Database Languages and their Compilers

Prof. Dr. Torsten Grust

Database Systems Research Group
U Tübingen

Winter 2010

5 Query Plan Generation

- If we inspect the algorithms provided by a relational DBMS query engine, we typically will **not** find *fold*.

- Instead, query engines provide a small set of basic **query operators**, commonly referred to as the **relational algebra**.

- The operators of the relational algebra are chosen so that

  ① (**Expressivity**) any SQL query can be computed if we build an appropriate tree of relational algebra operators, and
  ② (**Efficiency**) each operator may be implemented efficiently on top of the buffer manager, index structures, and file operations provided by the DBMS.

- This chapter will discuss how we can **generate relational algebra operator trees from MCC expressions**.

5.1 Basic Relational Algebra Operators

- For our discussion, we will assume the presence of the following query algebra.
- Since SQL is a query language operating on **bags** of tuples (notable exceptions are `DISTINCT` and `ORDER BY`), we will introduce relational algebra as an algebra over bags.

5.1.1 Access - α

- Accessing a relation on secondary storage is always the first step before query execution can begin. This is what operator α provides.
  - **Synopsis**:

    $$\alpha\ (id)$$

    Argument id is the **name of a relation registered in the DBMS catalog**. Note that bogus values for id would have been caught by the type checker (*name-unknown-*↯, see Section 3.3).
  - **Example** (see Chapter 0):

α (Mail) =

sender	subject	domain
'Alice'	'I love you'	'.nl'
'Bob'	'Re: You are fired'	'.uk'
'Cathy'	'You are fired'	'.uk'
'Dave'	'I love you'	'.de'
⋮	⋮	⋮

5.1.2 Projection – π

- Operator π projects its input relation onto a list of given attributes or computes new attributes from old ones. Duplicates are retained.
  - **Synopsis**:

$$\pi_L\,(r)$$

The **projection list** L is $p_1, p_2, \ldots, p_n$, where each p_i is of the form $e_i \to l_i$, l_i being the name of the new result attribute and e_i being an expression involving attribute names (of input r), constants, and arithmetic operators.

  - **Example**:

r	a	b	c
	0	1	2
	0	1	2
	3	4	5
	0	0	3

, $\pi_{a \to a, b+c \to x}\,(r) =$

	a	x
	0	3
	0	3
	3	9
	0	3

.

  - Projection items $a \to a$ may be abbreviated by a.

5.1.3 Selection – σ

- Selection σ operates as a filter that rejects all incoming tuples who fail to satisfy a given predicate (expression of type $\mathbb{B}$). Duplicates are retained.

  - **Synopsis**:

    $$\sigma_C\ (r)$$

    The **selection condition** $C :: \mathbb{B}$ may be built from attribute names (present in r), constants, arithmetic and relational $(<, \leq, =, \dots)$ operators, and boolean connectives $\neg, \vee, \wedge$).

  - **Example**:

r	a	b	c
	0	1	2
	0	1	2
	3	4	5
	0	0	3

, $\quad \sigma_{a+b>c\ \vee\ b=1}\ (r) \quad =$

	a	b	c
	0	1	2
	0	1	2
	3	4	5

.

5.1.4 Cross Product – $\times$

- Operator $\times$ forms all combinations of tuples – via tuple merging ($\|$) – of its two inputs (cartesian product). Duplicates are retained.
  - **Synopsis**:

    $$r \times s$$

    Note that, since result tuples are built via $\|$, $\times$ is commutative as well as associative.
  - **Example**:

r	a	b
	0	1
	2	3
	2	3

s	c	d
	1	4
	1	4
	2	5

, $r \times s =$

	a	b	c	d
	0	1	1	4
	0	1	1	4
	0	1	2	5
	2	3	1	4
	2	3	1	4
	2	3	2	5
	2	3	1	4
	2	3	1	4
	2	3	2	5

.

5.1.5 Duplicate Elimination – δ

- The duplicate eliminator δ removes duplicate tuples – decided via tuple equality $(=)$ – in its input, thus producing a set of tuples.
  - **Synopsis:**

$$\delta\ (r)$$

  - **Example:**

r	a	b
	0	1
	2	3
	2	3

s	c	d
	1	4
	1	4
	2	5

$$,\qquad \delta\ (r\ \times\ s)\quad =$$

	a	b	c	d
	0	1	1	4
	0	1	2	5
	2	3	1	4
	2	3	2	5

$$.$$

> ✎ **Implementing δ?**
> How would you try to efficiently implement the δ operator?

