

Compact Representation for Answer Sets of n-ary Regular Tree Queries

(Accepted to CIAA 2009, July 14-17)



Kazuhiro Inaba (kinaba@nii.ac.jp)

joint work with Hauro Hosoya

IPL Seminar Jun 16, 2009

BACKGROUND:

History

- 2004-2005 [My Master Thesis]
 - “MTran: XML Transformation Language Based on Monadic Second-Order Logic”
 - Language Design
 - Linear Time Algorithm for MSO Query
 - ...it turned out that the time complexity **was already achieved** by: “Query evaluation via tree-decompositions” [Flum, Frick, and Grohe, JACM 2002]
- 2008 [Sudden Realization]
 - “Oh, my algorithm still has something new!”

BACKGROUND:

MTran

- XSLT/XQuery-like language
- MSO formulae as query expressions

```
<d1>
  {gather x ::  $x \in \langle \text{NAME} \rangle$  ::
    <dt>{x}</dt>
    {gather y ::  $y \in \langle \text{TEL} \rangle \ \& \ \exists z. (z/x \ \& \ z/y)$  ::
      <dd>{y}</dd>}
  }
</d1>
```

BACKGROUND:

MSO-Based Query

- Logic formula

$$\Phi(x_1, \dots, x_n)$$

with n free variables (each denoting a node of a tree) is an n -ary query

– Given a tree t

– Compute the set $q_\Phi(t) =$

$$\{ (v_1, \dots, v_n) \mid t, \{x_1:v_1, \dots, x_n:v_n\} \models \Phi \}$$

- **We need efficient algorithm to compute the set!**

BACKGROUND:

Known Results

- A query is definable by MSO iff definable by Bottom-up Det. Tree Automaton [Thatcher&Wright 1968]
 - The size $|\mathcal{A}|$ of BDTA is non-elementary wrt $|\Phi|$
- “Optimal” $O(|\mathcal{A}| \cdot (|t| + |q_\Phi(t)|))$ time algorithm for computing the set $|q_\Phi(t)|$ [Flum&Frick&Grohe 2002] ~~[Inaba 2004]~~
 - $|t|$: the size of the input
 - $|q_\Phi(t)|$: the size of the output (# of answers)

“Optimal”, but...

- $O(|\mathcal{A}| \cdot (|t| + |q_\phi(t)|))$
 - $\sim |t|^n$ for n-ary queries in the worst case
 - Big
- Do we really need to write down whole the contents of $q_\phi(t)$? → No!

Input Tree

size: IN
height: H

Run Query

$O(IN + OUT)$

Set of Output Tuples

size: $OUT \in O(IN^n)$

Today's Topic

Run Query
 $O(IN)$

Intermeditate Data Structure

size: $E \in O(\min(IN, OUT))$

Enum
 $O(OUT)$

isMember: $O(H)$

Get-Size: $O(E)$

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

"Selection": $O(H)$



MOTIVATION

Use-Cases of n-ary Queries (1)

(in MTran)

- Normal, tuple-selecting queries

```
{gather x,y,z ::  $\phi(x,y,z)$  ::  
  do something on (x,y,z)  
}
```

- For running this translation, there's no need to keep the whole set $q_\phi(t)$ on memory
- "One-by-one" enumeration is sufficient

Use-Cases of n-ary Queries (2)

(in MTran)

- “Relative” queries

```
{gather x ::  $\phi(x)$  ::  
  do something on x  
  {gather y ::  $\psi(x,y)$  ::  
    do something on y}}
```

- Simple Evaluation Strategy:

- Run the 1-ary query q_ϕ , and for each $v \in q_\phi(t)$,
 - Run the 1-ary query $q_{\psi(v,-)}$, regarding v as constant

→ slow

Use-Cases of n-ary Queries (2)

(in MTran)

- Example: replace every $\langle \text{foo} \rangle$ node with its descendant $\langle \text{bar} \rangle$ node
(assume that such y uniquely exists)

```
{visit x :: x:<foo> ::  
  {gather y :: x//y:<bar> :: y}}
```

