Synthesis Applications

Alvin Cheung
Computer Science and Artificial Intelligence Laboratory
M.I.T.

with slides from Prof. Solar-Lezama and Rishabh Singh

December 11, 2013
harness vs implements

```c
int count=0;
int twox(int x, int y){
    ++count;
    return x + x;
}

int expr1(int x, int y) { return x*?? + y*??; }

int expr2(int x, int y) implements twox{
    count = count+??;
    return x*?? + y*??;
}

harness void foo(){
    assert expr1(5,2)*expr2(2,4)== 24;
}
```
Recap: Sketch internals

- Language support for defining $\{P_c\}$
  - Simple integer hole (??) + generators
- Constraints generated incrementally by CEGIS (CounterExample Guided Inductive Synthesis)
Example: Population count

0010 0110 → 3

```c
int pop (bit[W] x)
{
    int count = 0;
    for (int i = 0; i < W; i++) {
        if (x[i]) count++;
    }
    return count;
}
```

```c
int popSketched (bit[W] x)
{
    implements pop {
        repeat(??) {
            x = (x & ??) + ((x >> ??) & ??);
        }
        return x;
    }
}
```
Example: Population count
0010 0110 $\rightarrow$ 3

```c
int pop (bit[W] x) {
    int count = 0;
    for (int i = 0; i < W; i++) {
        if (x[i]) count++;
    }
    return count;
}
```

\[ F(x) = \]
Example: Population count
0010 0110 \rightarrow 3

\textbf{int} \ \text{popSketched} \ (\text{bit}[W] \ x) \ \text{implements} \ \text{pop} \ \{ \\
\text{repeat}(??) \ { \\
\Rightarrow \ \ x = (x \ & \ ??) \ \\
\Rightarrow \ + \ ((x \ >> \ ??) \ & \ ??); \\
\Rightarrow \ } \\
\Rightarrow \ \text{return} \ x; \\
\}
Using Program Synthesis for Social Recommendations

Alvin Cheung
Armando Solar-Lezama
Samuel Madden

CIKM 2012
Building social networking apps using phone data

• Mobile phones now come with various sensors
  – Accelerometer
  – GPS
  – Proximity

• Data collected from sensors can be used to automatically generate events
  – Enables new social networking apps
LifeJoin: Automatic generation of interesting events

Learned model

Inferred events

Event Ratings
Learning example

• Given labeled data:

<table>
<thead>
<tr>
<th>User</th>
<th>Location</th>
<th>Time</th>
<th>Interested?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>Office</td>
<td>10am</td>
<td>N</td>
</tr>
<tr>
<td>Bill</td>
<td>Home</td>
<td>3pm</td>
<td>N</td>
</tr>
<tr>
<td>Joe</td>
<td>Office</td>
<td>11pm</td>
<td>Y</td>
</tr>
<tr>
<td>Joe</td>
<td>Bar</td>
<td>6am</td>
<td>Y</td>
</tr>
</tbody>
</table>

• Possible classifier:
  (User = Joe) and
  (location = Office or location = Bar) and
  (time < 7 am or time > 10pm)
Applying machine learning to LifeJoin

• Does not create decomposable models

(User = Joe) and
(location = Office or location = Bar) and
(time < 7 am or time > 10pm) vs. 0.25x₁ + 0.65x₂ > 0

• Need to encode inputs into feature space representation
  – Create substantial time and space overhead

• Events are highly personalized
  – Hard to leverage labeled data from others
Applying synthesis to LifeJoin

Input-output spec
Interest([Peter, jog, Charles, 5PM]) = Y
Interest([Mary, office, 9AM]) = N
...

