Advanced SQL

05 — Recursion

Torsten Grust
Universität Tübingen, Germany

Computational Limits of SQL

SQL has grown to be an **expressive data-oriented language.** Intentionally, it has *not* been designed as a general-purpose programming language:

1. *SQL does not loop forever:*
   Any SQL query is expected to **terminate,** regardless of the size/contents of the input tables.

2. *SQL can be **evaluated efficiently:***
   A SQL query over table T of c columns and r rows can be evaluated in $O(r^c)$ space and time.[1]

[1] SQL cannot compute the set of all subsets of rows in T which requires $O(2^r)$ space, for example.

A Giant Step for SQL

The addition of **recursion** to SQL changes everything:

Expressiveness SQL becomes a **Turing-complete language** and thus a general-purpose PL (albeit with a particular flavor).

Efficiency ⚠️ **No longer** are queries guaranteed to **terminate** or to be **evaluated with polynomial effort.**

Like a pact with the 😠 — but the payoff is magnificient…

Recursion in SQL: WITH RECURSIVE

Recursive common table expresssion (CTE):

```
WITH RECURSIVE
  <T1>(<c11>,…,<c1,k1>) AS (
    <q1> ),
   ⋮
  <Tn>(<cn1>,…,<cn,kn>) AS (
    <qn> )
<q>
```

Queries <qj> may refer ***to all*** <Ti>

<q> may refer ***to all*** <Ti>

- In particular, any <qj> may refer to itself (↻)! Mutual references are OK, too. (Think letrec in FP.)
- Typically, final query <q> performs post-processing only.

Shape of a Self-Referential Query

```
WITH RECURSIVE
  T(c1,…,ck) AS (    -- common schema of q0 and qϑ(•)
     q0              -- base case query, evaluated once

    UNION [ ALL ]    -- either UNION or UNION ALL

     qϑ(T)           -- recursive query refers to T itself,
  )                  --   evaluated repeatedly
q(T)                 -- final post-processing query
```

- Semantics in a nutshell:

$$q(\underbrace{q\vartheta(\cdots q\vartheta(q\vartheta(q_0))\cdots) \cup \cdots \cup q\vartheta(q\vartheta(q_0)) \cup q\vartheta(q_0)}_{\text{repeated evaluation of } q\vartheta \text{ (when to stop?)}} \cup q_0)$$

Semantics of a Self-Referential Query (UNION Variant)

Iterative and recursive semantics—both are equivalent:

```
iterate(q↻, q₀):
  r ← q₀
  t ← r
  while t ≠ ∅
  | t ← q↻(t) \ r
  | r ← r ⊎ t
  return r
```

```
recurse(q↻, r):
  if r ≠ ∅ then
  | return r ⊎ recurse(q↻, q↻(r) \ r)
  else
  | return ∅
```

- Invoke the recursive variant with recurse($q↻$, q_0).
- ⊎ denotes disjoint set union, \ denotes set difference.
- $q↻(\cdot)$ evaluated over **the *new* rows found in the last iteration/recursive call.** Exit if there were no new rows.

A Home-Made `generate_series()`

Generate a single-column table of integers $i \in \{\langle from \rangle, \langle from \rangle+1, \ldots, \langle to \rangle\}$:

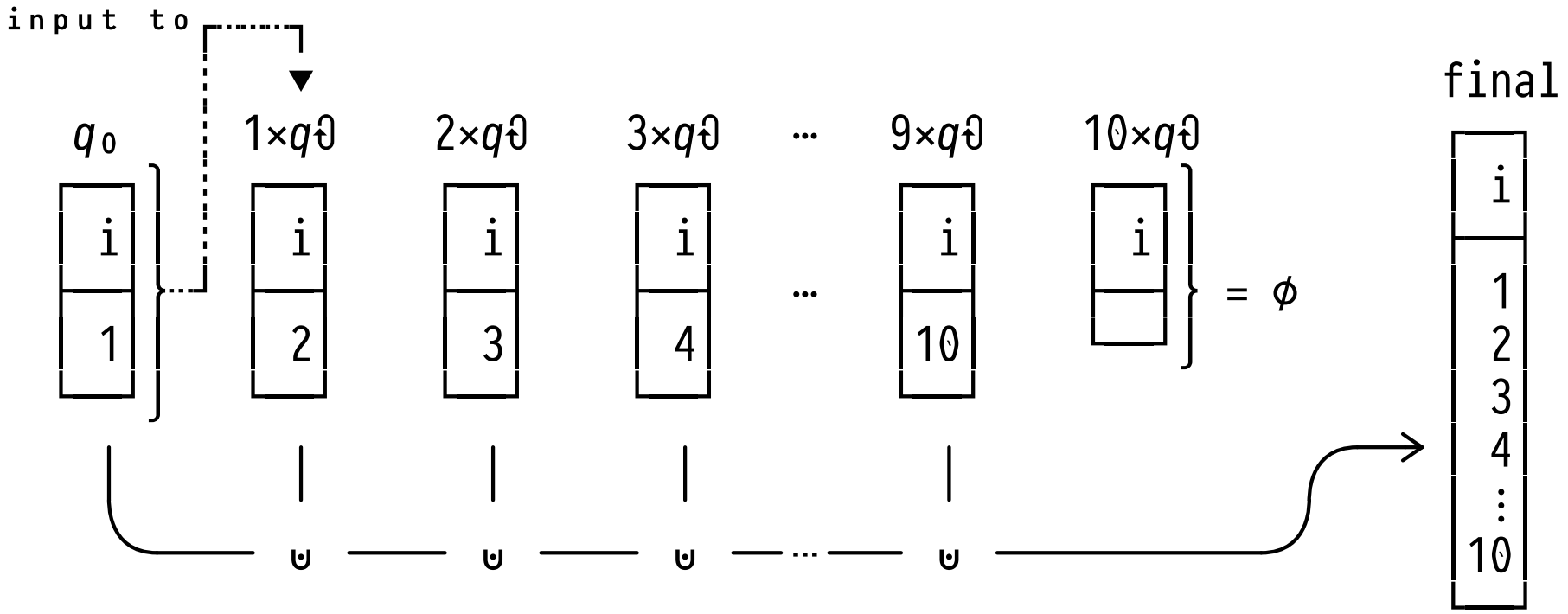
```
WITH RECURSIVE
  series(i) AS (
  ▲ VALUES (⟨from⟩)              -- q0
  ┆    UNION
  ┆ SELECT s.i + 1 AS i         --
  └FROM····▶series AS s         --  } q↺(series)      ▲ self-
    WHERE  s.i < ⟨to⟩           --                    ┆ reference
)                                                     └··▶
TABLE series
```

- **Q:** Given the predicate `s.i < ⟨to⟩`, will ⟨*to*⟩ indeed be in the final table?

A Home-Made `generate_series()`

- Assume ‹*from*› = 1, ‹*to*› = 10:

New rows in table **series** after evaluation of…



Semantics of a Self-Referential Query (UNION ALL Variant) ✎

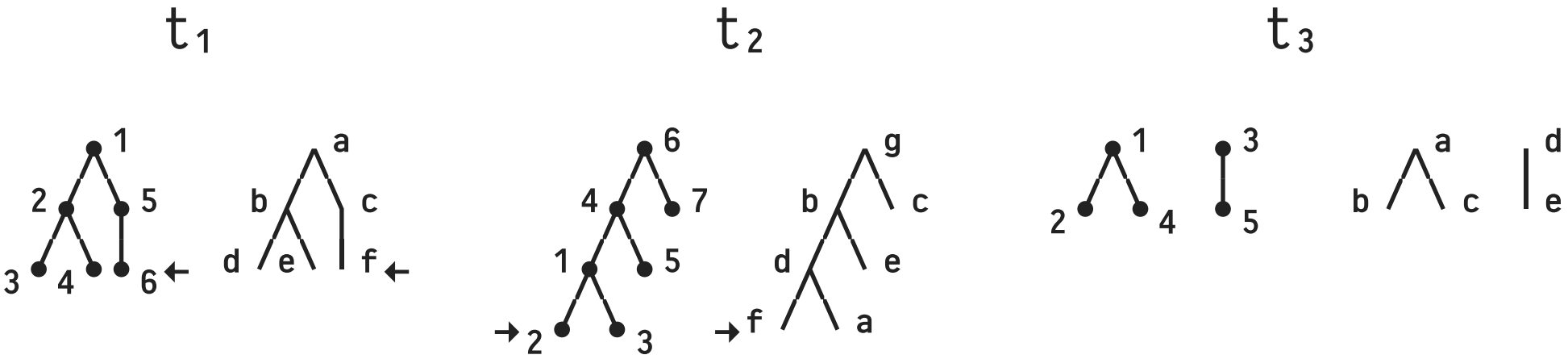
With UNION ALL, recursive query $q\eth$ sees ***all*** **rows added in the last iteration/recursive call:**

```
iterate^all(qð, q₀):
  r ← q₀
  t ← r
  while t ≠ ∅
  | t ← qð(t)
  | r ← r ⊎ t
  return r
```

```
recurse^all(qð, r):
  if r ≠ ∅ then
  | return r ⊎ recurse^all(qð, qð(r))
  else
  | return ∅
```

- Invoke the recursive variant via $\text{recurse}^{\text{all}}(q\eth, q_0)$.
- $\uplus$ denotes bag (multiset) union.
- ⚠️ Need to be extra careful to control termination.

1 Traverse the Paths from Nodes 'f' to their Root