- Simple Strategy takes $O(|t|^2)$ time
 - For each $\langle \text{foo} \rangle$ -node we do $O(|t|)$ query
- Binary Query Strategy [Berlea&Seidl 2004]
 - Run $q_{\psi(x,y)}$ as a 2-ary query
 - $O(|t|)$ time, run only once per translation

Use-Cases of n-ary Queries (2)

(in MTran)

- Problem of Binary Query Strategy
 - It works more efficiently than Simple Strategy, only when $|q_{\psi(x,y)}(t)| \sim |t|$

```
{visit x ::  $\phi(x) = x:\langle\text{foo}\rangle$  ::  
  {gather y ::  $\psi(x,y)=x//y:\langle\text{bar}\rangle$  :: y}}
```

- If $|q_{\psi(x,y)}(t)| \sim |t|^2$, then
 - Same asymptotic running time
 - Requires $O(|t|^2)$ working space
 - » Simple Strategy's $O(|t|)$

Our Approach

```
{gather x,y,z ::  $\phi(x,y,z)$  ::  
  do something on (x,y,z) }
```

- Represent $q_\phi(t)$ by a compact $O(|t|)$ -size structure that allows **one-by-one enum.**

```
{visit x ::  $\phi(x) = x:\langle\text{foo}\rangle$  ::  
  {gather y ::  $\psi(x,y)=x//y:\langle\text{bar}\rangle$  :: y}}
```

- Represent $q_\psi(t)$ by a compact $O(|t|)$ -size structure that allows **selection, i.e.,**
 $\text{sel}(S, v_x) = \{v_y \mid (v_x, v_y) \in S\}$, in $o(|t|)$ time

↑ **Small-o**



IMPLEMENTATION

(Bottom-up Deterministic) Tree Automaton

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

- $\mathcal{A} = (\Sigma_0, \Sigma_2, Q, F, \delta)$
 - Σ_0 : finite set of leaf labels
 - Σ_2 : finite set of internal-node labels
 - Q : finite set of states
 - δ : transition function
 $(\Sigma_0 \cup \Sigma_2 \times Q \times Q) \rightarrow Q$
 - $F \subseteq Q$: accepting states