Search space grammar
interest(e) { act(e) | loc(e) |
act(e) & loc(e) | ...}

act(e) ::= e.user = u |
e.activity = a |
e.time = t |
act(e) & act(e) |
...
Our new hybrid approach

Labeled data → Interest grammar → Program Synthesizer

Interest functions

- user = Peter and activity = jog
- user ≠ Mary and time > 4PM
Our new hybrid approach

Labeled data
Interest grammar

Program Synthesizer

SVM features
- user = Peter
- activity = jog
- user ≠ Mary
- time > 4PM

Labeled data

SVM

SVM model
- SVs

Interest grammar

Program Synthesizer

Symbolic encoding of search space

Generalization guarantees

Active learning

Decomposable model

user = Peter and time > 4PM
Experimental results: active learning

Hybrid approach has a much faster learning rate
Optimizing Database-Backed Applications Using Query Synthesis

Alvin Cheung
Armando Solar-Lezama    Samuel Madden
PLDI 2013
List getUsersWithRoles () {
    List users = User.getAllUsers();
    List roles = Role.getAllRoles();
    List results = new ArrayList();
    for (User u : users) {
        for (Role r : roles) {
            if (u.roleId == r.id)
                results.add(u); } }
    return results; }

convert to

List getUsersWithRoles () {
    return executeQuery(
        "SELECT u FROM user u, role r
        WHERE u.roleId == r.id
        ORDER BY u.roleId, r.id"; }

SELECT * FROM user
SELECT * FROM role
Relational Operations in Imperative Code

List getUsersWithRoles () {
    List users = User.getAllUsers();
    List roles = Role.getAllRoles();
    List results = new ArrayList();
    for (User u : users) {
        for (Role r : roles) {
            if (u.roleId == r.id)
                results.add(u); }
    }
    return results; }

Goal
Find a post-condition that we can rewrite into a SQL expression

convert to

List getUsersWithRoles () {
    return executeQuery(
        "SELECT   u FROM user u, role r
        WHERE    u.roleId == r.id
        ORDER BY u.roleId, r.id"; }

output variable
Query By Synthesis (QBS)

• Identify potential code fragments
  – i.e., regions of code that fetches persistent data and return values

• Find SQL exprs for output variables

• Try to prove that those expressions preserve program semantics
  – if so, convert the code!
Theory of Ordered Relations (TOR)

• Similar to relational algebra
• Model relations as ordered lists

L := program var
| []
| L : L | L : e
| top_e(L)
| L \f L | \sigma_f(L)
| \pi_f(L) | order_e(L)

e := L[i]
| e op e
| max(L) | min(L)
| sum(L) | avg(L)
| size(L)
• Semantics defined using axioms, e.g.:

\[
\begin{align*}
\text{top}_i([]) &= [] \\
\text{top}_i(L) &= [] & \text{if } i = 0 \\
\text{top}_i(h : L) &= h : \text{top}_{i-1}(L) & \text{if } i > 0
\end{align*}
\]
• Standard Hoare logic rules

• Treat loop invariants and post-conditions for output variables as function calls
  – Leave function bodies to be synthesized
Example

List getUsersWithRoles () {
    List users = query(select * from users);
    List roles = query(select * from roles);
    List results = [];
    for (User u : users) {
        for (Role r : roles) {
            if (u.roleId == r.id)
                results = results : []
        }
    }
    return results;
}

Verification conditions

assume(preCondition = true)

preCondition →
    outerInvariant(users/query(...), results/[], ...)

outerInvariant(...) ∧ outer loop terminates →
    results = postCondition(users, roles) ...

results = postCondition(users, roles)

outerInvariant(users, roles, u, results, ...)

innerInvariant(users, roles, u, r, results, ...)

Verification conditions
Synthesis Templates for Invariants and Post-conditions

• Template for invariants:
\[ \land (\text{<variable in scope> } = \text{<TOR expr>}) \]
  – Only consider expressions that type check

• Template for post-conditions:
\[ \text{<output variable> } = \text{<TOR expr>} \]
  – Limit to TOR expressions that are translatable to SQL
## Real-world Evaluation

**Wilos** (project management application) – 62k LOC

<table>
<thead>
<tr>
<th>Operation type</th>
<th># Fragments found</th>
<th># Fragments converted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projection</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Selection</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>Join</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Aggregation</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>33</strong></td>
<td><strong>28</strong></td>
</tr>
</tbody>
</table>
Performance Evaluation: Join Query

Nested-loop join $\Rightarrow$ Hash join!

