

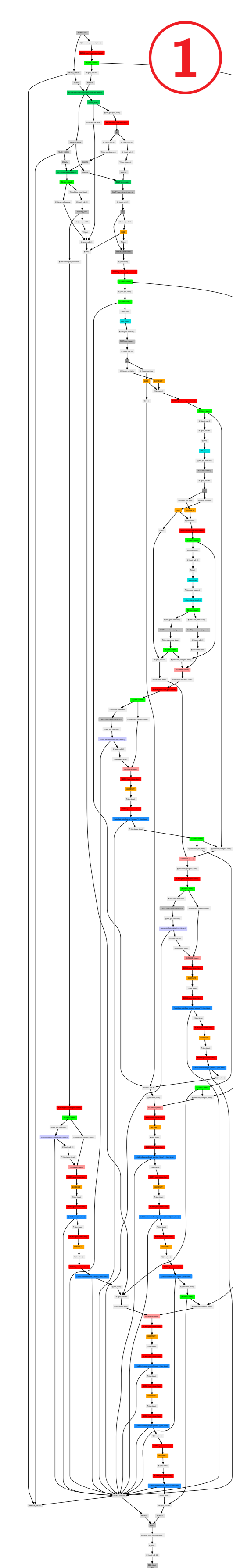
Abstract

Relational encodings of the **static** aspects of the XQuery data model, i.e., tabular representations for XML documents and ordered sequences of items, are widely used today. Since 2002, the **Pathfinder** and **MonetDB/XQuery** companion projects pursue the primary goal to also embrace the complete **dynamic semantics** of XQuery (expression evaluation and runtime aspects) with the help of relational database systems. This makes proven optimization techniques immediately applicable to the construction

of XQuery processors and leads to unprecedented scalability in **MonetDB/XQuery**. This is a demonstration of the relational optimizer of **Pathfinder**, the query compiler behind MonetDB/XQuery. To account for the significant size and unusual shape of the relational query plans derived from input XQuery expressions, Pathfinder implements various optimization techniques in a **peephole-style** fashion and provides support for **graph-shaped plans** from the ground up.

Our demonstration system shows **graphical representations of the relational query plans** and allows the inspection of plan characteristics at various stages of Pathfinder's highly-configurable optimizer pipeline. Stages may be separately enabled to judge their impact on plan quality and XQuery evaluation performance. Our system is preloaded with various XML instances (up to 1 GB serialized size), against which users may run ad-hoc queries in an interactive fashion.

Loop-Lifting: Turning XQuery into Relational Plans



Loop-lifted sequence representation

- Encode independent iterations in a **single** relation.
- Maintain **sequence order** by means of column *pos*.
- Identify independent **iterations** with column *iter*.
- E.g.,

```
for $x in (5, 6, 7) return $x to 7
```

- This is the encoding used for **all** XQuery subexpressions.
- Each subexpression is represented in dependence of its enclosing for iterations.

iter	pos	item
1	1	5
1	2	6
1	3	7
2	1	6
2	2	7
3	1	7

Loop-lifting turns arbitrary XQuery expressions into purely relational plans.

- Fully compositional** compilation procedure.
- Trade **iteration** for efficient **set-oriented** processing.
- Yet, strict **adherence** to the W3C Recommendation.

We particularly benefit from

- an implementation of **staircase join** in MonetDB,
- relational **indexing technology**, and
- OLAP ranking functionality**.

Supported XQuery dialect and target language (Relational Algebra) of the Pathfinder compiler (excerpt):

atomic literals
sequences (e_1, e_2)
variables ($\$v$)
let $\$v := e_1$ return e_2
for $\$v$ at $\$p$ in e_1 return e_2
if e_1 then e_2 else e_3
typeswitch clauses
element { e_1 } { e_2 }
text { e }
 e_1 order by e_2, \dots, e_n
XPath ($e/\alpha::\nu$)
user defined functions

document order ($e_1 \ll e_2$)
node identity ($e_1 \text{ is } e_2$)
arithmetics (+, -, ...)
comparisons (eq, lt, ...)
Boolean operators (and, or, ...)
fn:doc(e), fn:root(e)
fn:id(e), fn:idref(e)
fn:data(e)
fn:distinct-doc-order(e)
fn:count(e), fn:sum(e)
fn:empty(e)
fn:position(), fn:last()

