Compact Representation for Answer Sets of n-ary Regular Queries

by Kazuhiro Inaba (National Institute of Informatics, Japan) and Hauro Hosoya (The University of Tokyo)

CIAA 2009, Sydney
N-ary Query over Trees

- ... is a function that
  - Takes a tree $t$ as an input, and
  - Returns some set of n-tuples of nodes of $t$
- Examples:
  - 1-ary: “select the leftmost leaf node”
  - 2-ary: “select all pairs $(x, y)$ s.t. $x$ is taken from the left subtree of the root, and $y$ is from the right”
  - 0-ary: “is the number of leaves odd?”
**BACKGROUND**

**N-ary Regular Queries over Trees**

- Query definable by a tree automaton

- Regular
  - iff definable by Monadic 2\textsuperscript{nd}-Order Logic
  - iff definable by Modal $\mu$-Calculus
  - iff definable by Monadic Datalog
  - iff definable by Boolean Attribute Grammar
  - ...
Efficiency of Regular Queries

- Given a query $q$ (represented by an automaton $A$), and an input tree $t$, we can compute $q(t)$ in...

$$O(|A| \cdot (|t| + |q(t)|))$$

[Flum & Frick & Grohe 2002]

- (In some sense) optimal
“Optimal”, but…

- $O\left( |A| \cdot (|t| + |q(t)|) \right)$
  - $\sim |t|^n$ for n-ary queries in the worst case

- In some applications, we do not need the concrete list of all the answers
  - At least, don’t need to list up them all at the same time
Input Tree
size: IN
height: H

Run Query
O(IN + OUT)

Set of Output Tuples
size: \( OUT \in O(IN^n) \)

"SRED" Data structure
size: \( O(\min(IN,OUT)) \)

Today's Topic

- Run Query: \( O(IN) \)
- Enum: \( O(OUT) \)
- isMember: \( O(H) \)
- Get-Size: \( O(\min(I,O)) \)
- "Projection": \( O(H \cdot \alpha) \)
- "Selection": \( O(H) \)
APPLICATION
(IN XML TRANSLATION)
“Relative” Queries in XML

- Select y relative to x
  - In many cases, # of y for each x is constant. E.g.,
    - “select the child labeled <name>”, “select next <h2>”

```xml
<list>
  {foreach x s.t. φ(x):
   <item>{x}</item>
   <sublist>
     {foreach y s.t. ψ(x,y):
      <item>{y}</item>}</item>
   </sublist>}
</list>
```
Two Evaluation Strategies

A := the answer set of 1-ary query \{x \mid \Phi(x)\}
for each x in A:
  B := the answer set of 1-ary query \{y \mid \Psi(x,y)\}
  for each y in B:
    print <item>y</item>

C := the answer set of 2-ary query \{(x,y) \mid \Phi(x) \& \Psi(x,y)\}
for each x in A:
  B := \{y \mid (x,y) \in C\} = C_{[1:x]}
  for each y in B:
    print <item>y</item>;
A := the answer set of \(1\)-ary query \(\{x \mid \Phi(x)\}\)

for each \(x\) in \(A\):
B := the answer set of \(1\)-ary query \(\{y \mid \Psi(x,y)\}\)
for each \(y\) in \(B\):
print <item>y</item>

A := the answer set of \(1\)-ary query \(\{x \mid \Phi(x)\}\)

C := the answer set of \(2\)-ary query \(\{(x,y) \mid \Phi(x) \& \Psi(x,y)\}\)
for each \(x\) in \(A\):
B := \(\{y \mid (x,y) \in C\}\) = \(C_{[1:x]}\)
for each \(y\) in \(B\):
print <item>y</item>;

\(O(|t|^2)\) time in “common” cases (= many \(x\), constant \(y\))

\(O(|t|)\) time in “common” cases
A := the answer set of 1-ary query \( \{ x \mid \Phi(x) \} \) for each \( x \) in A:

B := the answer set of 1-ary query \( \{ y \mid \Psi(x,y) \} \) for each \( y \) in B:

print \(<item>y</item>\)

\[
\begin{align*}
A := & \text{the answer set of 1-ary query } \{ x \mid \Phi(x) \} \\
B := & \text{the answer set of 1-ary query } \{ y \mid \Psi(x,y) \} \\
\text{for each } x \text{ in A:} \\
\text{for each } y \text{ in B:} \\
\text{print } \langle \text{item}y\text{/item} \rangle
\end{align*}
\]

\[O(|t|^2) \text{ time in "common" cases} = \text{many } x, \text{ constant } y\]

\[O(|t|) \text{ time in "common" cases} = \text{many } x, \text{ constant } y\]

\[O(|t|^2) \text{ time in "worst" cases} = \text{many } x, \text{ many } y\]

\[O(|t|) \text{ space in "worst" cases} = \text{many } x, \text{ many } y\]
### Two Evaluation Strategies

A := the answer set of 1-ary query \( \{x | \Phi(x)\} \) for each \( x \) in A:

1. B := the answer set of 1-ary query \( \{y | \Psi(x,y)\} \) for each \( y \) in B:
2. print \(<item>y</item>\)

#### Time and Space Complexity

- **O(|t|²) time** in “common” cases (= many \( x \), constant \( y \))
- **O(|t|) space**
- **O(|t|²) time** in “worst” cases (= many \( x \), many \( y \))
- **O(|t|) space**

### If We Use “SRED” to Represent the Set \( C \) ...

A := the answer set of 1-ary query \( \{x | \Phi(x)\} \)

C := the answer set of 2-ary query \( \{(x,y) | \Phi(x) & \Psi(x,y)\} \)

for each \( x \) in A:

1. B := \( \{y | (x,y) \in C\} = C_{[1:x]} \)
2. for each \( y \) in B:
   - print \(<item>y</item>\)

#### Time and Space Complexity

- **O(|t|³) time** in “common” cases
- **O(|t|) space**
- **O(|t|²) time** in “worst” cases
- **O(|t|³) space**
IMPLEMENTATION OF REGULAR QUERIES USING “SRED”
(Bottom-up Deterministic) Tree Automaton

(For simplicity, we limit our attention to binary trees)

- $\mathcal{A} = (\Sigma_0, \Sigma_2, Q, F, \delta)$
  - $\Sigma_0$: finite set of leaf labels
  - $\Sigma_2$: finite set of internal-node labels
  - $Q$: finite set of states
  - $\delta$: transition function
    $$(\Sigma_0 \cup \Sigma_2 \times Q \times Q) \rightarrow Q$$
  - $F \subseteq Q$: accepting states
Example (0-ary): ODDLEAVES

- \( Q = \{q_0, q_1\}, \ F=\{q_1\} \)
- \( \delta(L) = q_1 \)
- \( \delta(B, q_0, q_0) = q_0 \)
- \( \delta(B, q_0, q_1) = q_1 \)
- \( \delta(B, q_1, q_0) = q_1 \)
- \( \delta(B, q_1, q_1) = q_0 \)
Tree Automaton for Querying

- For any n-ary regular query $\Phi$ on trees over $\Sigma_0 \cup \Sigma_2$,
- There exists a BDTA $A_\Phi$ on trees over $\Sigma_0 \times B^n, \Sigma_2 \times B^n$ (where $B=\{0,1\}$) s.t.
  - $(v_1, \ldots, v_n) \in \Phi(t)$
  - iff
  - $A_\Phi$ accepts the tree $\text{mark}(t, v_1, \ldots, v_n)$

- $\text{mark}(t, \ldots) = t$ with the i-th B component is 1 at $v_i$ and 0 at other nodes
Example (1-ary): \textbf{LEFTMOST}