5.1.6 Grouping – γ

- γ partitions its input relation using a list of grouping attributes and optionally applies aggregates to each partition.
  - **Synopsis**:

$$\gamma_{G|A}\ (r)$$

$G = g_1, \ldots, g_r$ is the list of grouping attributes (of input r). The optional $A = agg_1(a_1) \to l_1, \ldots, agg_n(a_n) \to l_n$ describes the list of aggregations to perform on each partition ($agg_i \in \{sum, count, max, min\}$). $\{g_i\} \cap \{a_j\} = \varnothing$.

  - **Example**:

r	a	b	c
	0	1	4
	0	1	2
	2	3	0
	2	3	6
	0	1	1

,

$$\gamma_{a,b|sum(c)\to s,max(c)\to m}\ (r) \quad = \quad$$

	a	b	s	m
	0	1	3	4
	2	3	6	6

.

5.1.7 Bag Operations – $\uplus$, $\dot{-}$, $\cap$

- These binary operators respect the multiplicity of elements in their inputs ($\uplus$ adds multiplicities, $\dot{-}$ subtracts, $\cap$ preserves the minimum multiplicity).
  - **Synopses:**

$$r \quad \uplus \quad s$$
$$r \quad \dot{-} \quad s$$
$$r \quad \cap \quad s$$

    The tuple types of the elements of r, s have to be identical.
  - **Examples:**

r	a	b
	0	1
	0	1
	2	3

s	a	b
	0	1
	2	3
	2	3
	4	5

,

$r \uplus s$	a	b
	0	1
	0	1
	0	1
	2	3
	2	3
	2	3
	4	5

$r \dot{-} s$	a	b
	0	1

$r \cap s$	a	b
	0	1
	2	3

.

5.1.8 Sorting – τ

- Operator τ imposes an ordering on the elements of a collection, thus producing a list of tuples.
  - **Synopsis**:

$$\tau_L\ (r)$$

    $L = s_1, \ldots, s_n$, the **sort criteria**, define a lexicographic ordering on the tuples of r; $s_i = (a, \leq)$ or $s_i = (a, \geq)$ to specify ascending/descending sort order for attribute a.
  - **Example**:

r	a	b
	1	4
	0	1
	0	2
	1	3

,

$$\tau_{(a,\leq),(b,\geq)}\ (r) \quad =$$

	a	b
	0	2
	0	1
	1	4
	1	3

.

5.2 The Construction of Algebraic Operator Trees

- As input to the plan generation phase we assume a normalized and simplified (see Chapter 4) expression of MCC.
- Before we present a systematic approach to generate relational algebra operator trees from MCC expressions, let us try to develop an intuition for the problem.

5.2.1 Step 1: Algebraic Operators and MCC

- In the previous section, we could have introduced the operators of relational algebra by **defining them through MCC expressions**, e.g., like this:

$$\begin{aligned}
\pi_{e_1 \to l_1, \ldots, e_n \to l_n}\ (r) &\stackrel{\text{def}}{=} \left[\, (l_1 : e_1, \ldots, l_n : e_n) \mid v \leftarrow r \,\right]^{bag} \\
\sigma_p\ (r) &\stackrel{\text{def}}{=} \left[\, v \mid v \leftarrow r, p \,\right]^{bag} \\
r \ \times \ s &\stackrel{\text{def}}{=} \left[\, v_1 \mid\mid v_2 \mid v_1 \leftarrow r, v_2 \leftarrow s \,\right]^{bag}
\end{aligned}$$

✎ **Operator δ's definition in terms of MCC?**
How would you express $\delta\ (r)$ in the monoid comprehension calculus?

- The same can be done for γ (refer to case (j) in the definition of $\mathbb{T}$).
- Operators $\uplus, \oplus, \setminus^+$ as well as $\tau\ (= sort)$ have already been introduced by $\mathbb{T}$ so that these need no further translation.

5.2.2 Step 2: Algebraic Operator Trees and MCC

- Now that we have seen that each operator on its own has a counterpart in MCC, let us try to map more complex MCC expressions to algebraic operator trees.

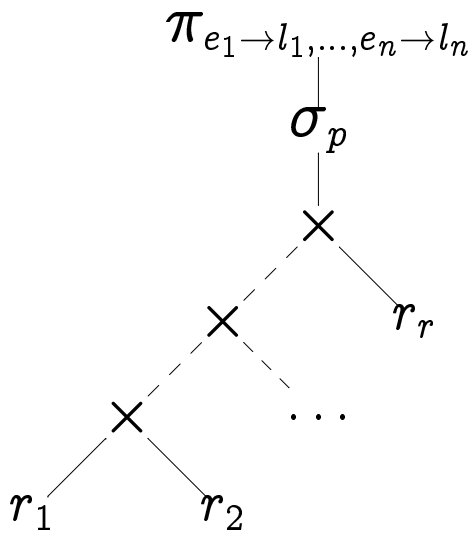
✎ **Which tree of algebraic operators is equivalent to the comprehension below ($R :: bag\ (a : \mathbb{S}, b : \mathbb{N})$, $S :: bag\ (c : \mathbb{N}, d : \mathbb{B})$)?**

$$\left[\, (a : x.a, d : y.d) \mid x \leftarrow R,\ y \leftarrow S,\ x.b = y.c \,\right]^{bag}$$