π column projection, renaming ($_$)
 σ row selection (■)
 \cup disjoint union (■)
 \setminus difference (■)
 δ duplicate elimination (■)
 \times Cartesian product (■)
 \bowtie equi-join (■)
 ρ row-numbering (■)
 \Join staircase join (■)
 ϵ, τ element/text node construction (■)
 \otimes arithmetic/comparison/Boolean operator * (■)

Two XQuery Examples

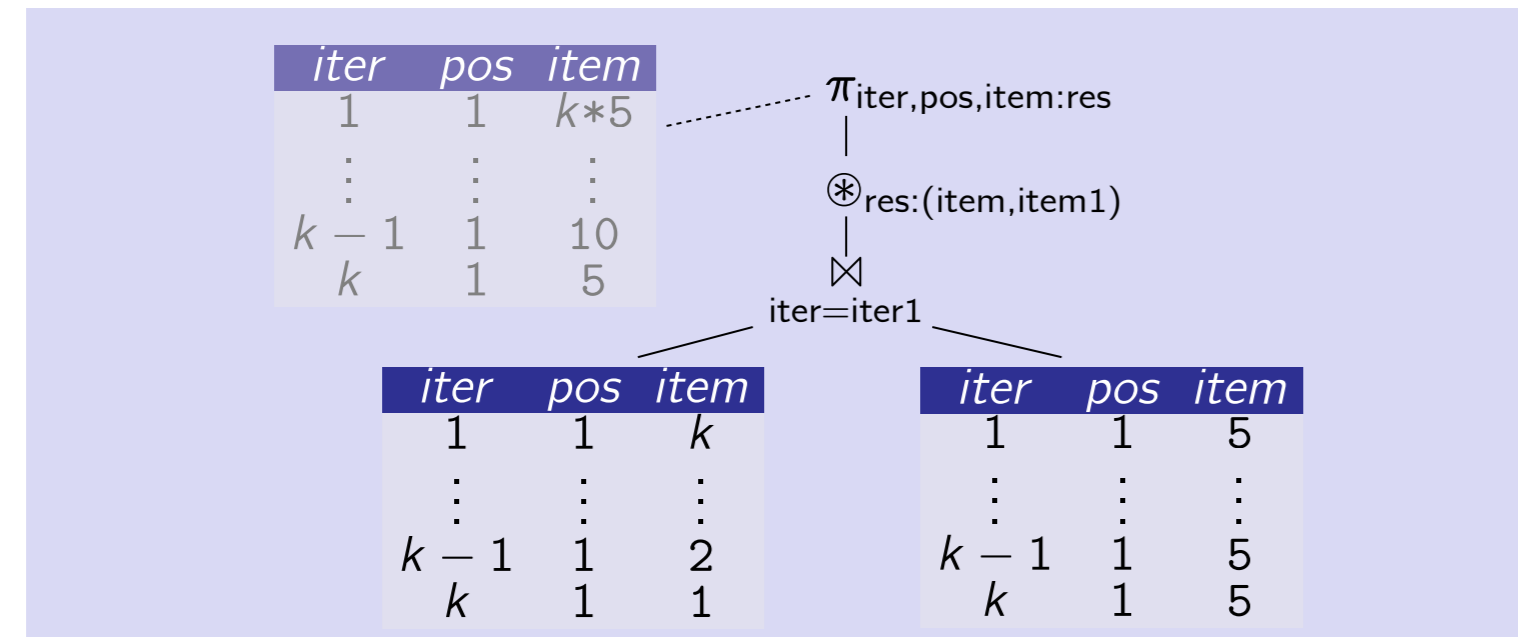
Basic XQuery FLWOR clause

```
for $x in (k, ..., 2, 1) return $x * 5
```

Encode iteration in column iter:

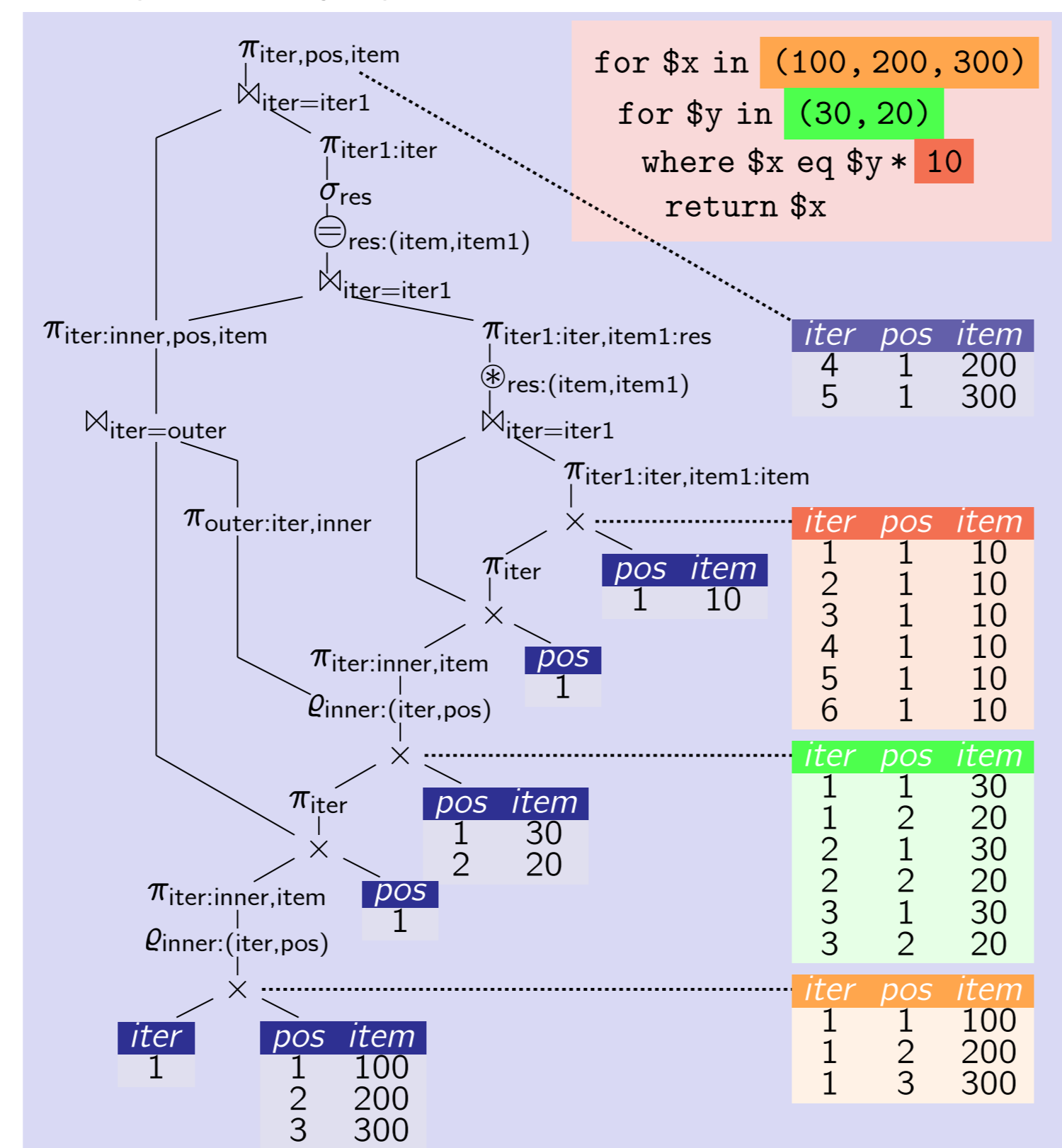
(k, ..., 2, 1)		\$x	5	
pos	item	iter	pos	item
1	k	1	1	5
⋮	⋮	⋮	⋮	⋮
k-1	2	k-1	1	5
k	1	k	1	5

Process for clause in bulk:



◀ The large graph printed on the left shows the loop-lifted query plan for Query Q8 from the XMark benchmark prior to optimization.