Array-based tree encoding (parent of node *n* ≡ parents[*n*]):

Trees

tree	parents (□ ≡ NULL)	labels
t_1	{□, 1, 2, 2, 1, 5}	{'a','b','d','e','c','f'}
t_2	{4, 1, 1, 6, 5, □, 6}	{'d','f','a','b','e','g','c'}
t_3	{□, 1, □, 1, 3}	{'a','b','d','c','e'}
	1 2 3 4 5 6 7	1 2 3 4 5 6 7 ←-- node

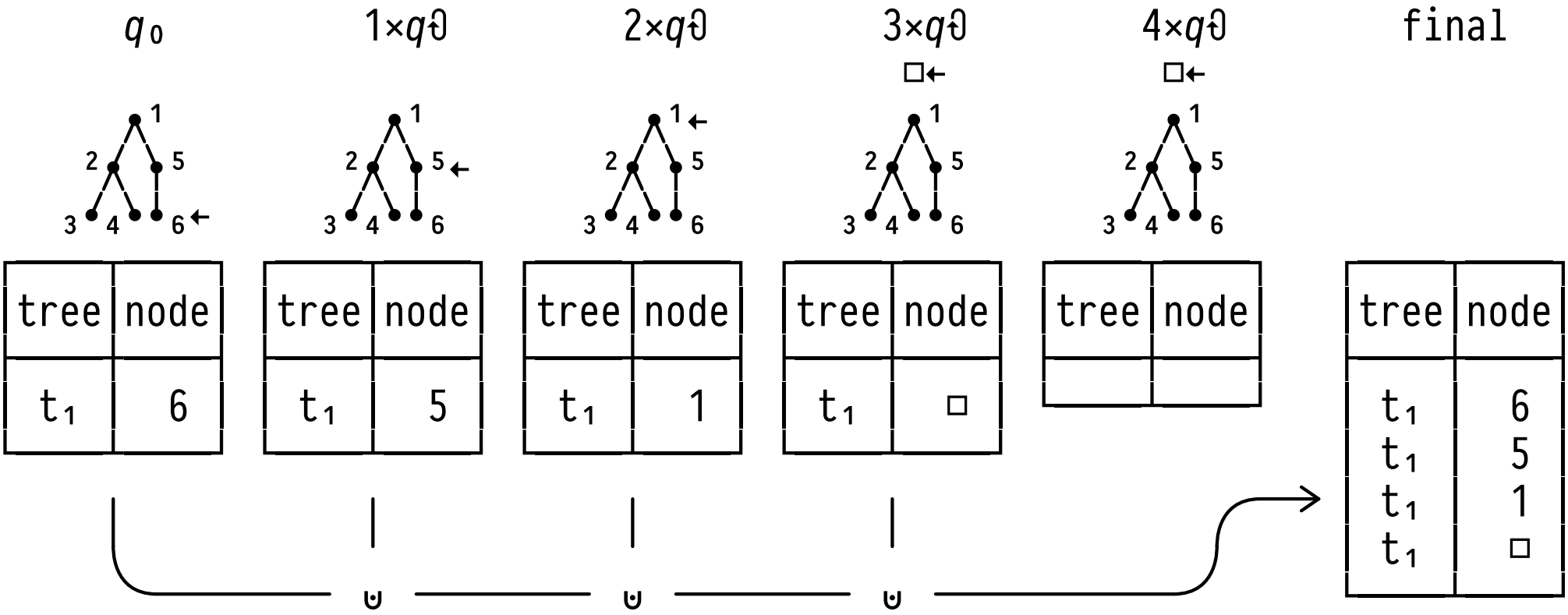
Traverse the Paths from Nodes 'f' to their Root

```
WITH RECURSIVE
  paths(tree, node) AS (
    SELECT t.tree, array_position(t.labels, 'f') AS node
    FROM   Trees AS t

      UNION

    SELECT t.tree, t.parents[p.node] AS node
    FROM   paths AS p,
           Trees AS t
    WHERE  p.tree = t.tree
)
TABLE paths
```

