A Simple Plan Generator

Outline:

1. preliminaries
2. reorderability
3. conflict detection
4. enumeration

Preliminaries (strict predicates)

Definition

A predicate is null rejecting for a set of attributes A if it evaluates to FALSE *or* UNKNOWN *on every tuple in which all attributes in A are* NULL*.*

Synonyms for null rejecting are used: null intolerant, strong, and strict.

Preliminaries (initial operator tree)

We assume that we have an initial operator tree, e.g., by a canonical translation of a SQL query.

Preliminaries (accessors)

For a set of attributes A, $\texttt{REL}(A)$ denotes the set of tables to which these attributes belong. We abbreviate $\texttt{REL}(\mathcal{F}(e))$ by $\mathcal{F}_{\mathsf{T}}(e)$. Let $\circ$ be an operator in the initial operator tree. We denote by left($\circ$) (right($\circ$)) its left (right) child. $\texttt{STO}(\circ)$ denotes the operators contained in the operator subtree rooted at $\circ$. $\texttt{REL}(\circ)$ denotes the set of tables contained in the subtree rooted at $\circ$.

Preliminaries (SES)

Then, for each operator we define its *syntactic eligibility sets* as its set of tables referenced by its predicate.
If $p \equiv R.a + S.b = S.c + T.d$, then $\mathcal{F}(p) = \{R.a, S.b, S.c, T.d\}$ and $\mathrm{SES}(\circ_p) = \{R, S, T\}$.

Preliminaries (degenerate predicates)

Definition

Let p be a predicate associated with a binary operator $\circ$ and $\mathcal{F}_{\mathsf{T}}(p)$ the tables referenced by p. Then, p is called *degenerate* if $\mathtt{REL}(\text{left}(\circ)) \cap \mathcal{F}_{\mathsf{T}}(p) = \emptyset \ \vee \ \mathtt{REL}(\text{right}(\circ)) \cap \mathcal{F}_{\mathsf{T}}(p) = \emptyset$ holds.

Here, we exclude degenerate predicates.

Preliminaries (hypergraph)

Definition

A *hypergraph* is a pair $H = (V, E)$ such that

1. V is a non-empty set of nodes, and
2. E is a set of hyperedges, where a *hyperedge* is an unordered pair (u, v) of non-empty subsets of V ($u \subset V$ and $v \subset V$) with the additional condition that $u \cap v = \emptyset$.

We call any non-empty subset of V a *hypernode*.

Preliminaries (Necessity of Hypergraphs)

possible join predicate: $R.a + S.b = S.c + T.d$
even without non-binary join predicates: conflict detectors introduce hypergraphs

Preliminaries (Neighborhood)

$$\min(S) = \{s | s \in S, \forall s' \in S\ s \neq s' \implies s \prec s'\}$$

Let S be a current set, which we want to expand by adding further relations. Consider a hyperedge (u, v) with $u \subseteq S$. Then, we will add $\min(v)$ to the neighborhood of S. We thus define

$$\overline{\min}(S) = S \setminus \min(S)$$

Note: we have to make sure that the missing elements of v, i.e. $v \setminus \min(v)$, are also contained in any set emitted.

Preliminaries (Neighborhood)

We define the set of non-subsumed hyperedges as the minimal subset $E{\downarrow}$ of E such that for all $(u, v) \in E$ there exists a hyperedge $(u', v') \in E{\downarrow}$ with $u' \subseteq u$ and $v' \subseteq v$.

$$E{\downarrow}'(S, X) = \{v | (u, v) \in E, u \subseteq S, v \cap S = \emptyset, v \cap X = \emptyset\}$$

Define $E{\downarrow}(S, X)$ to be the minimal set of hypernodes such that for all $v \in E{\downarrow}'(S, X)$ there exists a hypernode v' in $E{\downarrow}(S, X)$ such that $v' \subseteq v$.
Neighborhood:

$$\mathcal{N}(S, X) = \bigcup_{v \in E{\downarrow}(S,X)} \min(v) \tag{1}$$

where X is the set of forbidden nodes.