Complete XQuery expression

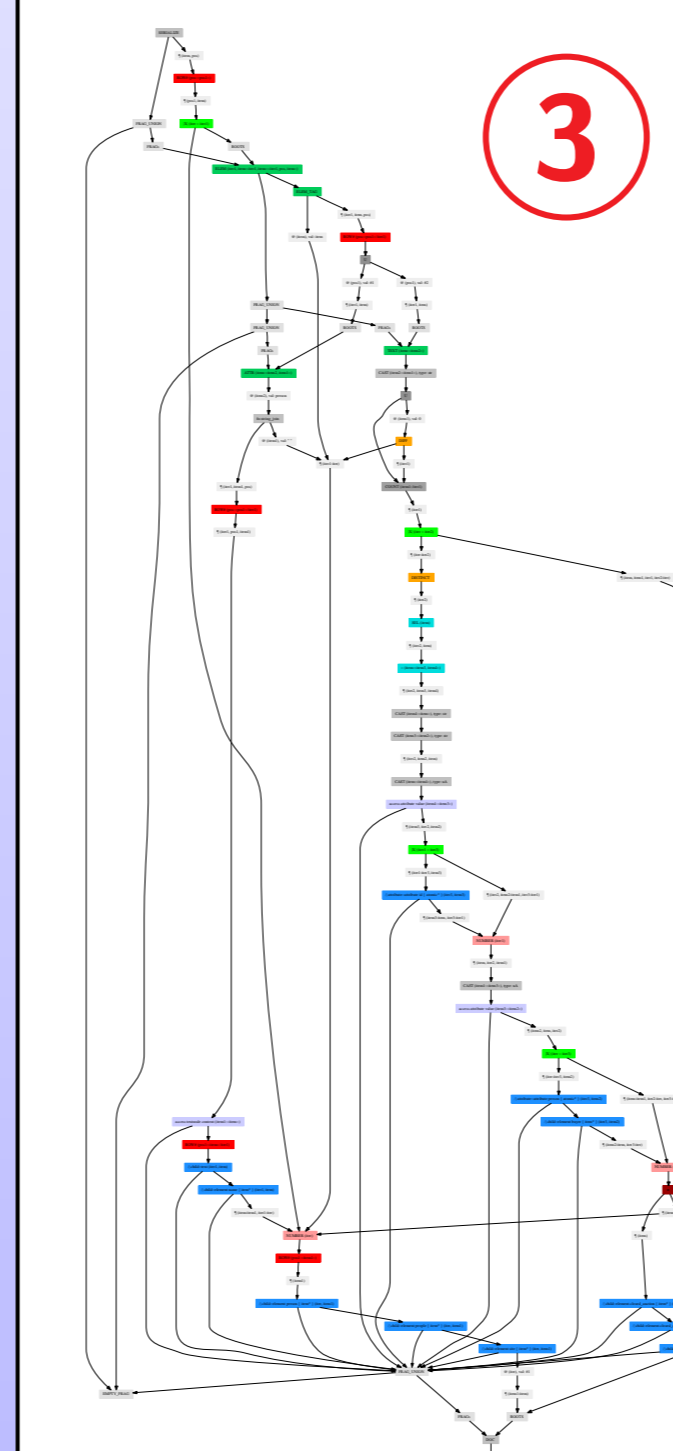


Rewriting Large Plan DAGs

Adapt techniques known from programming language compilers.

- Identify **basic blocks** to maintain focus during optimization.
- Peephole-style optimization**: inspect single plan operators at any time.
- Optimizer is guided by **plan annotations** obtained during a separate **property inference phase**.

These techniques help Pathfinder scale to the involved plan sizes.



Functional and Multi-Valued Dependencies

- Due to loop-lifting, invariant subexpressions surface as **functional and multi-valued dependencies**.
- Pathfinder tracks **degenerate functional dependencies** $\emptyset \rightarrow c$ for columns c that carry a constant value.
- The **degenerate multi-valued dependency** $\emptyset \twoheadrightarrow c_1, \dots, c_n$ indicates the **independence** of c_1, \dots, c_n from all remaining columns.

◀ Exploiting functional dependencies and data flow analyses further reduces the plan size for XMark Q8 to the plan shown on the left.

