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?”

N-ary Regular Queries over Trees

- Query definable by a tree automaton
- Regular
  - iff definable by Monadic 2nd-Order Logic
  - iff definable by Modal μ-Calculus
  - iff definable by Monadic Datalog
  - iff definable by Boolean Attribute Grammar
  - …

Efficiency of Regular Queries

- Given a query q (represnted by an automaton $\mathcal{A}$), and an input tree t, we can compute q(t) in…

- $\mathbf{O(|\mathcal{A}| \cdot (|t| + |q(t)|))}$ time

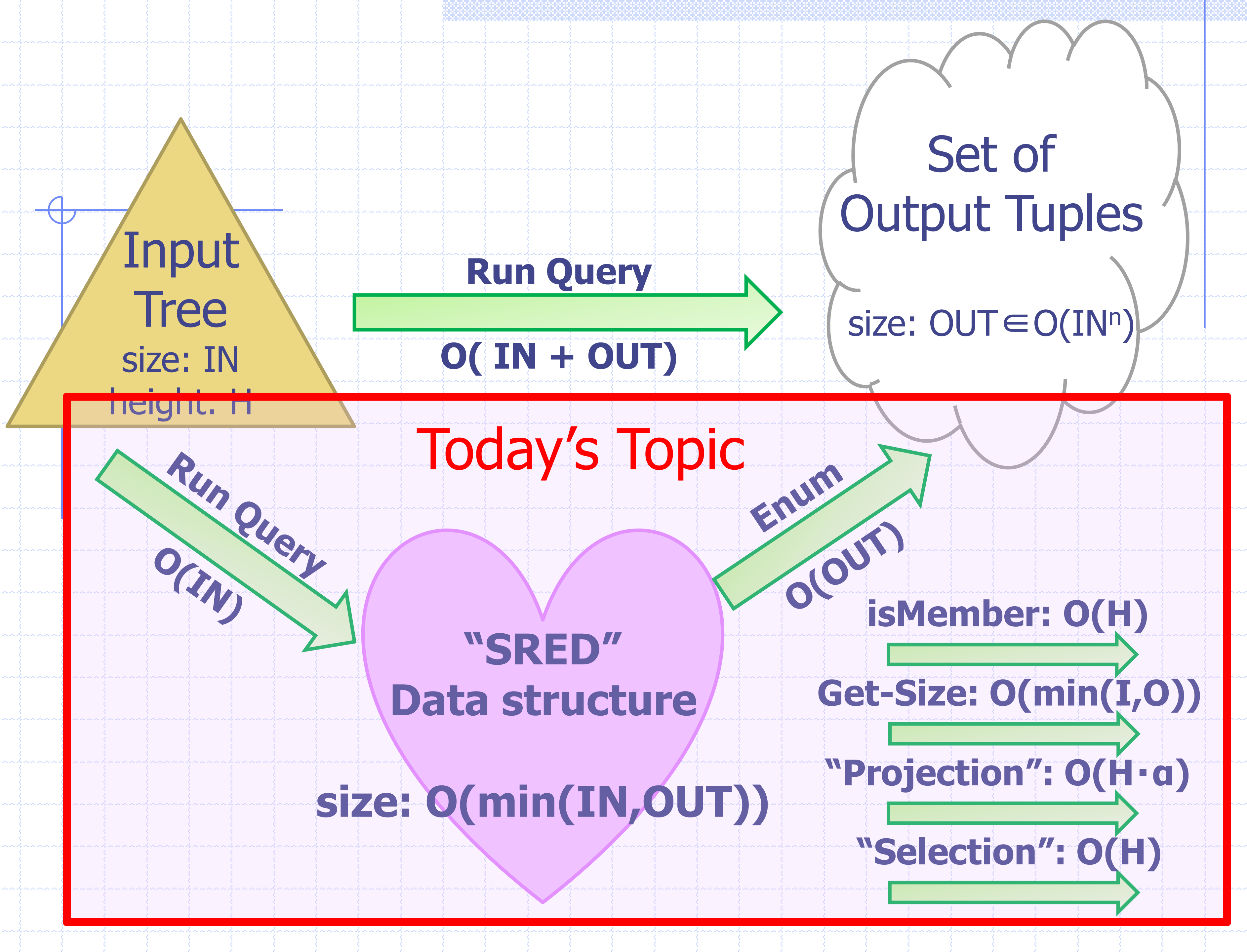
  [Flum & Frick & Grohe 2002]

  - (In some sense) optimal

“Optimal”, but…

Still Big

- $O(\,|\mathcal{A}|\cdot(\,|t| + |q(t)|\,)\,)$
  - $\sim|t|^n$ 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∈O(IN^n)
Today's Topic
Run Query
O(IN)
Enum
O(OUT)
"SRED"
Data structure
size: O(min(IN,OUT))
isMember: O(H)
Get-Size: O(min(I,O))
"Projection": O(H·α)
"Selection": O(H)

Outline

① Introduction
- N-ary Regular Queries & Their Complexity

② Application
- When Do We Need Compact Representation?