(*t*,*n*) ∈ paths ⇔ node *n* lies on path from 'f' to *t*'s root

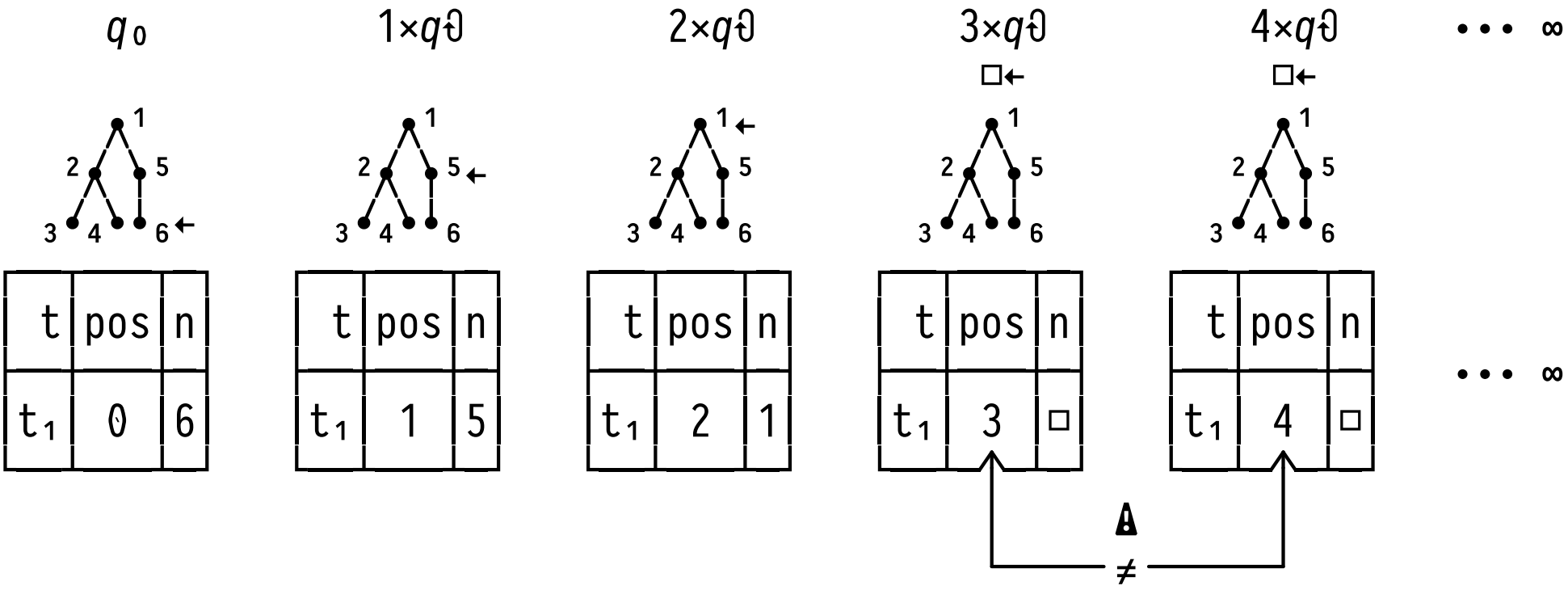
A Trace of the Path in Tree t_1

New rows produced by…



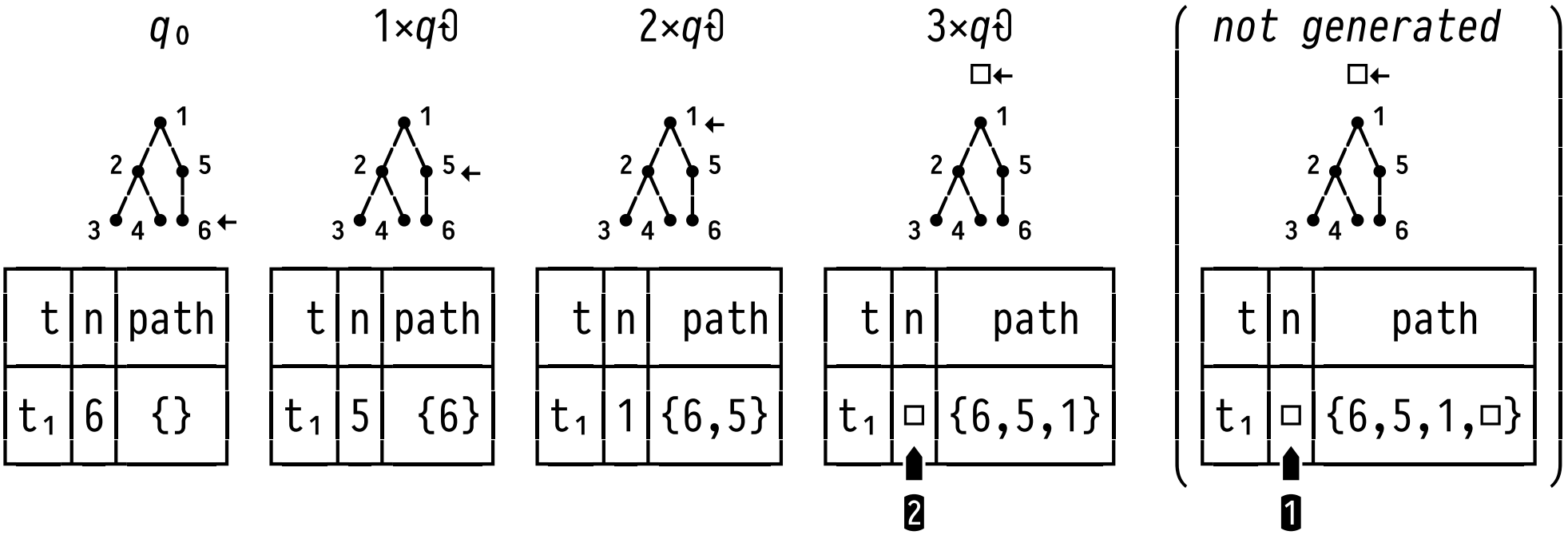
- $4\times q\wr$ yields no new rows (recall: t.parents[NULL] ≡ NULL).

Ordered Path in Tree t_1 (New Rows Trace)



The **(non-)generation of new rows to ensure termination** is the user's responsibility — a common source of .

Path as Array in Tree t_1 (New Rows Trace)



❶ Ensure termination (enforce ϕ): filter on $n \neq \square$ in $q\vartheta$.

❷ Post-process: keep rows of last iteration ($n = \square$) only.