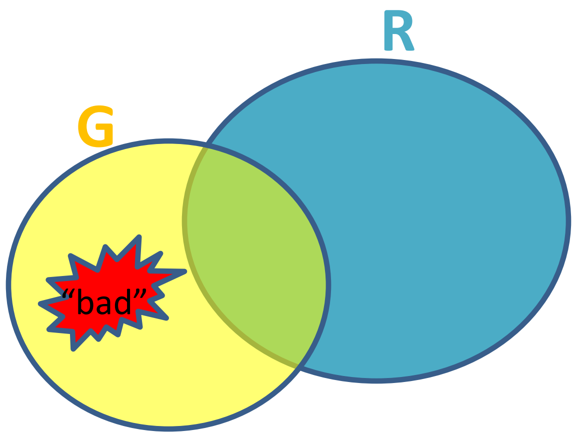
Using SMT

(iv) Let $C = (\langle q_{0A}, q_{0B} \rangle, V, \{q \in V \mid q \in F_A \times F_B\}, T)$.

(v) Eliminate *dead states* from C (states from which no final state is reachable).

Possible application of combining CF acceptors and Regular acceptors

- Decide if a CFG **G** is *not* a subset of Regex **R**?



- Does $Acc^{G}(x,k) \wedge \neg Acc^{R}(x,k)$ have a model M?
  - If *yes*, x^{M} is a *witness* of length k
  - **Equivalently**: is $Acc^{G\backslash R}(x,k)$ satisfiable?

Some related work

- G. V. Noord and D. Gerdemann, “Finite state transducers with predicates and identities,” *Grammars, vol. 4, 2001.*
- P. Hooimeijer and W. Weimer, “A decision procedure for subset constraints over regular languages,” in *PLDI, 2009, pp. 188–198.*
- A. Kiezun, V. Ganesh, P. J. Guo, P. Hooimeijer, and M. D. Ernst, “Hampi: a solver for string constraints,” in *ISSTA ’09. New* York, NY, USA: ACM, 2009, pp. 105–116.
- N. Li, T. Xie, N. Tillmann, P. de Halleux, and W. Schulte, “Reggae: Automated test generation for programs using complex regular expressions,” in *ASE’09.*
- N. Tillmann and J. de Halleux, “Pex - white box test generation for .NET,” in *TAP’08, LNCS, vol.* 4966. 2008
- N. Bjørner, N. Tillmann, and A. Voronkov, “Path feasibility analysis for string-manipulating programs,” in *TACAS’09,* LNCS, vol. 5505. 2009.
- M. Veanes, P. Grigorenko, P. de Halleux, and N. Tillmann, “Symbolic query exploration,” in *ICFEM’09, LNCS,* vol. 5885. , 2009.
- M. Veanes, P. de Halleux, N. Tillmann, “Qex: Symbolic Query Explorer”, Microsoft Research, Tech. Rep. MSR-TR-2009-2015. October 2009.
- M. Veanes, P. de Halleux, and N. Tillmann, “Rex: Symbolic regular expression explorer,” Microsoft Research, Tech. Rep. MSR-TR-2009-137, October 2009.
- M. Veanes, N. Bjørner, L. de Moura, “Solving Extended Regular Constraints Symbolically” Microsoft Research, Tech. Rep. Dec 2009.

Future directions

- Pex, Qex, Rex, …
  - applications in program (DB) analysis
- Solving language theoretic problems with SMT
  - e.g. grammar ambiguity (string with two parse trees)
- General transition systems
  - Applications in model-based testing and model checking
- Symbolic automata theory

…

Tänan tähelepanu eest!

Questions?