$O(n^2)$ $\Rightarrow$ $O(n)$
Synthesizing Data Structure Manipulations from Storyboards

Rishabh Singh and Armando Solar-Lezama

FSE 2011
dllDelete(Node v) {
    n  = v.next;
    p  = v.prev;
    p.next = n;
    n.prev = p;
}
Bridging the semantic gap

dllDelete(Node v) {
    n = v.next;
    p = v.prev;
    p.next = n;
    n.prev = p;
}

WHAT  HOW

Program Synthesis
Idea:

combine many simple specs to constrain behavior
Linked List Reversal
Scenarios for LL-reversal
Inductive structure with fold/unfold

Unfold:

\[ x' = f \]
\[ x' = e \]
\[ e = e' \]
Inductive structure with fold/unfold

Unfold:

Fold:
void llReverse(Node head)
{
    /*1*/
    while /*p*/
    {
        /*2*/
    }
    /*3*/
}
void llReverse(Node head)
{
    cstmt* /*1*/
    while (cond /*p*/)
    {
        cstmt* /*2*/
    }
    cstmt* /*3*/
}
Conditional Statements

\[ \text{var(.ptr?) op var(.ptr?) | null} \]

\[ \text{cstmt : if(COND) then STMT} \]

\[ \text{var(.ptr?) = var(.ptr?)} \]

\[ \text{unfold/fold var} \]
Recap: Storyboard Constraints

Abstract Scenarios

Concrete Scenarios

void llReverse(Node head)
{
  /*1*/
  while /*p*/
  {
    /*2*/
  }
  /*3*/
}

Structural Info
## Case studies - II

<table>
<thead>
<tr>
<th>Manipulation</th>
<th>#Clauses</th>
<th>Memory</th>
<th>#Scen</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bst-search(contains)</td>
<td>0.62M</td>
<td>0.37G</td>
<td>1</td>
<td>1m02s</td>
</tr>
<tr>
<td>bst-search</td>
<td>0.77M</td>
<td>0.45G</td>
<td>1</td>
<td>6m07s</td>
</tr>
<tr>
<td>bst-find-min</td>
<td>0.63M</td>
<td>0.45G</td>
<td>1</td>
<td>0m58s</td>
</tr>
<tr>
<td>bst-find-max</td>
<td>0.57M</td>
<td>0.18G</td>
<td>1</td>
<td>0m23s</td>
</tr>
<tr>
<td>bst-left-rotate</td>
<td>1.41M</td>
<td>0.50G</td>
<td>3</td>
<td>3m18s</td>
</tr>
<tr>
<td>bst-right-rotate</td>
<td>1.47M</td>
<td>0.43G</td>
<td>3</td>
<td>3m15s</td>
</tr>
<tr>
<td>bst-insertion</td>
<td>1.04M</td>
<td>0.46G</td>
<td>3</td>
<td>1m52s</td>
</tr>
<tr>
<td>bst-deletion</td>
<td>0.63M</td>
<td>0.62G</td>
<td>6</td>
<td>3m13s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Manipulation</th>
<th>#Clauses</th>
<th>Memory</th>
<th>#Scen</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>aig-insertion</td>
<td>0.17M</td>
<td>0.31G</td>
<td>4</td>
<td>1m04s</td>
</tr>
</tbody>
</table>
"Not only did it take 1-2 weeks to grade problem, but the comments were entirely unhelpful in actually helping us fix our errors. ....Apparently they don't read the code – they just ran their tests and docked points mercilessly. What if I just had a simple typo, but my algorithm was fine? ...."
Scalability Challenges (>100k students)

Bigger Challenge in MOOCs
def computeDeriv(poly):
    deriv = []
    zero = 0
    if (len(poly) == 1):
        return deriv
    for e in range(0, len(poly)):
        if (poly[e] == 0):
            zero += 1
        else:
            deriv.append(pol y[e]*e)
    return deriv

def computeDeriv(poly):
    deriv = []
    zero = 0
    if (len(poly) == 1):
        return deriv
    for e in range(0, len(poly)):
        if (poly[e] == 0):
            zero += 1
        else:
            deriv.append(pol y[e]*e)
    return deriv
def computeDeriv(poly):
    deriv = []
    zero = 0
    if (len(poly) == 1):
        return deriv
    for e in range(0, len(poly)):
        if (poly[e] == 0):
            zero += 1
        else:
            deriv.append(poly[e] * e)
    return deriv

replace derive by [0]