Example (0-ary): ODDLEAVES

- $Q = \{q_0, q_1\}, F = \{q_1\}$

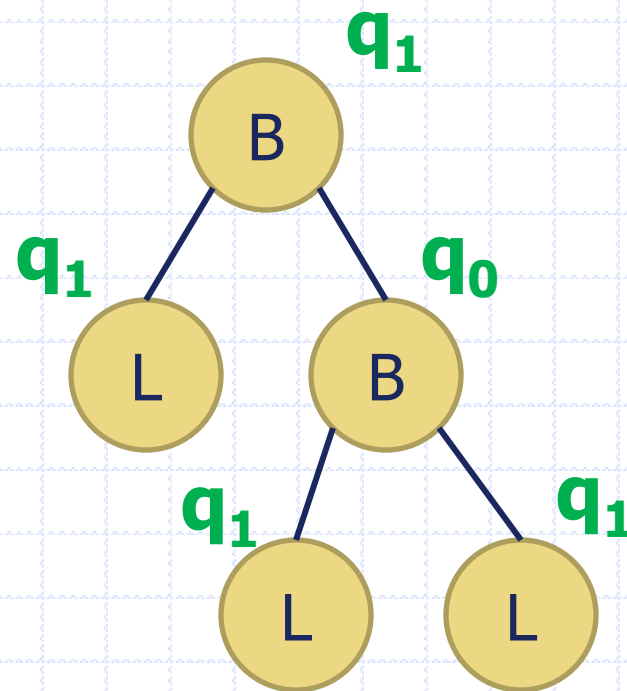
- $\delta(L) = q_1$

- $\delta(B, q_0, q_0) = q_0$

- $\delta(B, q_0, q_1) = q_1$

- $\delta(B, q_1, q_0) = q_1$

- $\delta(B, q_1, q_1) = q_0$



Tree Automaton for querying

- For any MSO formula $\Phi(x_1, \dots, x_n)$ on trees over $\Sigma_0 \cup \Sigma_2$,
- There exists a BDTA \mathcal{A}_Φ on trees over $\Sigma_0 \times B^n, \Sigma_2 \times B^n$ (where $B = \{0, 1\}$) s.t.
 - $t \models \Phi(v_1, \dots, v_n)$
 - iff
 - \mathcal{A}_Φ accepts the tree $\text{mark}(t, v_1, \dots, v_n)$
 - $\text{mark}(t, \dots) = t$ with the i -th B component is 1 at v_i and 0 at other nodes

Example (1-ary): LEFTMOST

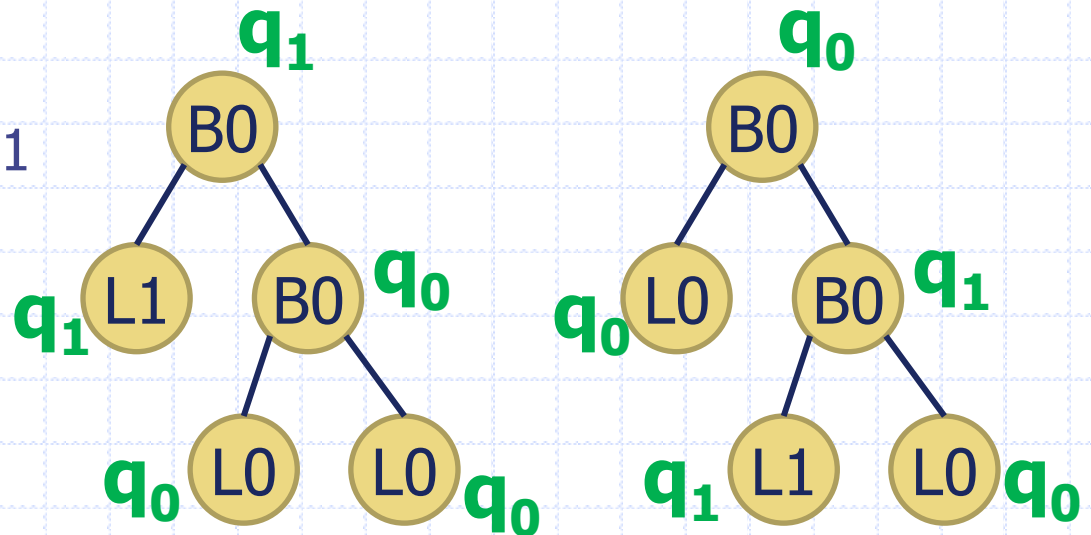
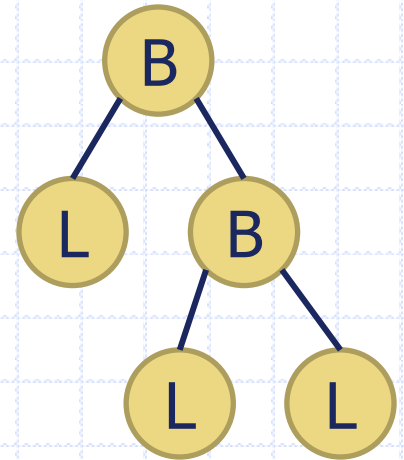
- $Q = \{q_0, q_1\}, F = \{q_1\}$

- $\delta(L0) = q_0$

- $\delta(L1) = q_1$

- $\delta(B0, q_1, q_0) = q_1$

- $\delta(\text{otherwise}) = q_0$



NA: Naïve n-ary query algorithm

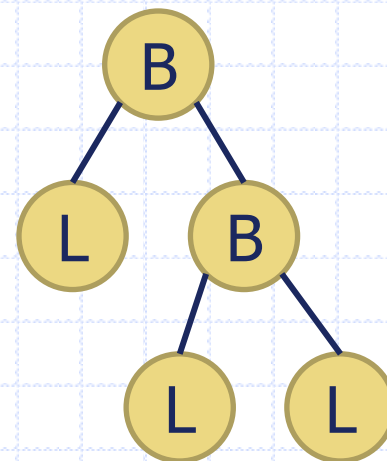
- For each tuple $(v_1, \dots, v_n) \in \text{Node}(t)^n$
 - Generate $\text{mark}(t, v_1, \dots, v_n)$
 - Run \mathcal{A}_Φ on it
 - If accepted, then (v_1, \dots, v_n) is one of the answer
- Run \mathcal{A}_Φ on t $O(|t|^n)$ times = $O(|t|^{n+1})$