- To generalize, remember that **comprehension normalization** generates MCC terms of the following form:

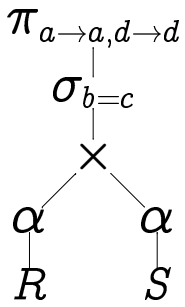
$$\left[\,(l_1 : e_1, \ldots, l_n : e_n) \mid v_1 \leftarrow r_1,\ v_2 \leftarrow r_2,\ \ldots,\ v_r \leftarrow r_r,\ p\,\right]^{bag}$$

  - Nested comprehensions have been unnested as far as possible, i.e., most r_i will simply be relation names;
  - occurring predicates p_i have been pushed to the far right, with $p = \bigwedge_i p_i$.
  - (Left-deep) **Algebraic operator tree**:



- To **verify that this translation from MCC into algebra is indeed correct**, simply replace the algebraic operators with their MCC definitions, then normalize. We should get back the original MCC expression.

- **Example** (using the query shown at the beginning of this section):

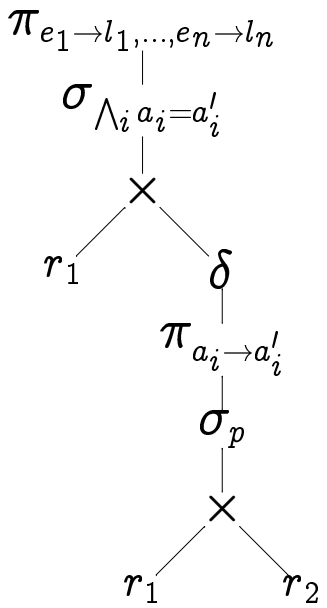


$$
\begin{aligned}
&\underset{\text{MCC definitions}}{=} && \big[\, (a : a, d : d) \mid v \leftarrow \big[\, v' \mid v' \leftarrow \underbrace{\big[\, x \mathbin{||} y \mid x \leftarrow R, y \leftarrow S \,\big]^{bag}}_{\text{unnest via rule (4)}}, b = c \,\big]^{bag} \,\big]^{bag} \\
&= && \big[\, (a : a, d : d) \mid v \leftarrow \big[\, v' \mid x \leftarrow R, y \leftarrow S, v' \equiv x \mathbin{||} y, b = c \,\big]^{bag} \,\big]^{bag} \\
&\underset{\text{remove } \equiv}{=} && \big[\, (a : a, d : d) \mid v \leftarrow \underbrace{\big[\, x \mathbin{||} y \mid x \leftarrow R, y \leftarrow S, b = c \,\big]^{bag}}_{\text{unnest via rule (4)}} \,\big]^{bag} \\
&= && \big[\, (a : a, d : d) \mid x \leftarrow R, y \leftarrow S, b = c, v \equiv x \mathbin{||} y \,\big]^{bag} \\
&\underset{\text{remove } \equiv}{=} && \big[\, (a : a, d : d) \mid x \leftarrow R, y \leftarrow S, b = c \,\big]^{bag} \\
&\underset{\text{explicit use of variables}}{=} && \big[\, (a : x.a, d : y.d) \mid x \leftarrow R, y \leftarrow S, x.b = y.c \,\big]^{bag} \quad .
\end{aligned}
$$

