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 $\varphi$?
  - 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

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

$\text{Eval}(q)$ \[ \emptyset \]
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 Regex explorer

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

\[d+|[a-z]\]

- 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

\[ \{ s \in L(r) \mid \text{len}(s) = k \} = \{ w^M \mid M \vDash Acc(w,k) \} \]

• $Th(A_r)$ is added into the background theory $T$
From regex to FSA

• Example:

Regrex: \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($r$)

$? (\d{1,3}, ?(\d{3}, ?) \ast \d{3}) (\ast \d{0,2}) ? | \d{1,3} (\ast \d{0,2}) ? | \ast \d{1,2} ?$

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

• The background universe is *multi-sorted*
  
  — Basic sorts:
    • Integers, rational numbers, bit-vectors, Booleans (\(\mathbb{B}\)),
  
  — Algebraic datatypes:
    • Lists: \(L\langle\sigma\rangle\) where \(\sigma\) is a sort
      
      ~ Constructor: \(nil: L\langle\sigma\rangle\), \(cons: \sigma \times L\langle\sigma\rangle \rightarrow L\langle\sigma\rangle\)
      
      ~ Accessors: \(hd: L\langle\sigma\rangle \rightarrow \sigma\), \(tl: L\langle\sigma\rangle \rightarrow L\langle\sigma\rangle\)
    • 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:
  - $Acc_q : \mathbb{L}\langle \mathbb{C} \rangle \times \mathbb{N} \rightarrow \mathbb{B}$, let $Acc$ be $Acc_{q0}$ were $q0$ is the initial state
  - Define the axioms (if $q$ is a final state, similarly for other states)
    - $\forall s \ (Acc_q(s, 0) \iff s = \text{nil})$
    - $\forall s \ n \ (Acc_q(s, \text{succ}(n)) \iff s \neq \text{nil} \land$
      $\left( (\varphi_1[\text{hd}(s)] \land Acc_{q1}(\text{tl}(s),n)) \lor \cdots \lor (\varphi_m[\text{hd}(s)] \land Acc_{qm}(\text{tl}(s),n)) \right)$

Note: $\varepsilon$-moves add additional disjuncts in rhs where $\text{succ}(n)$ 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)
  
  - $\forall s \ n \ Acc_q(s,\text{succ}(n)) \Leftrightarrow hd(s)=a \land Acc_p(tl(s),n)$
  
  - $\forall s \ Acc_p(s,0) \Leftrightarrow s=\text{nil}$  
    (p is final)
  
  - $\forall s \ n \ Acc_p(s,\text{succ}(n)) \Leftrightarrow false$  
    (p has no outgoing moves)

\[
\{a\} = \{x^M \mid M \models x=a\}
\]
Equational axioms have the form
\[ \forall x \ (lhs[x] = rhs[x]) \]
(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):

\[
\begin{align*}
\text{current goal} &= t \\
\text{new goal} &= rhs\theta
\end{align*}
\]

In general, axioms may also be nonequational and are triggered by associated patterns.
Step-by-step example (*E*-matching)

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

**E-matching:**

$$Acc_q(w, succ(0))$$

$$hd(w)=a \land Acc_p(tl(w),0)$$

$$hd(w)=a \land tl(w)=nil$$

$(\exists \theta = \{s \mapsto w, n \mapsto 0\}) \ Acc_q(w, succ(0)) = Acc_q(s, succ(n))\theta$

**Thus:** replace $Acc_q(w, succ(0))$ with $hd(s)=a \land Acc_p(tl(s),n)\theta$

$(\exists \theta = \{s \mapsto tl(w)\}) \ Acc_p(tl(w),0) = Acc_p(s,0)\theta$

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

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

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

(For FSAs both problems arise with $\varepsilon$-loops)

\[ \varphi = t \]

\[ \varphi = t \]
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
  - $Acc_q : L\langle C \rangle \times L\langle Z \rangle \times N \rightarrow B$, where $L\langle Z \rangle$ is a stack where $Z$ 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 $\varepsilon$-loops. $Th(A) \land Acc^A(s,k)$ is sat. $\iff s \in L(A)$ and $\text{len}(s) = k$.