③ Algorithm
- Query Algorithm (Suitable for “SRED” Repr.)
- “SRED” Data Structure

④ Conclusion

APPLICATION
(IN XML TRANSLATION)

“Relative” Queries in XML

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

- 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>”

Two Evaluation Stategies

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

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

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

A := the answer set of
  1-ary query {x | Φ(x)}
C := the answer set of
  2-ary query {(x,y) | Φ(x)&Ψ(x,y)}
for each x in A:
  B := **{y | (x,y)∈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 | Φ(x)}

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

A := the answer set of
  1-ary query {x | Φ(x)}
C := the answer set of
  2-ary query {(x,y) |
    Φ(x)&Ψ(x,y)}
for each x in A:
  B := **{y | (x,y)∈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

$O(|t|^2)$ time
$O(|t|)$ space
in "worst" cases
(= many x, many y)

$O(|t|^2)$ time
$O(|t|^2)$ space
in "worst" cases

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

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

A := the answer set of
  1-ary query {x | Φ(x)}
C := the answer set of
  2-ary query {(x,y) | Φ(x)&Ψ(x,y)}
for each x in A:
  B := **{y | (x,y)∈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

$O(|t|^2)$ time
$O(|t|)$ space
in “worst” cases
(= many x, many y)

$O(|t|^2)$ time
$O(|t|^2)$ space
in “worst” cases

A := the answer set of

If We Use “SRED”
to Represent the Set C …!!
$O(|t|)$ time
in “common” cases

$O(|t|^2)$ time $O(|t|)$ space
in “worst” cases

A := the answer set of
1-ary query {x | Φ(x)}
C := the answer set of
2-ary query {(x,y) | Φ(x)&Ψ(x,y)}
for each x in A:
B := **{y | (x,y)∈C}** $= C_{[1:x]}$
for each y in B:
print <item>y</item>;

IMPLEMENTATION OF REGULAR QUERIES USING "SRED"

(Bottom-up Deterministic) Tree Automaton

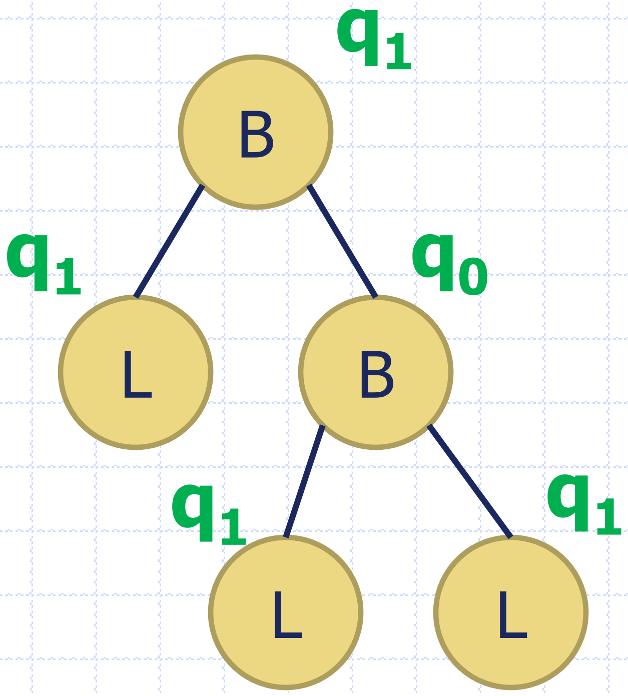
(For simplicity, we limit our attention to binary trees)

- $\mathcal{A} = (\Sigma_0, \Sigma_2, Q, F, \delta)$
  - 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

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

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

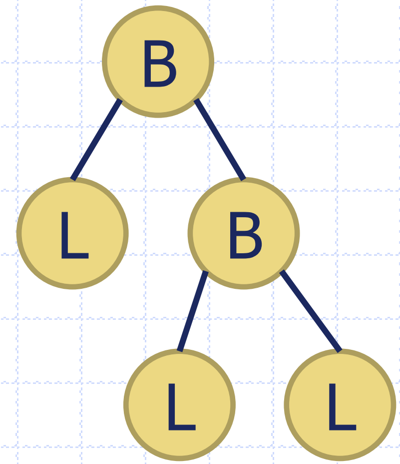


Tree Automaton for Querying

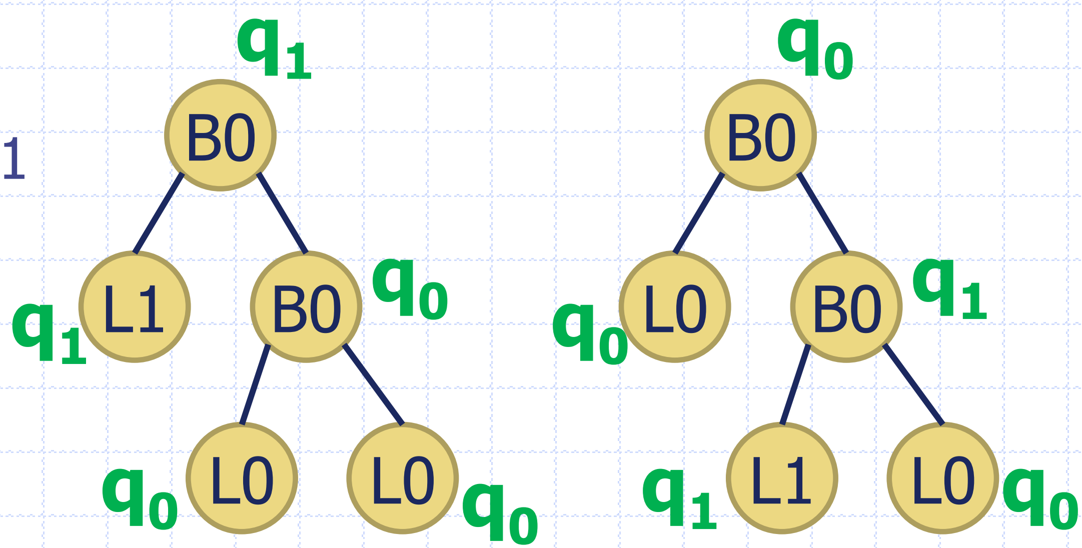
- For any n-ary regular query Φ on trees over $\Sigma_0 \cup \Sigma_2$,
- There exists a BDTA $\mathcal{A}_\Phi$ on trees over $\Sigma_0 \times B^n$, $\Sigma_2 \times B^n$ (where B={0,1}) s.t.
  - $(v_1,\ldots,v_n) \in \Phi(t)$
  - iff
  - $\mathcal{A}_\Phi$ accepts the tree mark$(t,v_1,\ldots,v_n)$
    - mark(t,…) = t with the i-th B component is 1 at vi 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