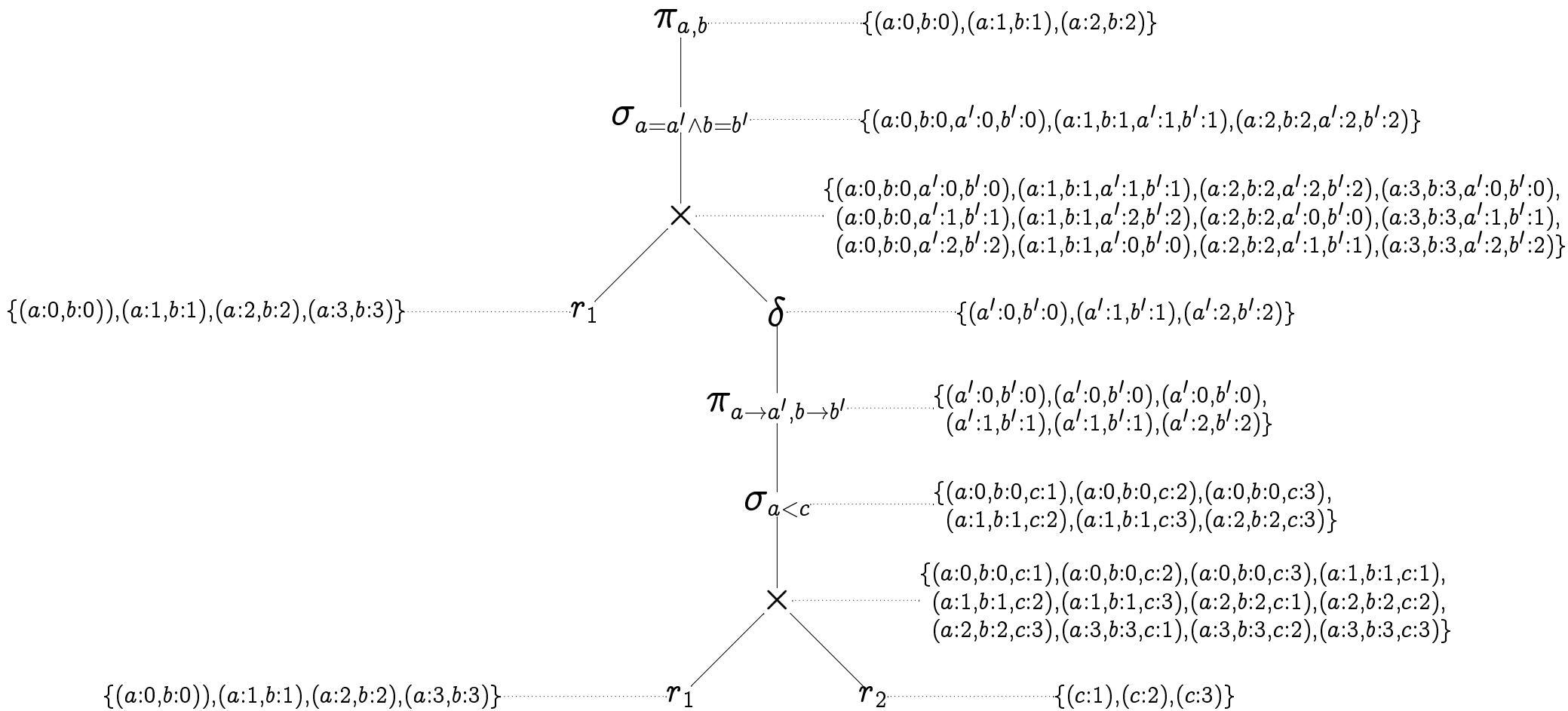
- Complex predicates (in an SQL `WHERE` clause) may lead to **existential and universal quantifiers** in MCC expressions. Operator σ cannot deal with these directly:
  - **Existential quantification**:

$$\left[\,(l_1 : e_1, \ldots, l_n : e_n) \mid v_1 \leftarrow r_1, \left[\,p \mid v_2 \leftarrow r_2\,\right]^{some}\,\right]^{bag}$$

  - Semantics: a tuple v_1 of r_1 contributes to the result if there exists **any** v_2 in r_2 so that p is satisfied (if we find two or more v_2 this makes no difference).
  - **Algebraic operator tree** $(r_1 :: bag\ (a_1 : \tau_1, \ldots, a_r : \tau_r))$:



$$\mathbb{T}\left(\begin{array}{ll}\texttt{SELECT} & x.a, x.b \\ \texttt{FROM} & r_1\ \texttt{AS}\ x \\ \texttt{WHERE} & \texttt{EXISTS}\ y\ \texttt{IN}\ r_2 : x.a < y.c\end{array}\right) \quad = \quad \left[\, (a : x.a, b : x.b) \mid x \leftarrow r_1, \left[\, x.a < y.c \mid y \leftarrow r_2 \,\right]^{some} \,\right]^{bag}$$



▶ **N.B.:**

① Adding δ on top of the root of the operator trees makes the previous discussion **applicable to** set **comprehensions** $([\,|\,]^{set})$;

② comprehension normalization (5) unnests existential quantifiers $([\,|\,]^{some})$ inside set comprehensions.