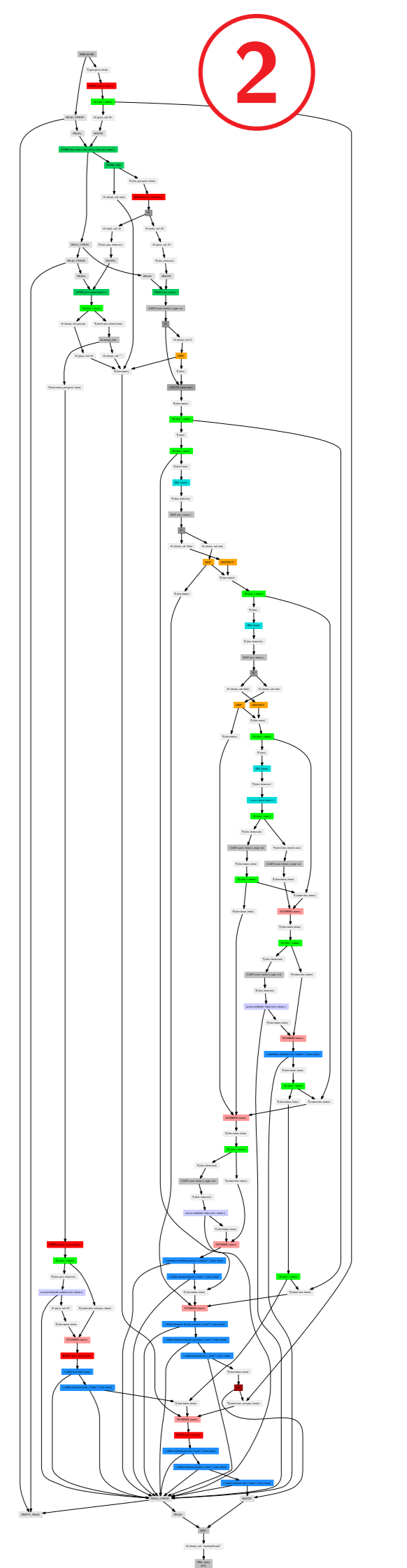
Table Column Pruning

- Compositional compilation often leads to redundant columns.
- Information on **strictly required columns** enables the peephole-style implementation of **projection pushdown**.

Constant propagation

- Pathfinder avoids the generation of columns at runtime if they carry constant values only.

Applied to XMark Q8, column pruning and constant propagation lead to the plan shown on the right.



Data Flow Analyses Based on Active Domains

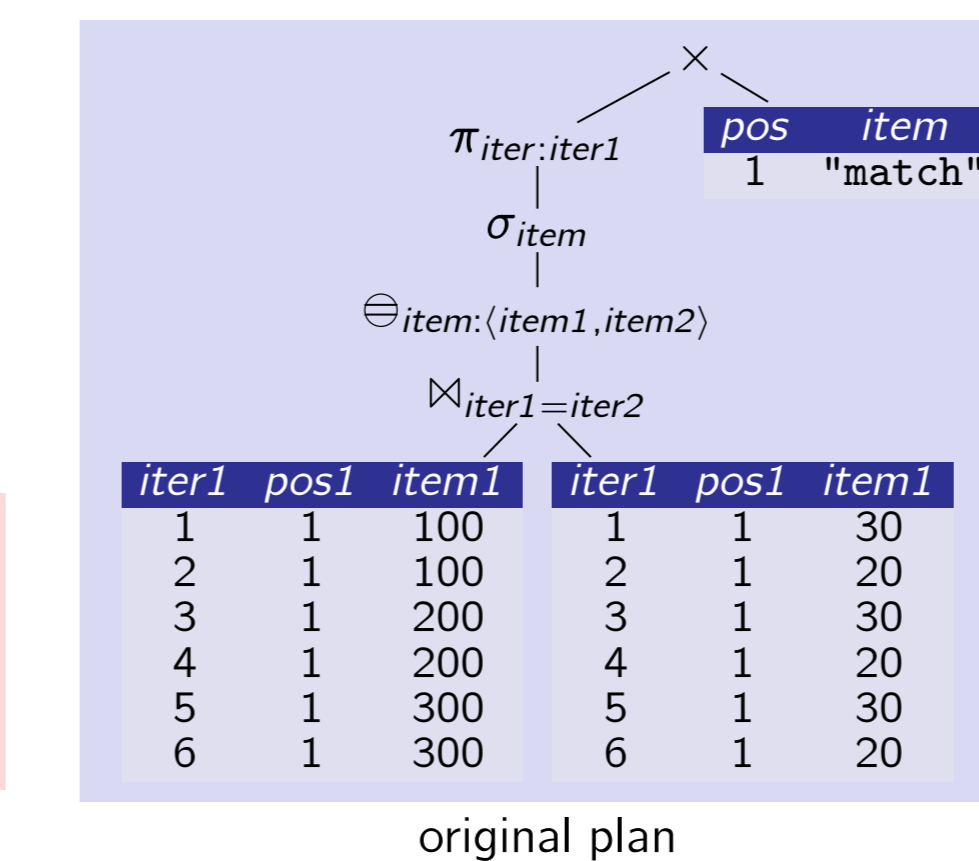
- Pathfinder infers an approximation α_c of the **active domain** for each column c .
- The **inclusion** of active domains $\alpha_{c_1} \subseteq \alpha_{c_2}$ then indicates a **data flow** between respective plan operators.
- Essential also for a derivation of **cardinality estimates** for arbitrary subexpressions.