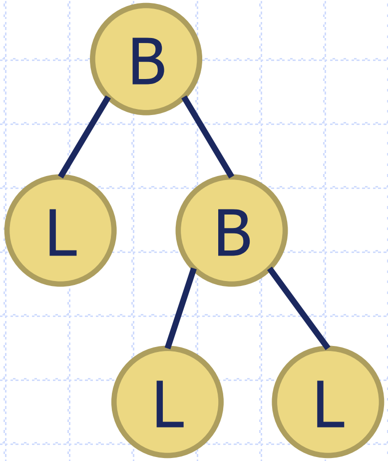
- For each tuple $(v_1,..,v_n) \in Node(t)^n$
  - Generate mark$(t, v_1, \ldots, v_n)$
  - Run $\mathcal{A}_\Phi$ on it
    - If accepted, then $(v_1,\ldots,v_n)$ is one of the answer

- Run $\mathcal{A}_\Phi$ 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 $r_v(q, b_1, \ldots, b_n) \subseteq (Node(t) \cup \{\perp\})^n$ s.t.
  - $(v_1, \ldots, v_n) \in r_v(q, b_1, \ldots, b_n)$ iff ($\forall i$: "descendant v_i of v is marked and $b_i=1$" or "$v_i=\perp$ and $b_i=0$") $\Rightarrow$ "automaton assigns q 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)
- $\delta(\text{otherwise}) = q_4$

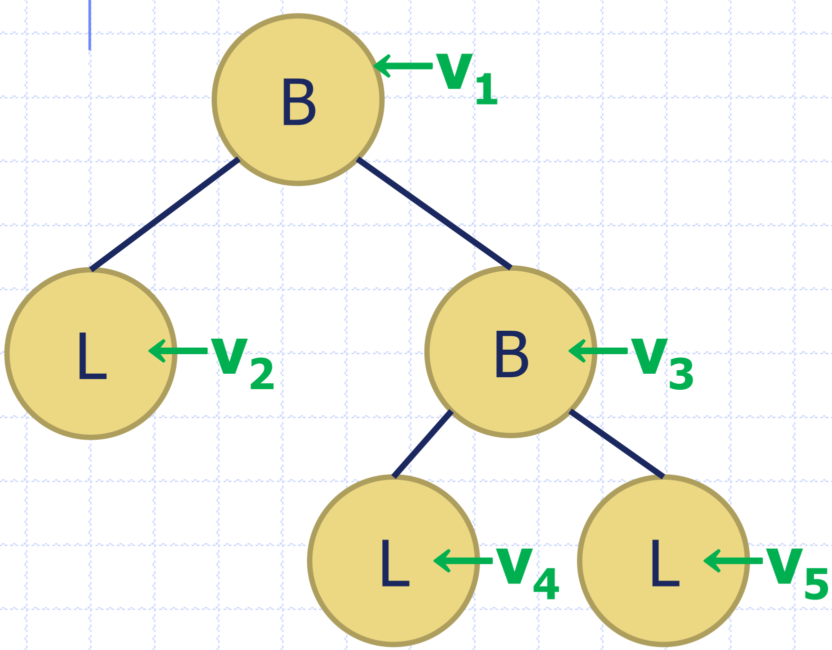
$\delta(L00) = \delta(B10, q_0, q_0) = q_0$

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

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

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

$\delta(B00, q_0, q_2) = q_2$ …



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

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

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

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

$\delta(B00, q_0, q_2) = q_2$ …

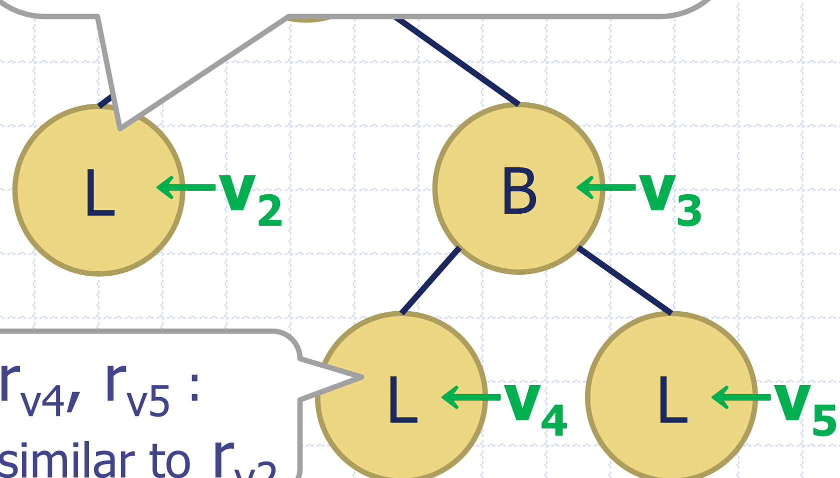
$r_{v2}(q_0, 00) = \{ (\perp, \perp) \}$