Teacher’s Solution

Error Model
Technical Challenges

Large space of possible corrections

Minimal corrections

Dynamically-typed language

Constraint-based Synthesis to the rescue
computeDeriv

Compute the derivative of a polynomial

poly = [10, 8, 2]  # f(x) = 10 + 8x + 2x^2
=> [8, 4]     # f'(x) = 8 + 4x
def computeDeriv(poly):
    result = []
    if len(poly) == 1:
        return [0]
    for i in range(1, len(poly)):
        result += [i * poly[i]]
    return result
Simplified Error Model

- \( \text{return } a \Rightarrow \text{return } \{[0], ?a\} \)
- \( \text{range}(a_1, a_2) \Rightarrow \text{range}(a_1+1, a_2) \)
- \( a_0 == a_1 \Rightarrow \text{False} \)
Rewriting using Error Model

\[ \text{range}(0, \text{len}(\text{poly})) \]

\[ \text{range}([0, 1], \text{len}(\text{poly})) \]

**default choice**

\[ a \rightarrow a+1 \]
Rewriting using Error Model

\[ \text{range}(0, \text{len}(\text{poly})) \]

\[ \text{range}([0, 1], \text{len}(\text{poly})) \]

\[ a \rightarrow a+1 \]
Rewriting using Error Model

\[
\text{range}(0, \text{len}(\text{poly}))
\]

\[
\text{range}\left(\{0, 1\}, \text{len}\left(\{\text{poly, poly+1}\}\right)\right)
\]

\[a \rightarrow a+1\]
Rewriting using Error Model

$$\text{range}(0, \text{len}(\text{poly}))$$

$$\text{range}([0, 1], \{\text{len}([\text{poly}, \text{poly+1}]), \text{len}([\text{poly}, \text{poly+1}])+1\})$$

$$a \rightarrow a + 1$$
def computeDeriv(poly):
    deriv = []
    zero = 0
    if ({len(poly) == 1, False}):
        return {deriv,[0]}
    for e in range({0,1}, len(poly)):
        if (poly[e] == 0):
            zero += 1
        else:
            deriv.append(poly[e]*e)
    return {deriv,[0]}

Problem: Find a program that minimizes cost metric and is functionally equivalent with teacher’s solution
Translation to Sketch

(1) Handling python’s dynamic types

(2) Translation of expression choices
(1) Handling Dynamic Types

```c
struct MultiType{
    int type;
    int ival; bit bval;
    MTString str; MTList lst;
    MTDict dict; MTTuple tup;
}
```
Python Constants using MultiType

\[ [1,2] \rightarrow \text{LIST} \]

\[ \text{len=2, lVals = \{*, *\}} \]
Python Exprs using MultiType

```
x + y →
```

```python
5 + 10 = 15
```

```
5
7
```

```
10
```

```
15
```
Python Exprs using MultiType

\[ x + y \rightarrow \]

\[
\begin{array}{c}
\text{int} \\
\text{list} \\
[1,2] \\
\text{LIST} \\
\text{bool}
\end{array}
\] 
\[+\]

\[
\begin{array}{c}
\text{int} \\
\text{list} \\
[3] \\
\text{LIST} \\
\text{bool}
\end{array}
\] 

= 

\[
\begin{array}{c}
\text{int} \\
\text{list} \\
[1,2,3] \\
\text{LIST} \\
\text{bool}
\end{array}
\]
Python Expressions using MultiType

Typing rules are encoded as constraints

\[ x + y \rightarrow \]

\[ \text{INT} \] + \[ \text{LIST} \] = \[ \text{X} \]
harness main(int N, int[N] poly){

    MultiType polyMT = MTFFromList(poly);

    MultiType result\_1 = computeDeriv\_teacher(polyMT);
    MultiType result\_2 = computeDeriv\_student(polyMT);

    assert MTEquals(result\_1,result\_2);
}

harness main(int N, int[N] poly){
    totalCost = 0;
    MultiType polyMT = MTFromList(poly);

    MultiType result_1 = computeDeriv_teacher(polyMT);
    MultiType result_2 = computeDeriv_student(polyMT);