OA: One-pass Algorithm

- For each combination of node v , state q , and $b_1, \dots, b_n \in B$
 - Compute the set
$$r_v(q, b_1, \dots, b_n) \subseteq (\text{Node}(t) \cup \{\perp\})^n \text{ s.t.}$$
 - $(v_1, \dots, v_n) \in r_v(q, b_1, \dots, b_n)$
iff
 $(\forall i : \text{"descendant } v_i \text{ of } v \text{ is marked and } b_i=1" \text{ or } \text{"}v_i=\perp \text{ and } b_i=0\text{"}) \Rightarrow \text{"automaton assigns } q \text{ at node } v\text{"}$

Example (2-ary): LEFT&RIGHT

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



- $\delta(L00) = \delta(B10, q_0, q_0) = q_0$

- $\delta(L10) = \delta(B10, q_0, q_0) = q_1$

- $\delta(L01) = \delta(B01, q_0, q_0) = q_2$

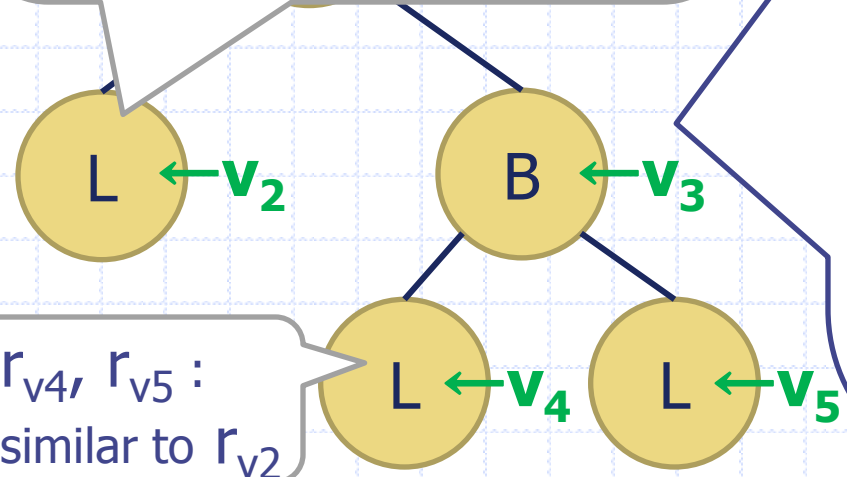
- $\delta(B00, q_1, q_2) = q_3$

- $\delta(B00, q_0, q_i) = \delta(B00, q_i, q_0) = q_i$ (for $i=1,2$)

- $\delta(\text{otherwise}) = q_4$

$$\begin{aligned} \delta(L00) &= \delta(B10, q_0, q_0) = q_0 \\ \delta(L10) &= \delta(B10, q_0, q_0) = q_1 \\ \delta(L01) &= \delta(B01, q_0, q_0) = q_2 \\ \delta(B00, q_1, q_2) &= q_3 \\ \delta(B00, q_0, q_2) &= q_2 \quad \dots \end{aligned}$$