- $Q = \{q_0, q_1\}$, $F = \{q_1\}$
- $\delta(L0) = q_0$
- $\delta(L1) = q_1$
- $\delta(B0, q_1, q_0) = q_1$
- $\delta(\text{otherwise}) = q_0$
NA: Naïve n-ary Query Algorithm

- For each tuple \((v_1, \ldots, v_n) \in \text{Node}(t)^n\)
  - Generate \(\text{mark}(t, v_1, \ldots, v_n)\)
  - Run \(A_\Phi\) on it
    - If accepted, then \((v_1, \ldots, v_n)\) is one of the answer

- Run \(A_\Phi\) on \(t\) \(O(|t|^n)\) times = \(O(|t|^{n+1})\)
OA: One-Pass Algorithm

- For each combination of node \( v \), state \( q \), and \( b_1, \ldots, b_n \in B \)
  - Compute the set
    \[ r_v (q, b_1, \ldots, b_n) \subseteq (\text{Node}(t) \cup \{ \bot \})^n \] s.t.
    \[ (v_1, \ldots, v_n) \in r_v (q, b_1, \ldots, b_n) \]
    iff
    \[ (\forall i : \text{“descendant } v_i \text{ of } v \text{ is marked and } b_i=1” \text{ or } “v_i=\bot \text{ and } b_i=0”) \Rightarrow “\text{automaton assigns } q \text{ at node } v” \]
Example (2-ary): LEFT&RIGHT

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $F=\{q_3\}$

- $\delta(L00) = \delta(B10, q_0, q_0) = q_0$
- $\delta(L10) = \delta(B10, q_0, q_0) = q_1$
- $\delta(L01) = \delta(B01, q_0, q_0) = q_2$
- $\delta(B00, q_1, q_2) = q_3$
- $\delta(B00, q_0, q_i) = \delta(B00, q_i, q_0) = q_i$ (for $i=1,2$)
- $\delta(\text{otherwise}) = q_4$
\[ \delta(L00) = \delta(B10, q_0, q_0) = q_0 \]
\[ \delta(L10) = \delta(B10, q_0, q_0) = q_1 \]
\[ \delta(L01) = \delta(B01, q_0, q_0) = q_2 \]
\[ \delta(B00, q_1, q_2) = q_3 \]
\[ \delta(B00, q_0, q_2) = q_2 \]
\[ \ldots \]
\[ \delta(L00) = \delta(B10, q_0, q_0) = q_0 \]
\[ \delta(L10) = \delta(B10, q_0, q_0) = q_1 \]
\[ \delta(L01) = \delta(B01, q_0, q_0) = q_2 \]
\[ \delta(B00, q_1, q_2) = q_3 \]
\[ \delta(B00, q_0, q_2) = q_2 \]

\[ r_{v2}(q_0, 00) = \{ (\bot, \bot) \} \]
\[ r_{v2}(q_1, 10) = \{ (v2, \bot) \} \]
\[ r_{v2}(q_2, 01) = \{ (\bot, v2) \} \]
\[ r_{v2}(q_4, 11) = \{ (v2, v2) \} \]
\[ r_{v2}(\_, \_) = \{} \]
\[\delta(L00) = \delta(B10, q_0, q_0) = q_0\]
\[\delta(L10) = \delta(B10, q_0, q_0) = q_1\]
\[\delta(L01) = \delta(B01, q_0, q_0) = q_2\]
\[\delta(B00, q_1, q_2) = q_3\]
\[\delta(B00, q_0, q_2) = q_2\]

\[r_{v2}(q_0, 00) = \{ (\bot, \bot) \} \]
\[r_{v2}(q_1, 10) = \{ (v2, \bot) \} \]
\[r_{v2}(q_2, 01) = \{ (\bot, v2) \} \]
\[r_{v2}(q_4, 11) = \{ (v2, v2) \} \]
\[r_{v2}(\_, \_) = \{ \} \]