    .......... if(choice_k) totalCost++;
    .......... assert MTEquals(result_1, result_2);
    minimize(totalCost);
}
Incremental Solving for minimize(x)

(\(P, x\)) \rightarrow \text{Sketch} \rightarrow (\(P_1, x=7\))

\{x<7\} \downarrow \uparrow

\text{Sketch} \Rightarrow \{x<4\}

\text{UNSAT} \leftarrow \text{Sketch} \leftarrow (\(P_2, x=4\))
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>evalPoly-6.00</td>
<td>13</td>
</tr>
<tr>
<td>compBal-stdin-6.00</td>
<td>52</td>
</tr>
<tr>
<td>compDeriv-6.00</td>
<td>103</td>
</tr>
<tr>
<td>hangman2-6.00x</td>
<td>218</td>
</tr>
<tr>
<td>prodBySum-6.00</td>
<td>268</td>
</tr>
<tr>
<td>oddTuples-6.00</td>
<td>344</td>
</tr>
<tr>
<td>hangman1-6.00x</td>
<td>351</td>
</tr>
<tr>
<td>evalPoly-6.00x</td>
<td>541</td>
</tr>
<tr>
<td>compDeriv-6.00x</td>
<td>918</td>
</tr>
<tr>
<td>oddTuples-6.00x</td>
<td>1756</td>
</tr>
<tr>
<td>iterPower-6.00x</td>
<td>2875</td>
</tr>
<tr>
<td>recurPower-6.00x</td>
<td>2938</td>
</tr>
<tr>
<td>iterGCD-6.00x</td>
<td>2988</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TestSet</th>
<th>Generated Feedback</th>
<th>Percentage</th>
<th>AvgTime(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>13365</td>
<td>8579</td>
<td>64.19%</td>
<td>9.91</td>
</tr>
</tbody>
</table>
Learning Semantic String Transformations from Examples

Rishabh Singh and Sumit Gulwani
VLDB 2012
Transformations

- **Syntactic Transformations**
  - Concatenation of regular expression based substring
    - “VLDB2012” $\rightarrow$ “VLDB”

- **Semantic Transformations**
  - More than just characters
    - “1/5/2010” $\rightarrow$ “May 1$^{st}$ 2010”
Semantic Transformations

- Semantic information as relational tables
  - 1 $\rightarrow$ January, 2 $\rightarrow$ February

- Learn table lookup queries
  - VLOOKUP macro 2nd most problematic
Lookup Transformation Language

$L_t$

Expression $e_t$ : $v_i$
| $\text{Select}(C, T, b)$

Boolean Condition $b$ : $p_1 \land \ldots \land p_n$

Predicate $p$ : $C = s$
| $C = e_t$
Example - Lookup

Select(Name, EmpRecord, (SSN = v₁))

<table>
<thead>
<tr>
<th>Input $v₁$</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>044-58-3429</td>
<td>Steve Russell</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Emp Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>027-36-4557</td>
</tr>
<tr>
<td>034-83-7683</td>
</tr>
<tr>
<td>044-58-3429</td>
</tr>
<tr>
<td>018-45-8949</td>
</tr>
<tr>
<td>023-34-3254</td>
</tr>
</tbody>
</table>
Example – Transitive Lookup

\[
\text{Select(Price, PriceRec, (ItemId = Select(ItemId, ItemRec, Item = v_1))}
\]

<table>
<thead>
<tr>
<th>Input v_1</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stroller</td>
<td>$145.67</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ItemRec</th>
<th>PriceRec</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ItemId</strong></td>
<td><strong>Item</strong></td>
</tr>
<tr>
<td>ST-340</td>
<td>Stroller</td>
</tr>
<tr>
<td>BI-567</td>
<td>Bib</td>
</tr>
<tr>
<td>DI-328</td>
<td>Diapers</td>
</tr>
<tr>
<td>WI-989</td>
<td>Wipes</td>
</tr>
<tr>
<td>AS-469</td>
<td>Aspirator</td>
</tr>
</tbody>
</table>
Strings reachable from input row

\[ \text{GenerateStr}_t \]

\[ \eta_1 \quad \eta_2 \quad \eta_3 \]

\[ \text{Progs}[\eta_1] = \{v_1\} \]