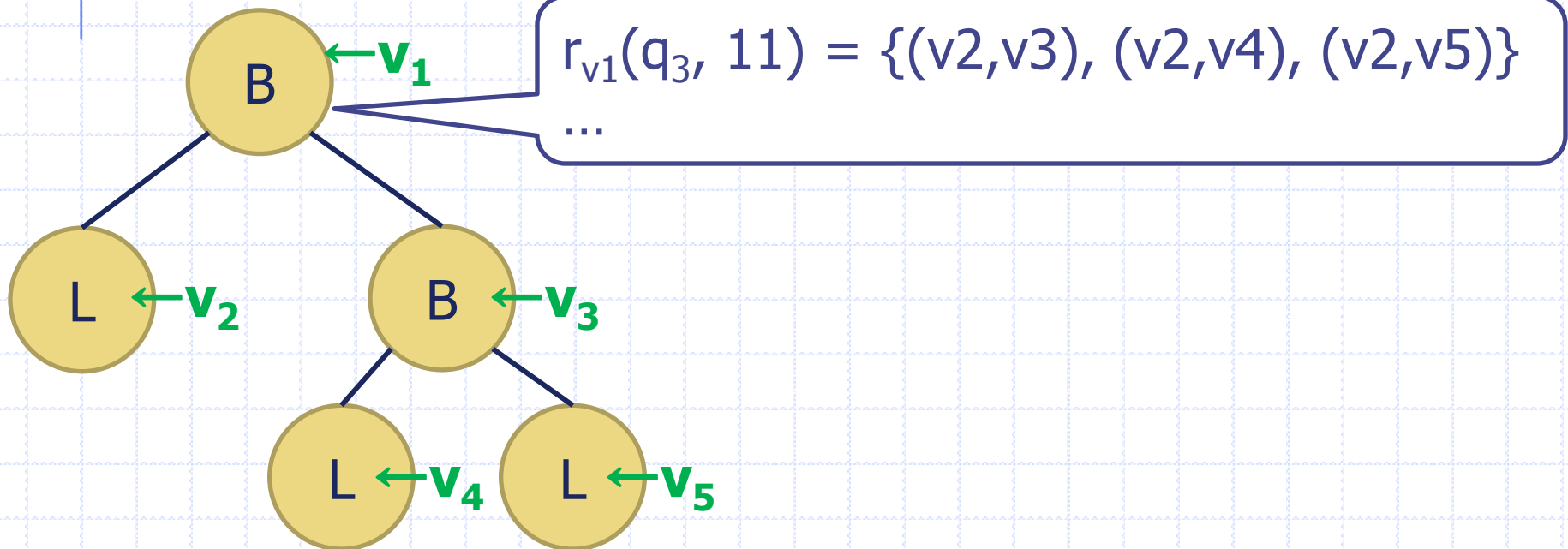
$$\begin{aligned} r_{v_2}(q_0, 00) &= \{ (\perp, \perp) \} \\ r_{v_2}(q_1, 10) &= \{ (v_2, \perp) \} \\ r_{v_2}(q_2, 01) &= \{ (\perp, v_2) \} \\ r_{v_2}(q_4, 11) &= \{ (v_2, v_2) \} \\ r_{v_2}(_, _) &= \{ \} \end{aligned}$$



$$\begin{aligned} r_{v_3}(q_0, 00) &= r_{v_4}(q_0, 00) * \{ (\perp, \perp) \} * r_{v_5}(q_0, 00) \\ &= \{ (\perp, \perp) \} \\ \hline r_{v_3}(q_3, 11) &= r_{v_4}(q_1, 00) * \{ (\perp, \perp) \} * r_{v_5}(q_2, 11) \\ &\cup r_{v_4}(q_1, 00) * \{ (v_3, \perp) \} * r_{v_5}(q_2, 01) \\ &\dots \\ &\cup r_{v_4}(q_1, 10) * \{ (\perp, \perp) \} * r_{v_5}(q_2, 01) \\ &\dots \quad (= \{ (v_4, \perp) \} * \{ (\perp, \perp) \} * \{ (\perp, v_5) \}) \\ &= \{ (v_4, v_5) \} \\ \hline r_{v_3}(q_2, 01) &= r_{v_4}(q_0, 00) * \{ (\perp, v_3) \} * r_{v_5}(q_0, 00) \\ &\cup r_{v_4}(q_0, 00) * \{ (\perp, \perp) \} * r_{v_5}(q_2, 01) \\ &\cup r_{v_4}(q_2, 01) * \{ (\perp, \perp) \} * r_{v_5}(q_2, 00) \\ &\dots \\ &= \{ (\perp, v_3), (\perp, v_4), (\perp, v_5) \} \end{aligned}$$

Example (2-ary): LEFT&RIGHT

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



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 $*$ operations
 - Each operand set of \cup and $*$ may be of size $O(|t|^n)$
 - \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,...)