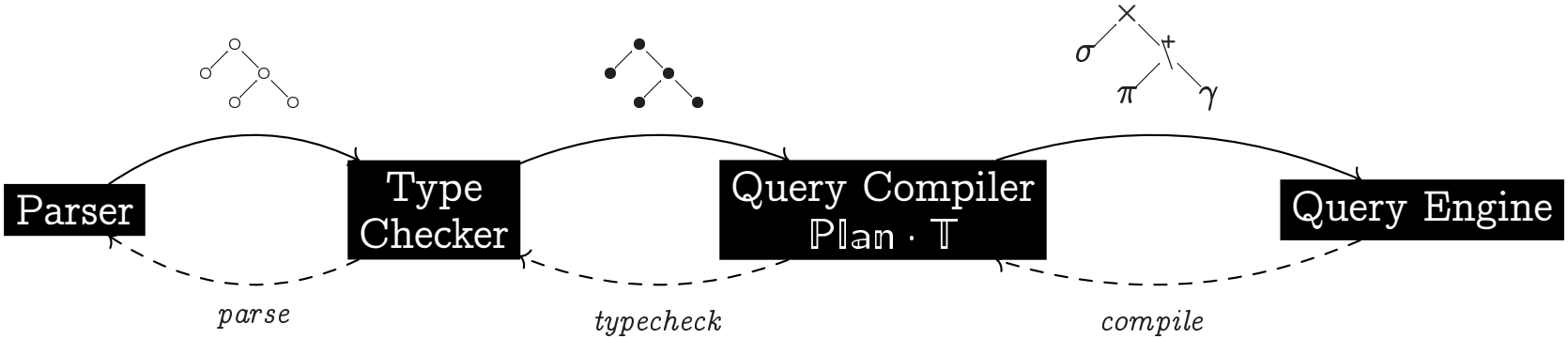
✎ **Universal quantification and relational algebra?**
Try to extend the mapping from MCC to algebraic operator trees so that it correctly generates plans for universal quantifiers:

$$[\,(l_1 : e_1, \ldots, l_n : e_n) \mid v_1 \leftarrow r_1, [\,p \mid v_2 \leftarrow r_2\,]^{all}\,]^{bag} \quad .$$

Hint: $[\,p \mid qs\,]^{all} = \neg[\,\neg p \mid qs\,]^{some}$.

5.2.3 Mapping MCC to Query Plans

- Let us complete this chapter by putting our findings about the generation of algebraic operator trees into a systematic framework.



  - The missing piece is Plan, a function that **maps an expression of MCC to a query plan** ready for execution (or further optimization).
  - To design Plan, we will need to
    ① consider the different expression forms that may occur in MCC;
    ② find an equivalent algebraic operator tree for each of these forms.

▶ Mapping $\mathbb{P}\mathtt{lan}$

Input: MCC expression
Output: Query plan (algebraic operator tree)

$$\mathbb{P}\mathtt{lan}\ (e \uplus e') = \mathbb{P}\mathtt{lan}\ (e) \ \ \mathbb{P}\mathtt{lan}\ (e') \quad (8)$$

$$\mathbb{P}\mathtt{lan}\ (e \setminus^{+} e') = \mathbb{P}\mathtt{lan}\ (e) \ \ \mathbb{P}\mathtt{lan}\ (e') \quad (9)$$

$$\mathbb{P}\mathtt{lan}\ (e \dot{\cap} e') = \mathbb{P}\mathtt{lan}\ (e) \ \ \mathbb{P}\mathtt{lan}\ (e') \quad (10)$$

$$\mathbb{P}\mathtt{lan}\ (sort\ os\ e) = \tau_{os} \ \mathbb{P}\mathtt{lan}\ (e) \quad (11)$$

$$\mathbb{P}\mathtt{lan}\ ([\, e \mid qs \,]^{bag}) = \mathbb{P}\ (\{()\}, [\, e \mid qs \,]^{bag}) \quad (12)$$

$$\mathbb{P}\mathtt{lan}\ ([\, e \mid qs \,]^{set}) = \delta \ \mathbb{P}\mathtt{lan}\ ([\, e \mid qs \,]^{bag}) \quad (13)$$

$$\mathbb{P}\mathtt{lan}\ (R) = \alpha \ R \quad (14)$$

▶ Auxiliary Mapping $\mathbb{P}$

Input: Partial query plan $\times$ Monoid comprehension
Output: Query plan

$$\mathbb{P}\,(e,\ [\,|\ ]^{bag}) \quad = \quad e \tag{15}$$

$$\mathbb{P}\,(e,\ [\,(l_1 : e_1, \ldots, l_n : e_n) \mid\ ]^{bag}) \quad = \quad \underset{\Big|\,e}{\pi_{e_1 \to l_1, \ldots, e_n \to l_n}} \tag{16}$$

$$\mathbb{P}\,(e,\ [\,e' \mid p, qs\,]^{bag}) \quad = \quad \mathbb{P}\left(\underset{\Big|\,e}{\sigma_p},\ [\,e' \mid qs\,]^{bag}\right) \tag{17}$$

$$\mathbb{P}\,(e,\ [\,e' \mid v \leftarrow e'', qs\,]^{bag}) \quad = \quad \mathbb{P}\left(e \times \mathbb{P}\mathrm{lan}\,(e''),\ [\,e' \mid qs\,]^{bag}\right) \tag{18}$$

- Predicate $p :: \mathbb{B}$ above is **simple**, i.e., p contains no quantifiers.
- **N.B.** $\{()\} \times r = r$ for any relation r.