Preliminaries (csg-cmp-pair)

Definition

Let $H = (V, E)$ be a hypergraph and S_1, S_2 two non-empty subsets of V with $S_1 \cap S_2 = \emptyset$. Then, the pair (S_1, S_2) is called a *csg-cmp-pair* if the following conditions hold:

1. S_1 and S_2 induce a connected subgraph of H, and
2. there exists a hyperedge $(u, v) \in E$ such that $u \subseteq S_1$ and $v \subseteq S_2$.

Reorderability (properties)

- commutativity (comm)
- associativity (assoc)
- l/r-asscom

Reorderability (comm)

$\circ$	
$\times$	+
$\bowtie$	+
$\ltimes$	-
$\triangleright$	-
⟕	-
⟗	+
$\bowtie^{\Gamma}$	-

Reorderability (assoc)

assoc:

$$(e_1 \circ^a_{12} e_2) \circ^b_{23} e_3 \equiv e_1 \circ^a_{12} (e_2 \circ^b_{23} e_3) \tag{2}$$

Reorderability (assoc)

$\circ^a$ \ $\circ^b$	$\times$	$\bowtie$	$\ltimes$	$\triangleright$	⟕	⟗	$\bowtie^{\Gamma}$
$\times$	+	+	+	+	+	-	+
$\bowtie$	+	+	+	+	+	-	+
$\ltimes$	-	-	-	-	-	-	-
$\triangleright$	-	-	-	-	-	-	-
⟕	-	-	-	-	+[1]	-	-
⟗	-	-	-	-	+[1]	+[2]	-
$\bowtie^{\Gamma}$	-	-	-	-	-	-	-

(1) if p_{23} rejects nulls on $\mathcal{A}(e_2)$ (Eqv. 2)

(2) if p_{12} and p_{23} reject nulls on $\mathcal{A}(e_2)$ (Eqv. 2)

Reorderability (l/r-asscom)

Consider the following truth about the semijoin:

$$(e_1 \ltimes_{12} e_2) \ltimes_{13} e_3 \equiv (e_1 \ltimes_{13} e_3) \ltimes_{12} e_2.$$

This is not expressible with associativity nor commutativity (in fact the semijoin is neither).

Reorderability (l/r-asscom)

We define the *left asscom property* (l-asscom for short) as follows:

$$(e_1 \circ^a_{12} e_2) \circ^b_{13} e_3 \equiv (e_1 \circ^b_{13} e_3) \circ^a_{12} e_2. \quad (3)$$

We denote by l-asscom($\circ^a, \circ^b$) the fact that Eqv. 3 holds for $\circ^a$ and $\circ^b$.
Analogously, we can define a *right asscom property* (r-asscom):

$$e_1 \circ^a_{13} (e_2 \circ^b_{23} e_3) \equiv e_2 \circ^b_{23} (e_1 \circ^a_{13} e_3). \quad (4)$$

First, note that l-asscom and r-asscom are symmetric properties, i.e.,

$$\begin{aligned} \text{l-asscom}(\circ^a, \circ^b) &\leftrightarrow \text{l-asscom}(\circ^b, \circ^a), \\ \text{r-asscom}(\circ^a, \circ^b) &\leftrightarrow \text{r-asscom}(\circ^b, \circ^a). \end{aligned}$$

Reorderability (l/r-asscom)

$\circ$	$\times$	$\bowtie$	$\ltimes$	$\triangleright$	⟕	⟗	$\bowtie^{\Gamma}$
$\times$	+/+	+/+	+/-	+/-	+/-	-/-	+/-
$\bowtie$	+/+	+/+	+/-	+/-	+/-	-/-	+/-
$\ltimes$	+/-	+/-	+/-	+/-	+/-	-/-	+/-
$\triangleright$	+/-	+/-	+/-	+/-	+/-	-/-	+/-
⟕	+/-	+/-	+/-	+/-	+/-	+[1] /-	+/-
⟗	-/-	-/-	-/-	-/-	+[2] /-	+[3] /+[4]	-/-
$\bowtie^{\Gamma}$	+/-	+/-	+/-	+/-	+/-	-/-	+/-

1 if p_{12} rejects nulls on $\mathcal{A}(e_1)$ (Eqv. 3)