RELATED WORK:

[Flum, Frick, Grohe 2002]

- $O(|t| + |a|)$
 - Where $a = r_{\text{root}}(q_f, 11\dots 11)$
- By two-pass preprocessing, skip the computations of unnecessary $r_v(q)$'s
 - (state q has no possibility to reach the final state at the root node)
 - (eventually taken product between $\{\}$)

Important Points of OA

- One-pass: $|t|$
- At each node, constant (wrt $|t|$) number $|Q|^2 \cdot 3^n$ of set-operations
- Only 2 (3?) kind of operations are used
 - \cup : union, in fact, “disjoint” union
 - $*$: product
 - (singleton-set construc

If we have $O(1)$ impl. for all of these, we get $O(|t|)$ algorithm!

OUR MAIN IDEA: Symbolic Repr. of Operations

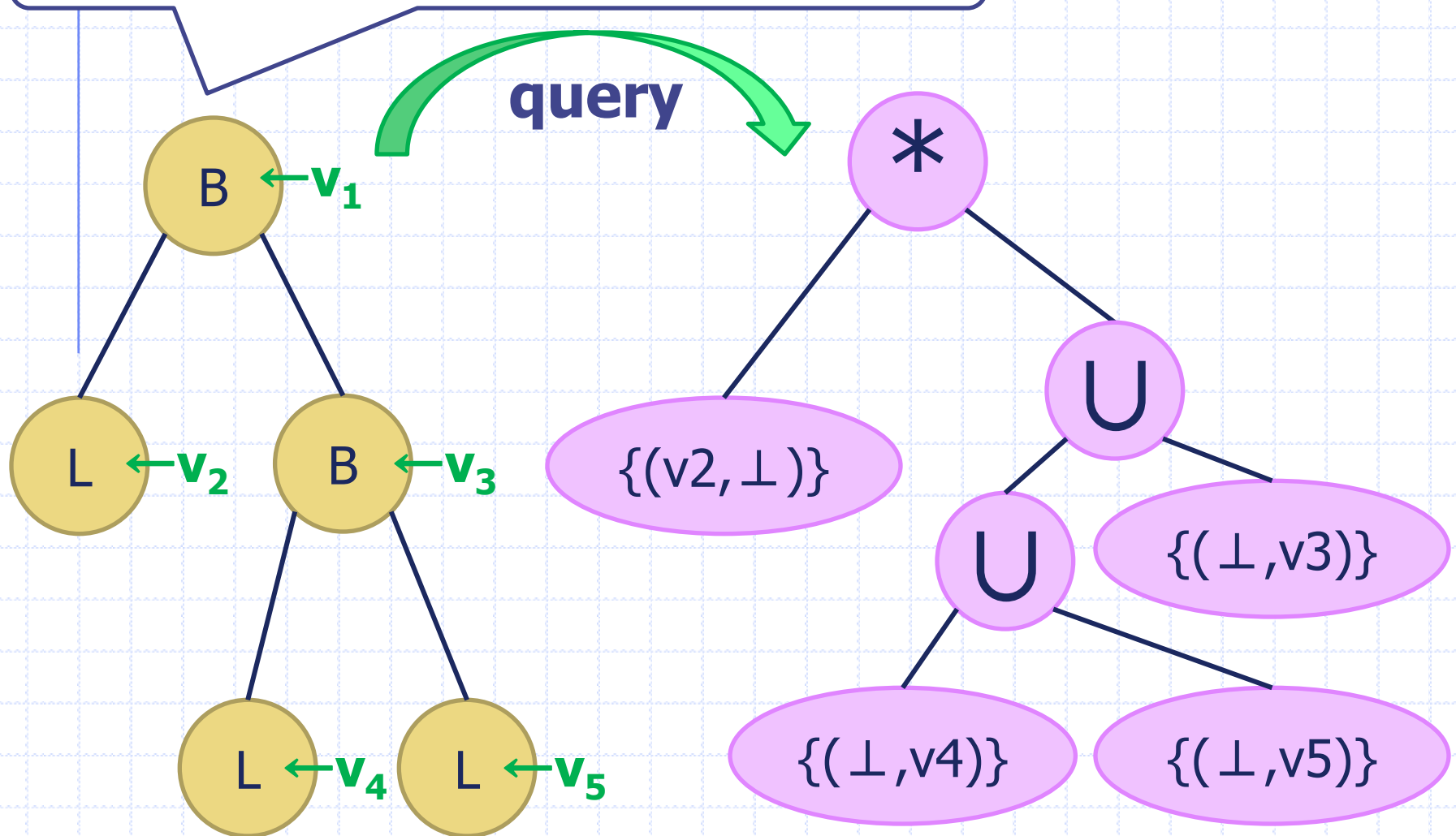
- data Set e =
 - Empty -- \emptyset
 - Unit -- $\{(\perp, \dots, \perp)\}$
 - Nonempty (Set1 e)
- data Set1 e =
 - Singleton e
 - DisjointUnion (Set1 e) (Set1 e)
 - Product (Set1 e) (Set1 e)