• 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 $\varepsilon$-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 $Acc^A(s,k) \land Acc^B(s,k) \iff Acc^{A \times B}(s,k)$
    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>
<thead>
<tr>
<th>#</th>
<th>Regex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>\w+([-+.])\w+*@\w+([-.]\w+)*.\w+([-.]\w+)*([,;]\s*\w+([-.]\w+)*@\w+([-.]\w+)*.\w+([-.]\w+)*</code></td>
</tr>
<tr>
<td>2</td>
<td><code>s?\{d\{1,3\},?,d\{3\},?,d\{3\}\}d\{3\}\{d\{0,2\}\}d\{1,3\}\{d\{0,2\}\}d\{1,2\}?</code></td>
</tr>
<tr>
<td>3</td>
<td>`([A-Z]{2}</td>
</tr>
<tr>
<td>5</td>
<td><code>\w[-]+\{[0-9]*\}([0-9]+\{[0-9]*\})\{([\w\d]+\{-\})\{[0-9]?\}\{([\d\w]+\{-\})\{[0-9]\}\}</code></td>
</tr>
<tr>
<td>6</td>
<td>`(\w</td>
</tr>
<tr>
<td>7</td>
<td>`(\w</td>
</tr>
<tr>
<td>8</td>
<td>`(\w</td>
</tr>
<tr>
<td>9</td>
<td>`(\w</td>
</tr>
<tr>
<td>10</td>
<td>`(\w</td>
</tr>
</tbody>
</table>

## Evaluation results for sample regexes.

<table>
<thead>
<tr>
<th></th>
<th>εFSA(r)</th>
<th>FSA(r)</th>
<th>DFSA(r)</th>
<th>mDFSA(r)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>size</td>
<td>t ms</td>
<td>size</td>
<td>t ms</td>
</tr>
<tr>
<td>#1</td>
<td>91</td>
<td>100</td>
<td>73</td>
<td>40</td>
</tr>
<tr>
<td>#2</td>
<td>90</td>
<td>10</td>
<td>64</td>
<td>10</td>
</tr>
<tr>
<td>#3</td>
<td>83</td>
<td>10</td>
<td>70</td>
<td>10</td>
</tr>
<tr>
<td>#4</td>
<td>45</td>
<td>40</td>
<td>35</td>
<td>60</td>
</tr>
<tr>
<td>#5</td>
<td>98</td>
<td>100</td>
<td>71</td>
<td>10</td>
</tr>
<tr>
<td>#6</td>
<td>31</td>
<td>0</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>#7</td>
<td>2728</td>
<td>840</td>
<td>920</td>
<td>1800</td>
</tr>
<tr>
<td>#8</td>
<td>2816</td>
<td>60</td>
<td>269</td>
<td>60</td>
</tr>
<tr>
<td>#9</td>
<td>1944</td>
<td>280</td>
<td>2128</td>
<td>260</td>
</tr>
<tr>
<td>#10</td>
<td>112</td>
<td>30</td>
<td>104</td>
<td>30</td>
</tr>
</tbody>
</table>
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

$\varphi = \text{Cond}(t_1) \land \text{Cond}(t_2)$,

let $p_1 = \text{Target}(t_1)$, and

let $p_2 = \text{Target}(t_2)$. If $\varphi$ 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).

(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 \textit{not} a subset of Regex $R$?

• Does $Acc^G(x,k) \land \neg Acc^R(x,k)$ have a model $M$?
  – If \textit{yes}, $x^M$ is a \textit{witness} of length $k$
  – \textbf{Equivalently:} is $Acc^{G \cap R}(x,k)$ satisfiable?
Some related work

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?