$r_{v2}(q_1, 10) = \{ (v2, \perp) \}$

$r_{v2}(q_2, 01) = \{ (\perp, v2) \}$

$r_{v2}(q_4, 11) = \{ (v2, v2) \}$

$r_{v2}(\_, \ \_) = \{\}$



r_{v4}, r_{v5} : similar to r_{v2}

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

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

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

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

$\delta(B00, q_0, q_2) = q_2$ …

$r_{v2}(q_0, 00) = \{ (\perp,\perp) \}$

$r_{v2}(q_1, 10) = \{ (v2,\perp) \}$

$r_{v2}(q_2, 01) = \{ (\perp,v2) \}$

$r_{v2}(q_4, 11) = \{ (v2,v2) \}$

$r_{v2}(\_,\ \ \_) = \{\}$

$r_{v3}(q_0, 00)$

$= r_{v4}(q_0,00) * \{(\perp,\perp)\} * r_{v5}(q_0,00)$

$= \{(\perp,\perp)\}$

$r_{v3}(q_3, 11)$

$= r_{v4}(q_1,00) * \{(\perp,\perp)\} * r_{v5}(q_2,11)$

$\cup\ r_{v4}(q_1,01) * \{(\perp,\perp)\} * r_{v5}(q_2,10)$

$\cup\ r_{v4}(q_1,10) * \{(\perp,\perp)\} * r_{v5}(q_2,01)$

$(= \{(v4,\perp)\} * \{(\perp,\perp)\} * \{(\perp,v5)\}$

$\cup\ r_{v4}(q_1,11) * \{(\perp,\perp)\} * r_{v5}(q_2,01)$

$= \{(v4,v5)\}$

$r_{v3}(q_2, 01)$

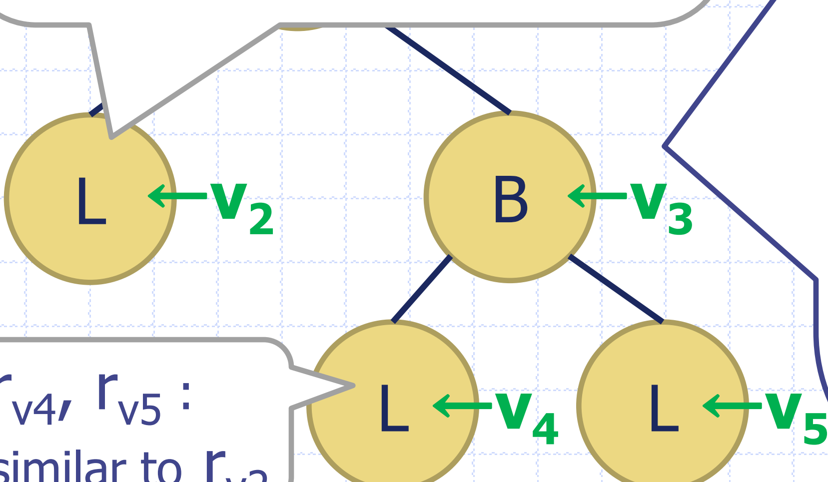
$= r_{v4}(q_0,00) * \{(\perp,v3)\} * r_{v5}(q_0,00)$

$\cup\ r_{v4}(q_0,00) * \{(\perp,\perp)\} * r_{v5}(q_2,01)$

$\cup\ r_{v4}(q_2,01) * \{(\perp,\perp)\} * r_{v5}(q_2,00)$

$= \{(\perp,v3), (\perp,v4), (\perp,v5)\}$

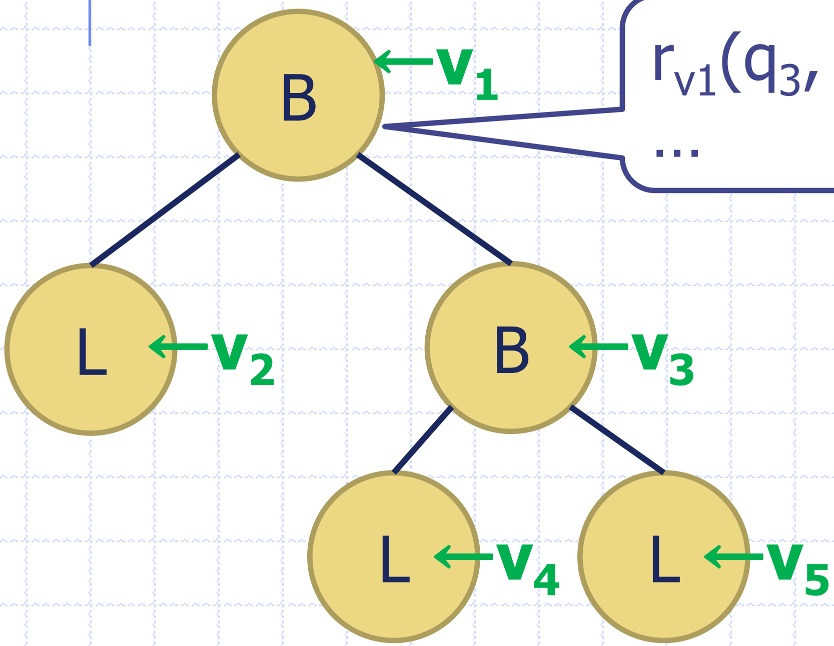
…



r_{v4}, r_{v5} : similar to r_{v2}

Example (2-ary): LEFT&RIGHT

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



$r_{v1}(q_3, 11) = \{(v2,v3), (v2,v4), (v2,v5)\}$

…

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 as large as $O(|t|^n)$
    - → 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,

  Constant wrt |t|!