Properties of SR

- Constant time \cup and $*$ implementation
 - Now we have $O(|t|)$ querying algorithm 😊
- Small
 - $O(|t|)$ time implies $O(|t|)$ space
 - By eliminating $\{\}$, we have $O(\text{OUT})$, too
- Easy to manipulate **!IMPORTANT!**
 - Conversion to the representing set
 - Several other set operations

Example (2-ary): LEFT&RIGHT

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



Set-Operations on SR

- Evaluation (“Decompression”)
 - **Simple!** (assumption: \cup is $O(1)$)
 - $\text{eval}(\text{Empty}) = \{\}$
 - $\text{eval}(\text{Unit}) = \{(\perp, \dots, \perp)\}$
 - $\text{eval}(\text{Nonempty } s) = \text{eval } s$
 - $\text{eval}(\text{Singleton } e) = \{e\}$
 - $\text{eval}(\text{DisjointUnion } s1 \ s2) = \text{eval } s1 \cup \text{eval } s2$
 - $\text{eval}(\text{Product } s1 \ s2) = \text{eval } s1 * \text{eval } s2$

Set-Operations on SR

- Many operations are easily implemented by **straightforward induction** on U -*-tree structure
 - Projection
 - $\text{proj}_i(S) = \{v_i \mid (v_1, \dots, v_i, \dots, v_n) \in S\}$
 - Selection
 - $\text{sel}_i(S, v) = \{(v_1, \dots, v_{i-1}, v_{i+1}, \dots, v_n) \mid (v_1, \dots, v_{i-1}, v, v_{i+1}, \dots, v_n) \in S\}$
 - isMember
 - ...

In fact, ...

- For efficiency, we add some annotation

- data Set1 e =

- Singleton e Node Int
- DisjointUnion (Set1 e) (Set1 e) Node Int
- Product (Set1 e) (Set1 e) Node Int

The **input node** where the operation was performed

The **"type"** of the set.
 $\{(v,v, \perp, \perp)\} :: 0b1100$

Related Work

- H. Meuss, K. U. Schulz, and F. Bry, “Towards Aggregated Answers for Semistructured Data”, ICDT 2001
 - Limited Expressiveness < Regular
- G. Bagan, “MSO Queries on Tree Decomposable Structures Are Computable with Linear Delay”, CSL 2006
- B. Courcelle, “Linear Delay Enumeration and Monadic Second-Order Logic”, to appear in Discrete Applied Mathematics, 2009
 - “Enumeration” only

Thank You for Listening!