Algebraic Join Detection and Order Awareness

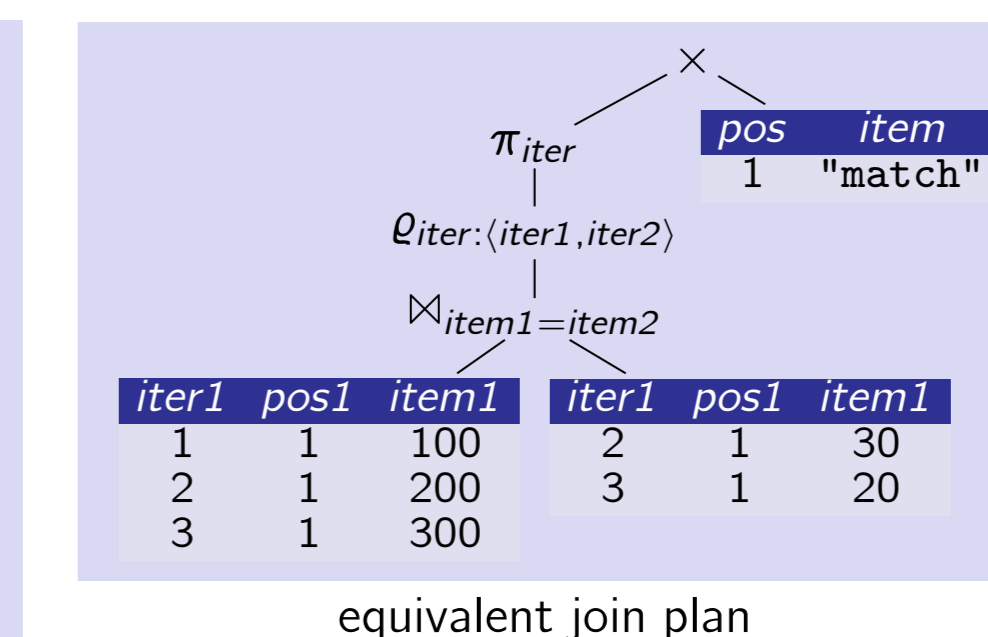
Algebraic Join Detection

- Algebraic **join recognition** based on multi-valued dependencies
- Indifferent to syntactic variations** in the original query text

```
for $x in (100, 200, 300)
for $y in (30, 20)
where $x eq $y * 10
return "match"
```