  - $|Q| \times 2^n$ entries of r are filled
  - Need $O(|Q|^2 \cdot 3^n)$ ∪ and * operations
  - Each operand set of ∪ and * may be as large as $O(|t|^n)$
  - ...ion takes $O(|t|^n)$ time in the ...ong as the "set"s are ... usual data structure (lists, rb-

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|!

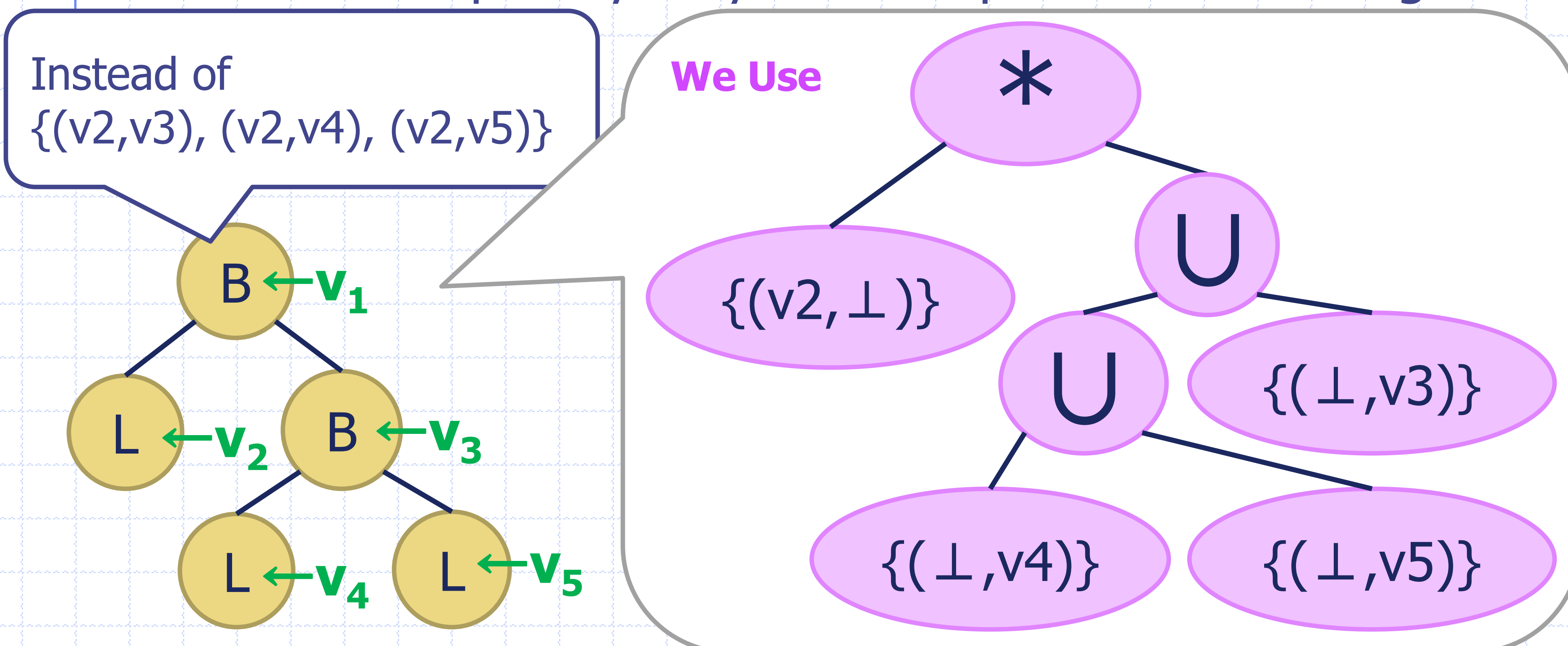
  - $|Q| \times 2^n$ entries of r are filled
  - Need $O(|Q|^2 \cdot 3^n)$ $\cup$ and * operations
  - Each operand set of $\cup$ and * may be as large as $O(|t|^n)$
  - …ion takes $O(|t|^n)$ time in the … …ong as the … usual data…

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

O(|t|) Time Quering!

!! Our Main Idea !!