$\mathbb{P}$ *continued on next slide . . .*

- $\mathbb{P}$ is extended to cover **existential and universal quantifiers**:

$$\mathbb{P}\,(e,\ [\,e' \mid [\,p \mid qs\,]^{some}, qs'\,]^{bag}) \quad = \quad \mathbb{P}\left(\begin{array}{c} \pi_{a_i} \\ | \\ \sigma_{\bigwedge_i a_i = a_i'} \\ | \\ \times \\ \swarrow \quad \searrow \\ e \qquad \delta \\ \qquad\quad | \\ \qquad \pi_{a_i \to a_i'} \\ \qquad\quad | \\ \qquad \sigma_p \\ \qquad\quad | \\ \qquad \times \\ \qquad\ \swarrow \quad \searrow \\ \qquad e \qquad \mathbb{P}\mathbb{l}\mathbb{a}\mathbb{n}\,([\,|\ qs\,|]^{bag}) \end{array}\ ,\ [\,e' \mid qs'\,]^{bag}\right) \tag{19}$$

$$\mathbb{P}\,(e,\ [\,e' \mid [\,p \mid qs\,]^{all}, qs'\,]^{bag}) \quad = \quad \mathbb{P}\left(\begin{array}{c} \backslash^{+} \\ \swarrow \quad \searrow \\ e \qquad \pi_{a_i} \\ \qquad\quad | \\ \qquad \sigma_{\neg p} \\ \qquad\quad | \\ \qquad \times \\ \qquad\ \swarrow \quad \searrow \\ \qquad e \qquad \mathbb{P}\mathbb{l}\mathbb{a}\mathbb{n}\,([\,|\ qs\,|]^{bag}) \end{array}\ ,\ [\,e' \mid qs'\,]^{bag}\right) \tag{20}$$

  - Let $e :: bag\ (a_1 : \tau_1, \ldots, a_r : \tau_r)$.

▶ **Example**. Plan compilation (assume $R :: bag\ (a : \mathbb{S}, b : \mathbb{N})$, $S :: bag\ (c : \mathbb{N}, d : \mathbb{B})$):

$$\mathbb{P}\mathrm{lan}\ \Big([\,(a : x.a, b : x.b) \mid x \leftarrow R, [\,x.a < y.c \mid y \leftarrow S\,]^{some}\,]^{bag}\Big)$$

$$\underset{(12)}{=}\quad \mathbb{P}\ \Big(\{()\},\ [\,(a : x.a, b : x.b) \mid x \leftarrow R, [\,x.a < y.c \mid y \leftarrow S\,]^{some}\,]^{bag}\Big)$$

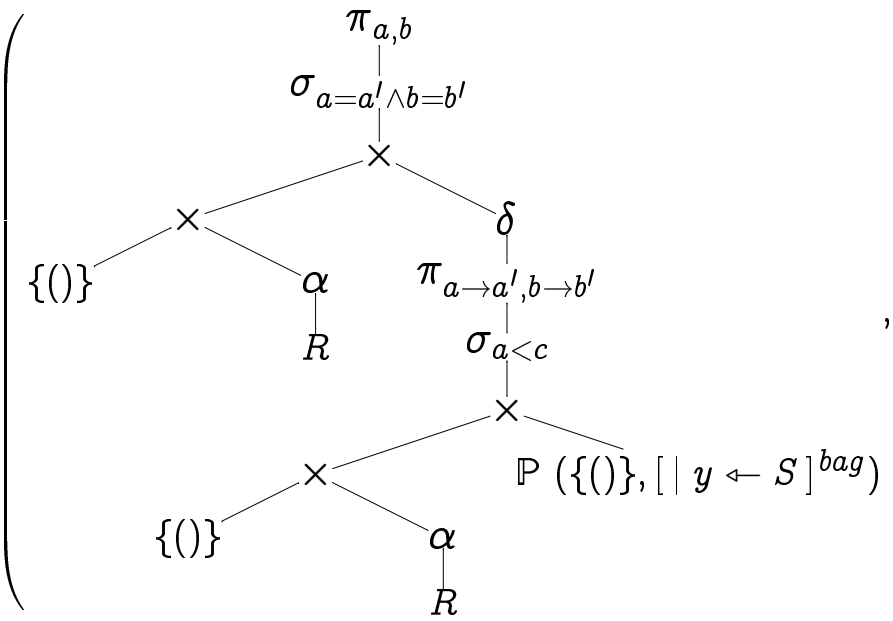
$$\underset{(11)}{=}\quad \mathbb{P}\ \Big(\{()\} \times \mathbb{P}\mathrm{lan}\ (R),\ [\,(a : x.a, b : x.b) \mid [\,x.a < y.c \mid y \leftarrow S\,]^{some}\,]^{bag}\Big)$$