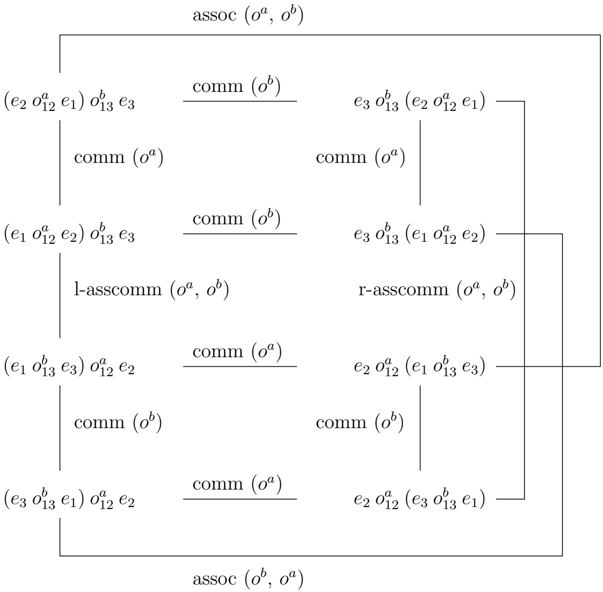
2 if p_{13} rejects nulls on $\mathcal{A}(e_3)$ (Eqv. 3)

3 if p_{12} and p_{13} rejects nulls on $\mathcal{A}(e_1)$ (Eqv. 3)

4 if p_{13} and p_{23} reject nulls on $\mathcal{A}(e_3)$ (Eqv. 4)

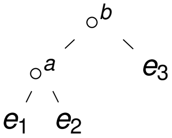
Conflict Detector CD-A: $\mathtt{SES}$

$$
\begin{aligned}
\mathtt{SES}(R) &= \{R\} \\
\mathtt{SES}(T) &= \{T\} \\
\mathtt{SES}(\circ_p) &= \bigcup_{R \in \mathcal{F}_\mathsf{T}(p)} \mathtt{SES}(R) \cap \mathtt{REL}(\circ_p) \\
\mathtt{SES}(\Join\!\!\urcorner_{p;a_1:e_1,\ldots,a_n:e_n}) &= \bigcup_{R \in \mathcal{F}_\mathsf{T}(p) \cup \mathcal{F}_\mathsf{T}(e_i)} \mathtt{SES}(R) \cap \mathtt{REL}(gj)
\end{aligned}
$$

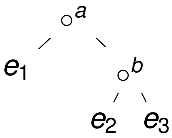

assoc $(o^a,\ o^b)$
$(e_2\ o^a_{12}\ e_1)\ o^b_{13}\ e_3$
comm (o^b)
$e_3\ o^b_{13}\ (e_2\ o^a_{12}\ e_1)$
comm (o^a)
comm (o^a)
$(e_1\ o^a_{12}\ e_2)\ o^b_{13}\ e_3$
comm (o^b)
$e_3\ o^b_{13}\ (e_1\ o^a_{12}\ e_2)$
l-asscomm $(o^a,\ o^b)$
r-asscomm $(o^a,\ o^b)$
$(e_1\ o^b_{13}\ e_3)\ o^a_{12}\ e_2$
comm (o^a)
$e_2\ o^a_{12}\ (e_1\ o^b_{13}\ e_3)$
comm (o^b)
comm (o^b)
$(e_3\ o^b_{13}\ e_1)\ o^a_{12}\ e_2$
comm (o^a)
$e_2\ o^a_{12}\ (e_3\ o^b_{13}\ e_1)$
assoc $(o^b,\ o^a)$