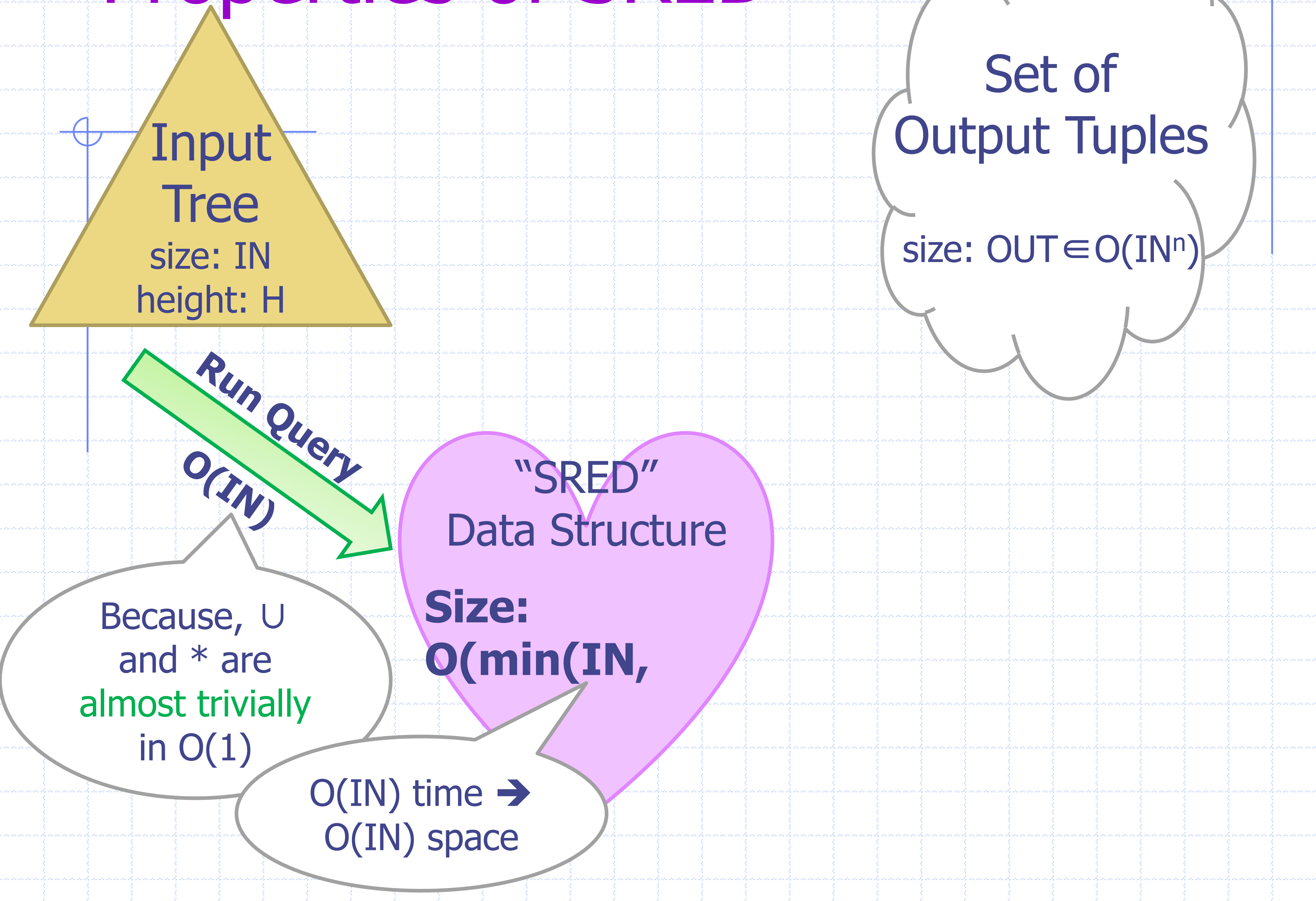
- SRED:
  Set Representation by Expression Dags
  - Set is Repr'd by a Symbolic Expression Producing it



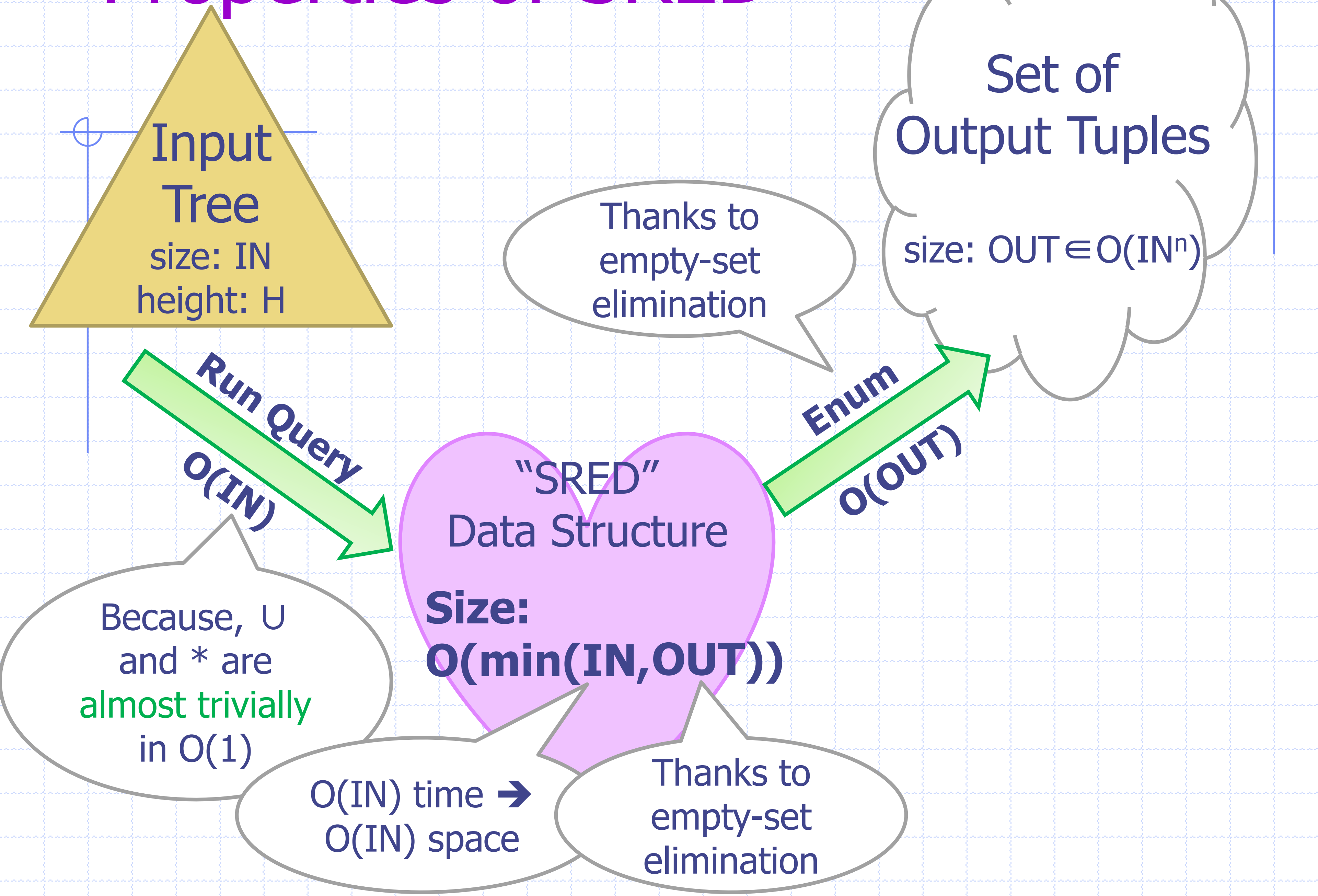
BNF for SRED (Simplified)