$$\underset{(14)}{=}\quad \mathbb{P}\ \Big(\{()\} \times \alpha(R),\ [\,(a : x.a, b : x.b) \mid [\,x.a < y.c \mid y \leftarrow S\,]^{some}\,]^{bag}\Big)$$

$$\underset{(19)}{=}\quad \mathbb{P}\ \Big(\pi_{a,b}\big(\sigma_{a=a' \wedge b=b'}\big(\{()\} \times \alpha(R)\big) \times \delta\big(\pi_{a \to a', b \to b'}\big(\sigma_{a<c}\big(\{()\} \times \alpha(R)\big) \times \mathbb{P}\mathrm{lan}\ ([\,\mid y \leftarrow S\,]^{bag})\big)\big)\big)\Big)$$

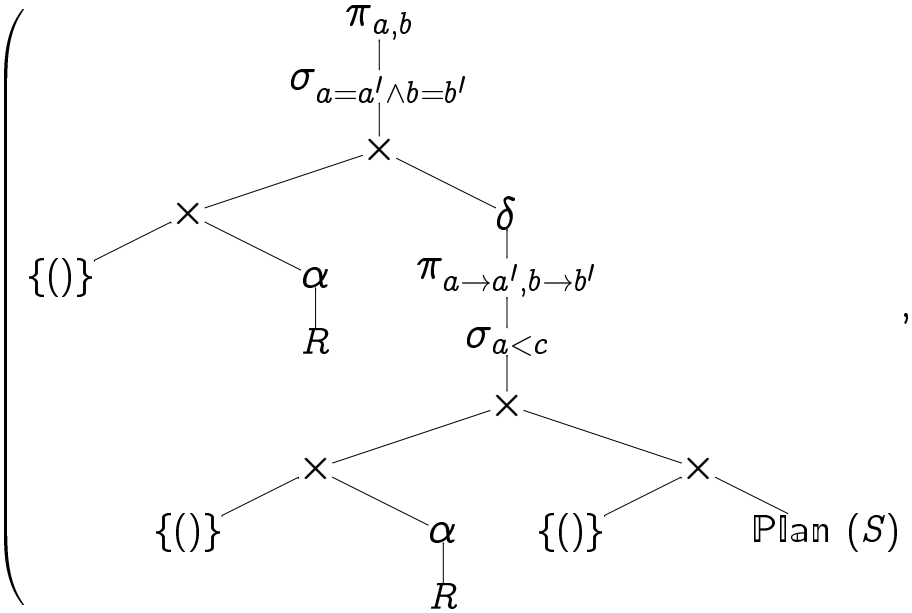
$$,\ [\,(a : x.a, b : x.b) \mid\,]^{bag}$$