<table>
<thead>
<tr>
<th>SSN</th>
<th>EmpId</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>027-36-4557</td>
<td>1254</td>
<td>John Henry</td>
</tr>
<tr>
<td>034-83-7683</td>
<td>2412</td>
<td>William Johnson</td>
</tr>
<tr>
<td>044-58-3429</td>
<td>1125</td>
<td>Steve Russell</td>
</tr>
<tr>
<td>018-45-8949</td>
<td>4257</td>
<td>Ian Jordan</td>
</tr>
</tbody>
</table>
GenerateStr\textsubscript{t}

\[ Prgs[\eta] \cup Select(C, T, B) \]

\[ B \equiv \left\{ \left( \land C_i = \{val^{-1}(T[C_i, r])\} \right) \right\}_j \]

strings in table rows of visited nodes

044-58-3429  1125  Steve Russell
GenerateStr$^t$

Repeat until k steps or fixpoint
\textbf{GenerateStr}_t

\[ \eta \quad \text{Progs}[\eta] \]

Steve Russell
Data structure $D_t$

- Maintains tree structure
  - share common sub-expressions

- CNF of Boolean Conditionals
  - independent column predicates
Intersect_{t} : D_{t_1} \land D_{t_2}
Syntactic String Language $L_S$

[GulwaniPOPL11]

$$
e_s := \text{Concatenate}(f_1, \ldots, f_n) \mid f$$

Atomic expr $f$ := $\text{ConstStr}(s) \mid v_i \mid \text{SubStr}(v_i, p_1, p_2)$

Position $p$ := $k \mid \text{pos}(r_1, r_2, c)$

Integer expr $c$ := $k \mid k_1 \omega + k_2$

Regular expr $r$ := $\epsilon \mid \tau \mid \text{TokenSeq}(\tau_1, \ldots, \tau_n)$
Combined Language $L_u$

Syntactic manipulations over lookup outputs

Atomic expr $f_s := \text{ConstStr}(s) | e_t$
$| \text{SubStr}(e_t, p_{s_1}, p_{s_2})$

Predicate $p_t := C = s | C = e_s$

$e_s := \text{Concatenate}(f_{s_1}, \ldots, f_{s_n}) | f_s$

Syntactic manipulations before indexing
SSN: 044-58-3429 → Mr. Steve Russell

<table>
<thead>
<tr>
<th>Emp Record</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SSN</strong></td>
</tr>
<tr>
<td>027-36-4557</td>
</tr>
<tr>
<td>034-83-7683</td>
</tr>
<tr>
<td>044-58-3429</td>
</tr>
<tr>
<td>018-45-8949</td>
</tr>
</tbody>
</table>
### GenerateStr\_u

<table>
<thead>
<tr>
<th>SSN</th>
<th>EmpId</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>027-36-4557</td>
<td>1254</td>
<td>John Henry</td>
</tr>
<tr>
<td>034-83-7683</td>
<td>2412</td>
<td>William Johnson</td>
</tr>
<tr>
<td><strong>044-58-3429</strong></td>
<td>1125</td>
<td>Steve Russell</td>
</tr>
<tr>
<td>018-45-8949</td>
<td>4257</td>
<td>Ian Jordan</td>
</tr>
</tbody>
</table>

### GenerateStr\'_t

SSN: 044-58-3429
<table>
<thead>
<tr>
<th>SSN</th>
<th>EmpId</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>027-36-4557</td>
<td>1254</td>
<td>John Henry</td>
</tr>
<tr>
<td>034-83-7683</td>
<td>2412</td>
<td>William Johnson</td>
</tr>
<tr>
<td>044-58-3429</td>
<td>1125</td>
<td>Steve Russell</td>
</tr>
<tr>
<td>018-45-8949</td>
<td>4257</td>
<td>Ian Jordan</td>
</tr>
</tbody>
</table>
Experiments

• 50 benchmark problems
  – 12 $L_t$, 38 $L_u$

• $\sim 10^{20}$ consistent expressions
  – Size of data structure: $\sim 2000$

• Performance: 96% less than 1 second

• Ranking: at most 3 examples (95% 2 examples)