\[r_{v3}(q_0, 00) = r_{v4}(q_0, 00) \cdot \{ (\bot, \bot) \} \cdot r_{v5}(q_0, 00) = \{ (\bot, \bot) \} \]
\[r_{v3}(q_3, 11) = r_{v4}(q_1, 00) \cdot \{ (\bot, \bot) \} \cdot r_{v5}(q_2, 11)\]
\[\cup r_{v4}(q_1, 01) \cdot \{ (\bot, \bot) \} \cdot r_{v5}(q_2, 10)\]
\[\cup r_{v4}(q_1, 10) \cdot \{ (\bot, \bot) \} \cdot r_{v5}(q_2, 01)\]
\[= \{ (v4, \bot) \} \cdot \{ (\bot, \bot) \} \cdot \{ (\bot, v5) \} \]
\[\cup r_{v4}(q_1, 11) \cdot \{ (\bot, \bot) \} \cdot r_{v5}(q_2, 01)\]
\[= \{ (v4, v5) \} \]

\[r_{v3}(q_2, 01) = r_{v4}(q_0, 00) \cdot \{ (\bot, v3) \} \cdot r_{v5}(q_0, 00)\]
\[\cup r_{v4}(q_0, 00) \cdot \{ (\bot, \bot) \} \cdot r_{v5}(q_2, 01)\]
\[\cup r_{v4}(q_2, 01) \cdot \{ (\bot, \bot) \} \cdot r_{v5}(q_2, 00)\]
\[= \{ (\bot, v3), (\bot, v4), (\bot, v5) \} \]

\[r_{v4}, r_{v5} : \text{similar to } r_{v2}\]
Example (2-ary): LEFT&RIGHT

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $F=\{q_3\}$
- Eventually...

$B \leftarrow v_1 \leftarrow v_2 \leftarrow v_3 \leftarrow v_4 \leftarrow v_5$

$r_{v_1}(q_3, 11) = \{(v_2,v_3), (v_2,v_4), (v_2,v_5)\}$

...
Time Complexity of OA: $O(|t|^{n+1})$

- One-pass traversal: $|t|$
- For each node,
  - $|Q| \times 2^n$ entries of r are filled
  - Need $O(|Q|^2 \cdot 3^n)$ $\cup$ and $\ast$ operations
  - Each operand set of $\cup$ and $\ast$ may be as large as $O(|t|^n)$
- $\Rightarrow$ each operation takes $O(|t|^n)$ time in the worst case, as long as the "set"s are represented by usual data structure (lists, rb-trees,...)
Time Complexity of OA: $O(|t|^{n+1})$

- One-pass traversal: $|t|$
- For each node, \( |Q| \times 2^n \) entries of \( r \) are filled
- Need \( O(|Q|^2 \cdot 3^n) \) \( \cup \) and \( \ast \) operations
- Each operand set of \( \cup \) and \( \ast \) may be as large as \( O(|t|^n) \)
- \( \ast \) operation takes \( O(|t|^n) \) time in the worst case, as long as the "set"s are represented by usual data structure (lists, rb-trees, …)

What happens if we have a set representation with \( O(1) \) operations??
Time Complexity of OA: $O(|t|^{n+1})$

- One-pass traversal: $|t|$
- For each node, Constant wrt $|t|$!
  - $|Q| \times 2^n$ entries of $r$ are filled
  - Need $O(|Q|^2 \cdot 3^n)$ $\cup$ and $\ast$ operations
  - Each operand set of $\cup$ and $\ast$ may be as large as $O( |t|^{n} )$

What happens if we have a set representation with $O(1)$ operations??

$O(|t|)$ Time Querying!
!! Our Main Idea !!