Conflict Detector CD-A: TES: left conflict

initially: $\mathtt{TES}(\circ_p) := \mathtt{SES}(\circ_p)$



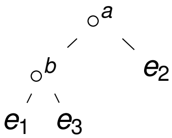
$\xrightarrow{\text{assoc}}$



$\neg\text{assoc}(\circ^a, \circ^b)$

$\mathtt{TES}(\circ^b) \cup= \mathtt{REL}(e_1)$

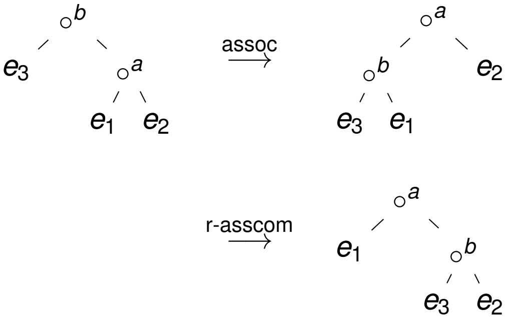
$\xrightarrow{\text{l-asscom}}$



$\neg\text{l-asscom}(\circ^a, \circ^b)$

$\mathtt{TES}(\circ^b) \cup= \mathtt{REL}(e_2)$

Conflict Detector CD-A: TES: right conflict



$\neg \text{assoc}(\circ^b, \circ^a)$

$\texttt{TES}(\circ^b) \cup= \texttt{REL}(e_2)$

$\neg \text{r-asscom}(\circ^a, \circ^b)$

$\texttt{TES}(\circ^b) \cup= \texttt{REL}(e_1)$

Conflict Detector CD-A: Remarks

- correct
- not complete

Conflict Detector CD-A: applicability test

$$\text{applicable}(\circ, S_1, S_2) := \text{tesl}(\circ) \subseteq S_1 \wedge \text{tesr}(\circ) \subseteq S_2.$$

where

$$\begin{aligned}\text{tesl}(\circ) &:= \texttt{TES}(\circ) \cap \texttt{REL}(\text{left}(\circ))\\ \text{tesr}(\circ) &:= \texttt{TES}(\circ) \cap \texttt{REL}(\text{right}(\circ))\end{aligned}$$

Query Hypergraph Construction

The nodes V are the relations.
For every operator $\circ$, we construct a hyperedge (l, r) such that
$r = \mathtt{TES}(\circ) \cap \mathtt{REL}(\text{right}(\circ)) = \mathtt{R\text{-}TES}(\circ)$ and
$l = \mathtt{TES}(\circ) \setminus r = \mathtt{L\text{-}TES}(\circ)$.

DP-PLANGEN

▷ **Input:** a set of relations $R = \{R_0, \ldots, R_{n-1}\}$
a set of operators O with associated predicates
a query hypergraph H

▷ **Output:** an optimal bushy operator tree

```
for all R_i ∈ R
    DPTable[R_i] ← R_i ▷ initial access paths
for all csg-cmp-pairs (S_1, S_2) of H
    for all ∘_p ∈ O
        if APPLICABLE(S_1, S_2, ∘_p)
            BUILDPLANS(S_1, S_2, ∘_p)
            if ∘_p is commutative
                BUILDPLANS(S_2, S_1, ∘_p)
return DPTable[R]
```

$\textsc{BuildPlans}(S_1, S_2, \circ_p)$

$\mathit{OptimalCost} \leftarrow \infty$

$S \leftarrow S_1 \cup S_2$

$T_1 \leftarrow \mathit{DPTable}[S_1]$

$T_2 \leftarrow \mathit{DPTable}[S_2]$

if $\mathit{DPTable}[S] \neq \mathit{NULL}$

  $\mathit{OptimalCost} \leftarrow \textsc{Cost}(\mathit{DPTable}[S])$

if $\textsc{Cost}(T_1 \circ_p T_2) < \mathit{OptimalCost}$

  $\mathit{OptimalCost} \leftarrow \textsc{Cost}(T_1 \circ_p T_2)$

  $\mathit{DPTable}[S] \leftarrow (T_1 \circ_p T_2)$

Csg-Cmp-Enumeration: Overview

1. The algorithm constructs ccps by enumerating connected subgraphs from an increasing part of the query graph;
2. both the primary connected subgraphs and its connected complement are created by recursive graph traversals;
3. during traversal, some nodes are *forbidden* to avoid creating duplicates. More precisely, when a function performs a recursive call it forbids all nodes it will investigate itself;
4. connected subgraphs are increased by following edges to neighboring nodes. For this purpose hyperedges are interpreted as $n : 1$ edges, leading from n of one side to one (specific) canonical node of the other side (cmp. Eq. 1).

The last point is like selecting a representative.