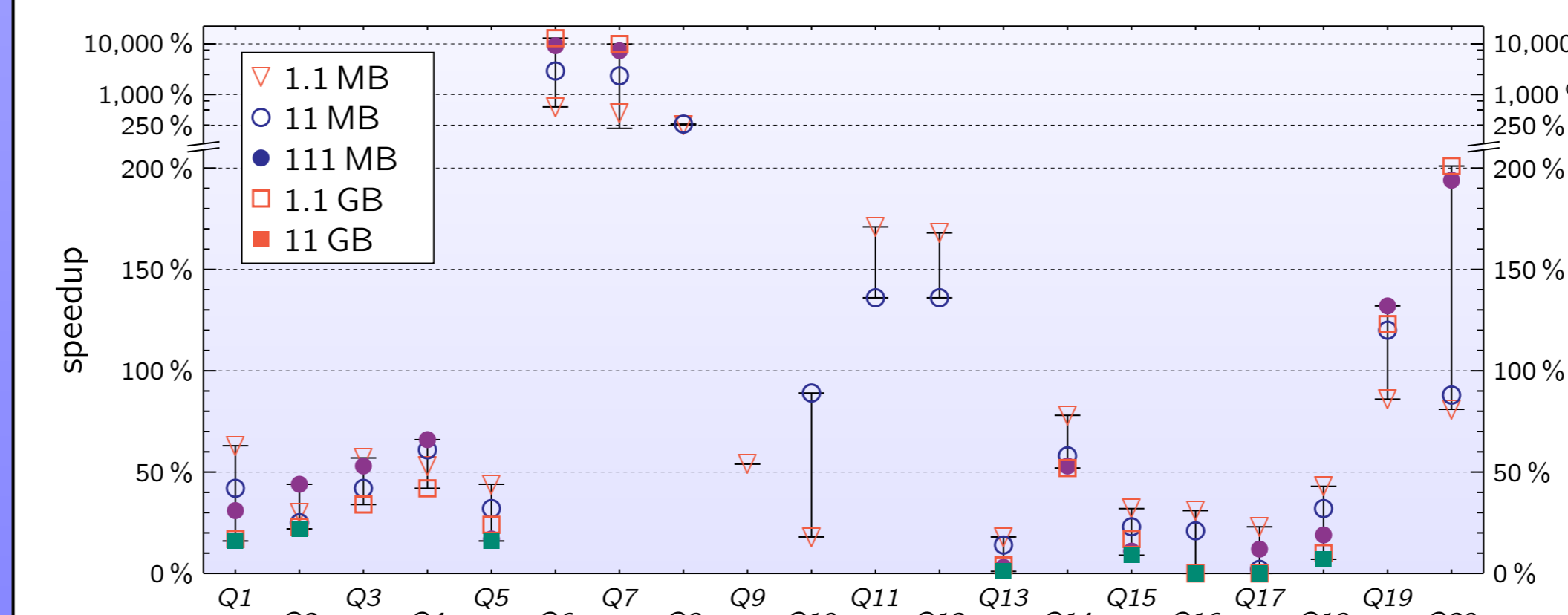
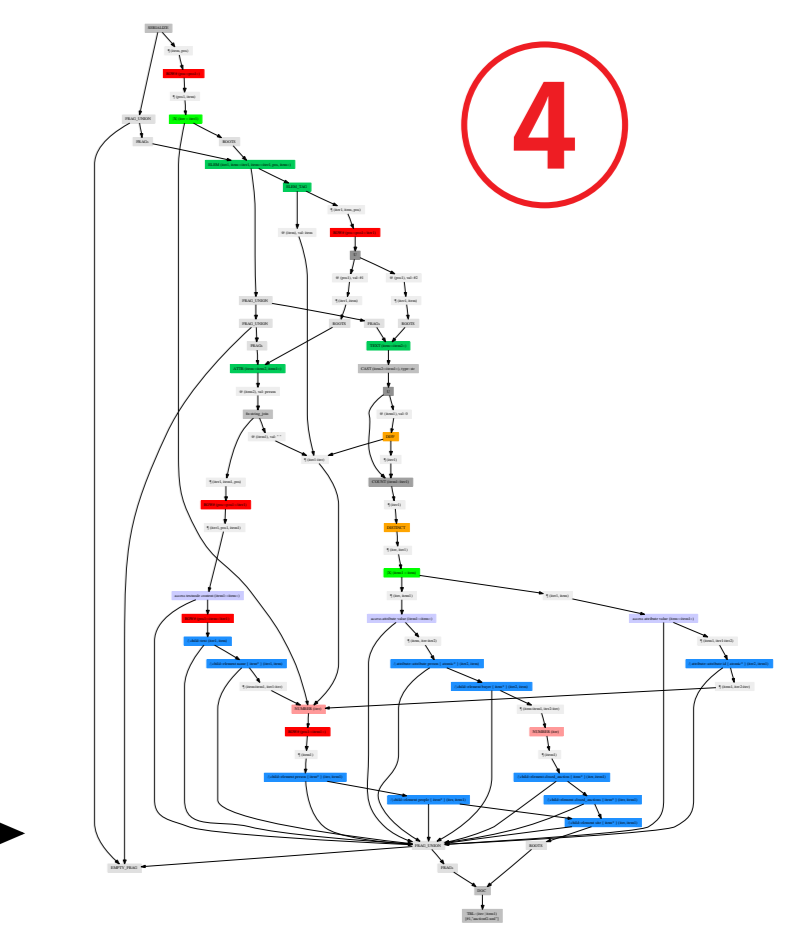


original plan



equivalent join plan

With joins rewritten this way, XMark Q8 now corresponds to the plan on the right.



Order-Related Optimization Opportunities

- Standards-compliant treatment of order can be costly on relational back-ends.
- In return, benefit whenever order is **not** significant, e.g.:
 - context set of XPath location steps**,
 - quantifiers** some and every,
 - XQuery general comparisons** (=, <, ...) and aggregates.
- The consideration of order can **speed up** query evaluation by **orders of magnitude** (see left).