$$\underset{(11)}{=} \quad \mathbb{P}\left(\right.$$

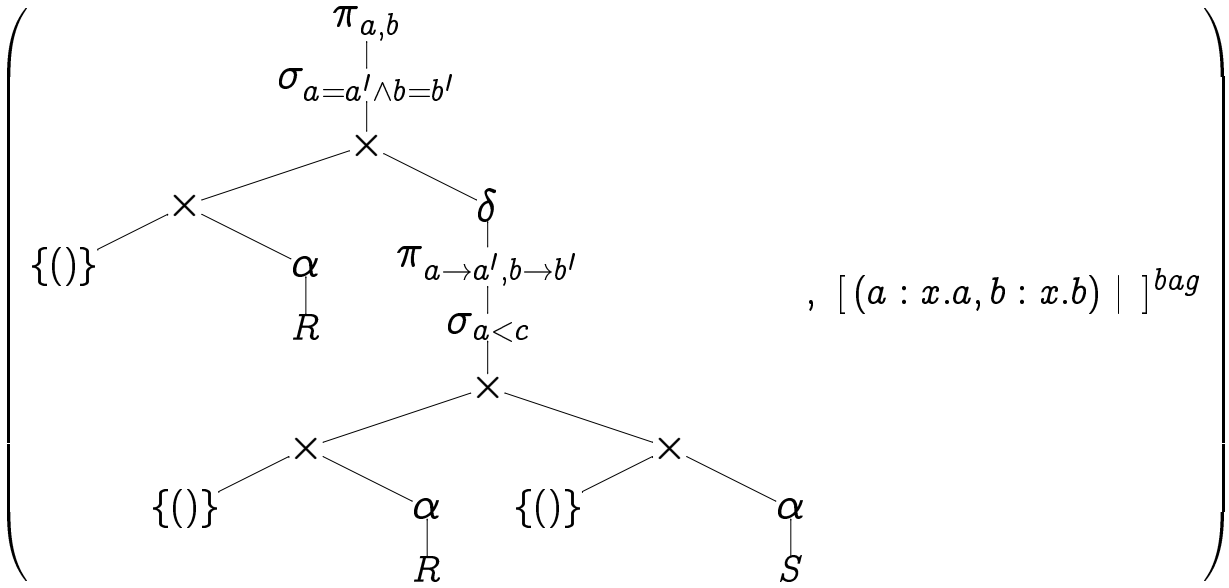


$$\left. , \ [\,(a : x.a, b : x.b)\ |\ ]^{bag} \right)$$

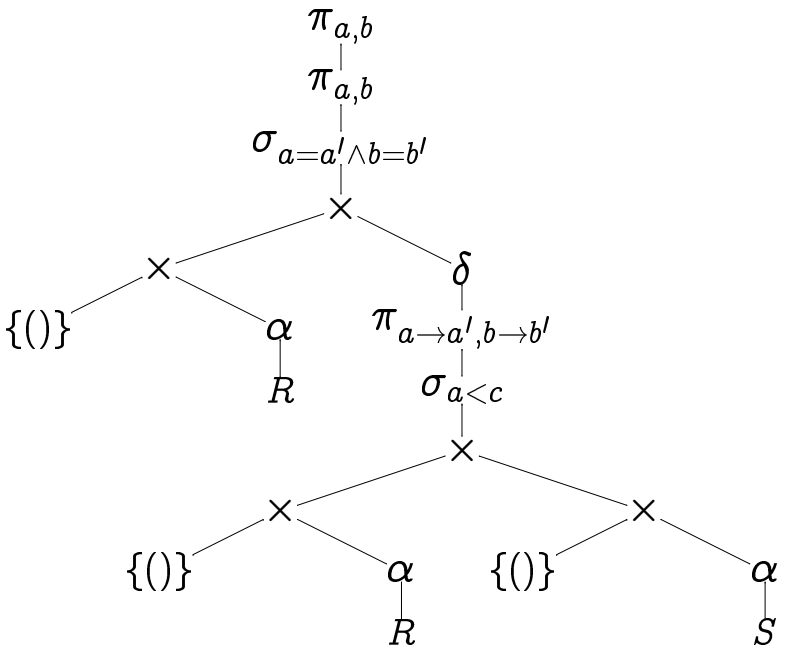
$$\underset{(18)}{=} \quad \mathbb{P}\left(\right.$$



$$\left. , \ [\,(a : x.a, b : x.b)\ |\ ]^{bag} \right)$$

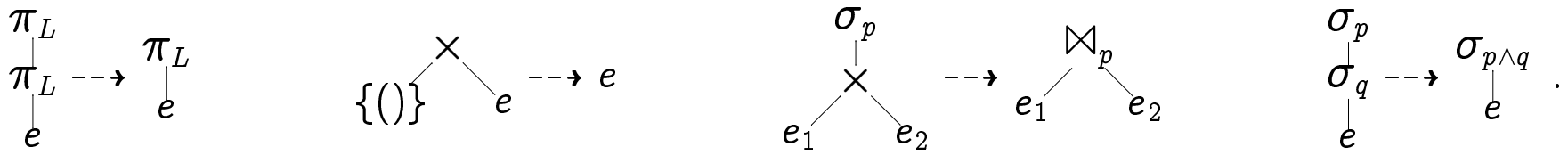
$\underset{(14)}{=}$ $\mathbb{P}$ 

, $[\,(a : x.a, b : x.b) \mid\,]^{bag}$

$\underset{(16)}{=}$ 

5.2.4 Optimizer Desperately Needed!

- As you can see, the plans generated by Plan provide obvious opportunities for **simplification and optimization** in general.
  - To facilitate modular design in a query processor, this optimization phase is carried out in a separate unit, the **query optimizer**.
  - The optimizer comes with a **rule base** of optimization opportunities, e.g.



  - The optimizer selects the next rule to be **fired** using a **heuristic** that characterizes useful or "promising" rules.

> ✎ **The list goes on . . .**
> Which other **algebraic optimization rules** can you think of?

5.3 Recommended Reading

[1] H. Garcia-Molina, J. D. Ullman, and J. Widom, *Database System Implementation*, Prentice Hall, New Jersey, 2000. ISBN 0-13-040264-8.

Contents