- **SRED:** Set Representation by Expression Dags
  - Set is Repr’d by a Symbolic Expression Producing it

Instead of \{\{(v2,v3), (v2,v4), (v2,v5)\}\}

We Use

\[\begin{align*}
\ast & \quad \{(v2, \bot)\} \\
\cup & \quad \{(\bot, v3)\} \\
\cup & \quad \{(\bot, v4)\} \\
\cup & \quad \{(\bot, v5)\}
\end{align*}\]
BNF for SRED (Simplified)

- **SET ::=**
  - Empty -- {}
  - Unit -- \{⊥,...,⊥\}
  - **NESET**

- **NESET ::=**
  - Singleton( **ELEMENT** )
  - DisjointUnion( **NESET, NESET** )
  - Product( **NESET, NESET** )
Properties of SRED

Input Tree
- Size: IN
- Height: H

Set of Output Tuples
- Size: OUT ∈ O(IN^n)

"SRED" Data Structure
- Size: O(min(IN, O(IN))

Run Query
- O(IN) time
- O(IN) space

Because, ∪ and * are almost trivially in O(1)
Properties of SRED

Input Tree
- size: \(\text{IN} \)
- height: \(H\)

Output Tuples
- size: \(\text{OUT} \in O(\text{IN}^n)\)

Data Structure
- Size:
  \[O(\min(\text{IN}, \text{OUT}))\]
- \(O(\text{IN})\) time
- \(O(\text{IN})\) space
- Thanks to empty-set elimination

Run Query
- \(O(\text{IN})\)

Thanks to empty-set elimination

Because, \(\cup\) and \(*\) are almost trivially in \(O(1)\)
Properties of SRED

Input Tree
size: IN
height: H

Run Query
O(IN)

Data Structure
"SRED"
Size:
O(min(IN, OUT))

O(IN) time \rightarrow O(IN) space

Because, $\cup$ and $*$ are almost trivially in $O(1)$

O(IN) time \rightarrow O(IN) space

Thanks to empty-set elimination

"Selection": $O(H)$

"Projection": $O(H \cdot \alpha)$

Get-Size: $O(\min(I, O))$

isMember: $O(H)$

Enum
O(OUT)

Set of Output Tuples
size: $OUT \in O(IN^n)$

Thanks to empty-set elimination

Very Easy to Derive
O(OUT) Enumeration of SRED
(or, “decompression”)

- Simple Recursion is Enough!
  (assumption: $\cup$ is O(1), $*$ is O(out) )
  - eval(Empty) = $\{\}$
  - eval(Unit) = $\{(\bot,\ldots,\bot)\}$
  - eval(Singleton(e)) = $\{e\}$
  - eval(DisjointUnion($s_1,s_2$)) = eval($s_1$) $\cup$ eval($s_2$)
  - eval(Product($s_1,s_2$)) = eval($s_1$) $*$ eval($s_2$)

- (NOTE: A bit more clever impl. enables O(OUT) time & O(1) working space)
For Advanced Operations...

- Actually we add a little more information on each SRED node
  - “Type”
  - “Origin”
“Selection” on SRED

- \( S_{[i:v]} = \{(v_1,\ldots,v_{i-1},v,v_{i+1},\ldots,v_n) \mid (v_1,\ldots,v_{i-1},v,v_{i+1},\ldots,v_n) \in S\} \)

- Again, Simple Recursion!

- \((S \cup T)_{[i:v]} = S_{[i:v]} \cup T_{[i:v]}\)

- \((S^{t1,u1} * T^{t2,v2})_{[i:v]} = S_{[i:v]} * T_{[i:v]} \) if \(i \in t1\)

- \(S^{t,u}_{[i:v]} = \{\} \) if \(v\) is not a descendant of \(u\)

- Other operations are also easy as long as they interact well with \(\cup\) and \(*\)
Comparison

  - Limited Expressiveness < Regular
- G. Bagan, “MSO Queries on Tree Decomposable Structures Are Computable with Linear Delay”, CSL 2006
  - “Enumeration” only
  - “Enumeration” only
  - His “AND-OR-DAG” is quite similar to SRED (say, “*-U-DAG”), but no clear set-theoretic meaning is assigned; hence it is not at all straightforward to derive other operations like selection