- SET ::=
  - Empty -- {}
  - Unit -- {(⊥,…,⊥)}
  - NESET
- NESET ::=
  - Singleton(ELEMENT)
  - DisjointUnion(NESET, NESET)
  - Product(NESET, NESET)

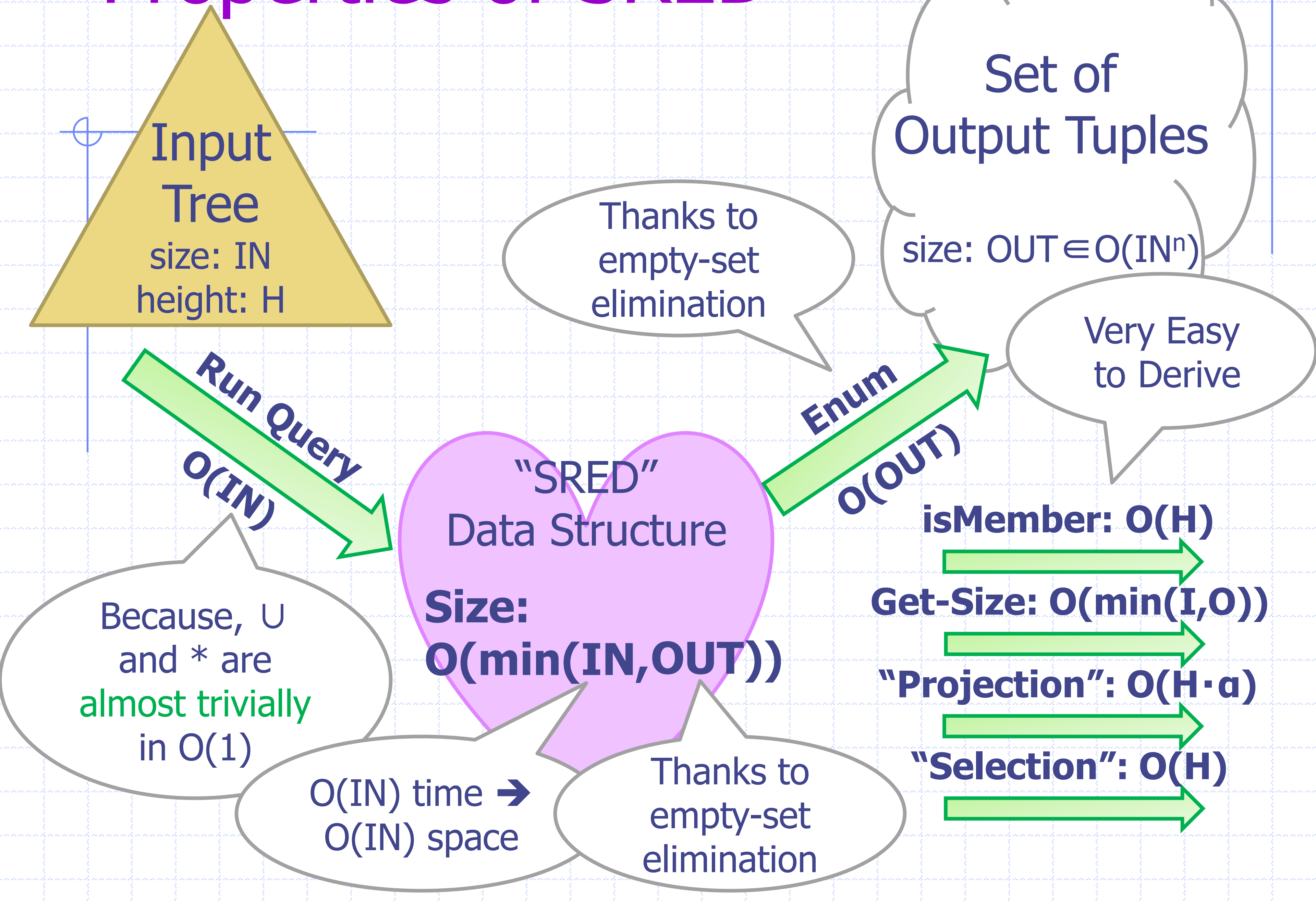
Properties of SRED



Properties of SRED



Properties of SRED

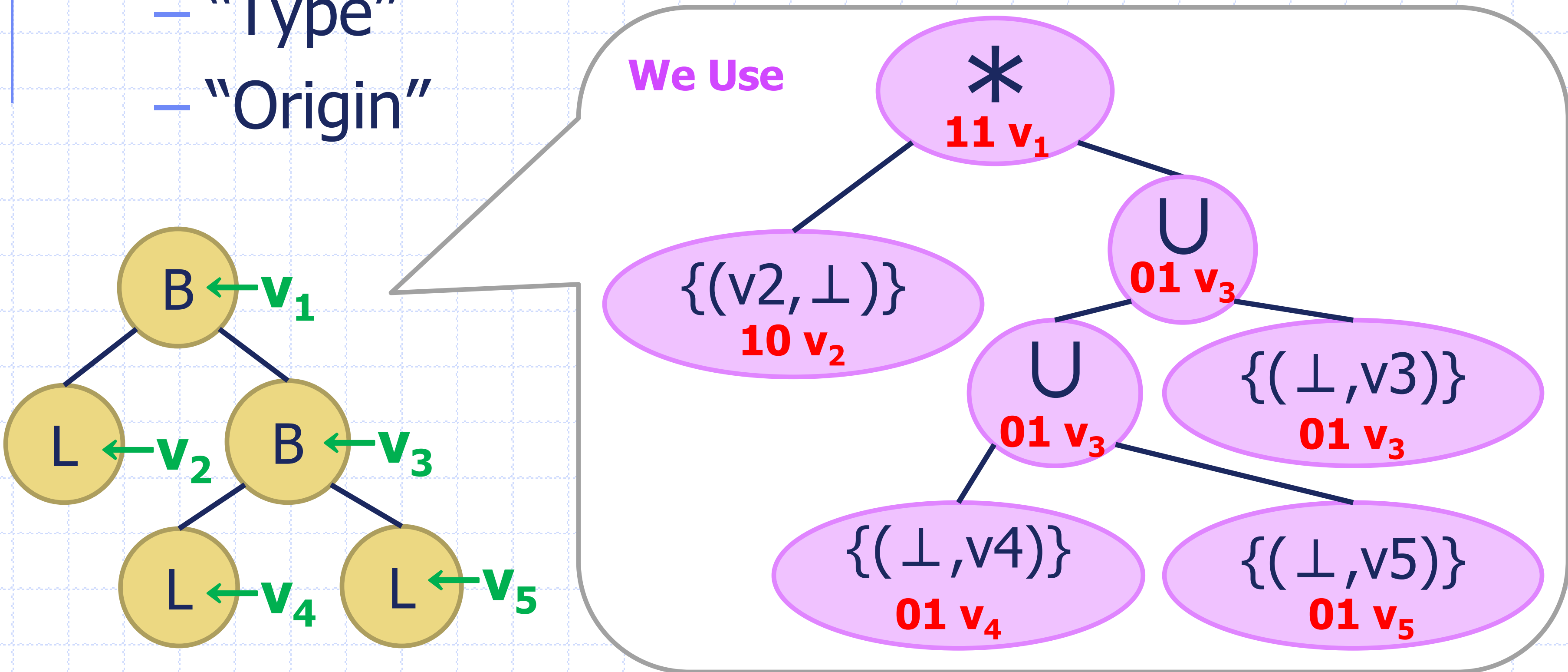


O(OUT) Enumeration of SRED
(or, "decompression")

- Simple Recursion is Enough!
  (assumption: $\cup$ is O(1), * is O(out))
  - eval(Empty) = {}
  - eval(Unit) = $\{(\perp, \ldots, \perp)\}$
  - 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

- $S_{[i:v]} \stackrel{def}{=} \{(v_1,\ldots,v_{i-1},v_{i+1},\ldots,v_n) \mid (v_1,\ldots,v_{i-1}, v, v_{i+1},\ldots,v_n) \in S\}$
  - Again, Simple Recursion!
  - $(S \cup T)_{[i:v]} = S_{[i:v]} \cup T_{[i:v]}$
  - $(S^{t1,u1} * T^{t2,v2})_{[i:v]} = S_{[i:v]} * T$ if $i \in t1$
    $= S * T_{[i:v]}$ if $i \in t2$
  - $S^{t,u}[i:v] = \{\}$ if v is not a descendant of u
- Other operations are also easy as long as they interact well with $\cup$ and $*$

Comparison

- 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
  - “Enumeration” only
- B. Courcelle, “Linear Delay Enumeration and Monadic Second-Order Logic”, to appear in Discrete Applied Mathematics, 2009
  - “Enumeration” only
  - His “AND-OR-DAG” is quite similar to SRED (say, “*-∪-DAG”), but no clear set-theoretic meaning is assigned; hence it is not at all straightforward to derive other operations like selection