Csg-Cmp-Enumeration: Complications

- “starting side” of an edge may contain multiple nodes
- neighborhood calculation more complex, no longer simply bottom-up
- choosing representative: loss of connectivity possible

Last point: use `DpTable` lookup as connectivity test

Csg-Cmp-Enumeration: Routines

1. top-level: `BuEnumCcpHyp`
2. `EnumerateCsgRec`
3. `EmitCsg`
4. `EnumerateCmpRec`

Csg-Cmp-Enumeration: `BuEnumCcpHyp`

`BuEnumCcpHyp`()
for each $v \in V$ // initialize DpTable
    DpTable[$\{v\}$] = plan for v
for each $v \in V$ **descending** according to $\prec$
    `EmitCsg`($\{v\}$) // process singleton sets
    `EnumerateCsgRec`($\{v\}$, **:B**$_v$) // expand singleton sets
return DpTable[V]

where $B_v = \{w | w \prec v\} \cup \{v\}$.

Csg-Cmp-Enumeration: EnumerateCsgRec

`EnumerateCsgRec`(S_1, X)
for each $N \subseteq \mathcal{N}(S_1, X)$: $N \neq \emptyset$
    if DpTable[$S_1 \cup N$]$\neq \emptyset$
        `EmitCsg`$(S_1 \cup N)$
for each $N \subseteq \mathcal{N}(S_1, X)$: $N \neq \emptyset$
    `EnumerateCsgRec`$(S_1 \cup N, X \cup \mathcal{N}(S_1, X))$

Csg-Cmp-Enumeration: `EmitCsg`

`EmitCsg`(S_1)
$X = S_1 \cup \mathbf{B}_{min(S_1)}$
$N = \mathcal{N}(S_1, X)$
for each $v \in N$ **descending** according to $\prec$
    $S_2 = \{v\}$
    if $\exists (u, v) \in E : u \subseteq S_1 \wedge v \subseteq S_2$
        `EmitCsgCmp`(S_1, S_2)
    `EnumerateCmpRec`$(S_1, S_2, X \cup B_v(N))$

where $B_v(W) = \{w | w \in W, w \leq v\}$ is defined as in `DPccp`.

Csg-Cmp-Enumeration: `EnumerateCmpRec`

```
EnumerateCmpRec(S_1, S_2, X)
for each N ⊆ N(S_2, X): N ≠ ∅
    if DpTable[S_2 ∪ N] ≠ ∅ ∧
            ∃(u, v) ∈ E : u ⊆ S_1 ∧ v ⊆ S_2 ∪ N
        EmitCsgCmp(S_1, S_2 ∪ N)
X = X ∪ N(S_2, X)
for each N ⊆ N(S_2, X): N ≠ ∅
    EnumerateCmpRec(S_1, S_2 ∪ N, X)
```

Csg-Cmp-Enumeration: `EmitCsgCmp`

The procedure `EmitCsgCmp(`S_1`,`S_2`)` is called for every S_1 and S_2 such that (S_1, S_2) forms a csg-cmp-pair.
important. Since it is called for either (S_1, S_2) or (S_2, S_1), somewhere the symmetric pairs have to be considered.

Csg-Cmp-Enumeration: Neighborhood Calculation

Let $G = (V, E)$ be a hypergraph not containing any subsumed edges.
For some set S, for which we want to calculate the neighborhood, define the set of reachable hypernodes as

$$W(S, X) := \{w | (u, w) \in E, u \subseteq S, w \cap (S \cup X) = \emptyset\},$$

where X contains the forbidden nodes. Then, any set of nodes N such that for every hypernode in $W(S, X)$ exactly one element is contained in N can serve as the neighborhood.

```
CalcNeighborhood(S, X)
N := ∅
if isConnected(S)
    N = simpleNeighborhood(S) \ X
else
    foreach s ∈ S
        N ∪= simpleNeighborhood(s)
F = (S ∪ X ∪ N) // forbidden since in X or already handled
foreach (u, v) ∈ E
    if u ⊆ S
        if v ∩ F = ∅
            N += min(v)
            F ∪= N
    if v ⊆ S
        if u ∩ F = ∅
            N += min(u)
            F ∪= N
```