Preliminaries

We define the event log expressions using relational algebra inspired by Maier [3].

Definition 1 (column, table, schema, population, tuple, database). *Let $\mathcal{C} \subseteq \Sigma^*$ be a set of* columns. *$R \subseteq \mathcal{C}, R \neq \varnothing$ is a* table*, where $Schema(R) = R$, and $Pop(R) \in \wp(R \to \Omega)$ is the* population *of R. Ω is a set of values, and $\wp$ is the power set. A* tuple *t in R is defined as $t \in Pop(R)$, i.e. $t\colon R \to \Omega$. $Dom\colon C \to \wp(\Omega)$ is a function that returns the domain or type of a column, i.e. its set of possible values. For each $t \in Pop(R)$ and $c \in R$, $t[c] \in Dom(c)$. A* database *is a set of tables with their population.*

Table 1a shows an example of a table R with $Schema(R) = \{c, a, time, r\}$ and $Pop(R) = \{\ldots, \{c \mapsto c_1, a \mapsto a_0, time \mapsto 0, r \mapsto r_0\}, \ldots\}$.

c	a	$time$	r
.	.	.	.
c_1	a_0	0	r_0
c_1	a_1	1	r_1
c_1	a_2	2	r_1
c_2	a_2	1	r_3
c_2	a_1	2	r_3
c_2	a_3	3	r_3
.	.	.	.

(a) A relation

a	$time$	r
.	.	.
a_1	1	r_1
a_1	2	r_3
.	.	.

(b) Applying operators

$\downarrow c$	$\downarrow a$	$\downarrow time$	$\downarrow r$	$\uparrow c$	$\uparrow a$	$\uparrow time$	$\uparrow r$
.	.	.	.	.	.	.	.
c_1	a_0	0	r_0	c_1	a_1	1	r_1
c_1	a_1	1	r_1	c_1	a_2	2	r_1
c_2	a_2	1	r_3	c_2	a_1	2	r_3
c_2	a_1	2	r_3	c_2	a_3	3	r_3
.	.	.	.	.	.	.	.

(c) Directly follows application

Table 1: Relational algebra examples.

Relational algebra expressions make use of a number of well-known operators.

Definition 2 (relational algebra operators). *Let R and S be two tables.*

- *If $Schema(R) = Schema(S)$, then the* union *$R \cup S$ is a table with $Schema(R \cup S) = Schema(R) = Schema(S)$ and $Pop(R \cup S) = Pop(R) \cup Pop(S)$, consisting of all tuples that belong to either R or S or both.*
- *If $Schema(R) = Schema(S)$, the* set difference *$R \backslash S$ is a table with $Schema(R \backslash S) = Schema(R) = Schema(S)$ and $Pop(R \backslash S) = Pop(R) \backslash Pop(S)$ consisting of tuples that are in R but not in S.*
- *The* join *$R \bowtie S$ is a table with $Schema(R \bowtie S) = Schema(R) \cup Schema(S)$ and $Pop(R \bowtie S) = \{t | t[Schema(R)] \in Pop(R) \wedge t[Schema(S)] \in Pop(S)\}$ consisting of tuples that combine tuples of R with tuples of S. If $R \cap S = \varnothing$ then $R \bowtie S$ is the* Cartesian product *of R and S.*
- *The* selection *$\sigma_F(R)$ of R using a selection formula F is a table with $Schema(\sigma_F(R)) = Schema(R)$ and $Pop(\sigma_F(R)) = \{t \in Pop(R) | t \models F\}$ that only includes tuples that satisfy the selection formula.*
- *The* projection *$\pi_{q_1\colon p_1, \cdots, q_n\colon p_n}(R) = \pi_M(R)$, where $q_1, \cdots, q_n \in \mathcal{C}$ are different column names and $p_1, \cdots, p_n \in Schema(R)$, is a table with Schema(*

$\pi_M(R)) = \{q_1, \cdots, q_n\}$ *and* $Pop(\pi_M(R)) = \{s | t \in Pop(R), |s| = n, \forall_{1 \leq i \leq n} : s[q_i] = t[p_i]\}$ *that only includes a subset of columns of R. Alternatively, the* projection $\pi_S(R)$*, where* $S = \{s_1, \cdots, s_n\} \subseteq Schema(R)$ *are columns of R can be used.*

– *The* extended projection *operator* $\pi_{q_1 : F_1, \cdots, q_n : F_n}(R)$*, where* $q_1, \cdots, q_n \in \mathcal{C}$ *are different column names and* $F_1, \cdots, F_n$ *are formulas over* $Schema(R)$*, enables applying functions to tuples [2], such that* $Schema(\pi_{q_1 : F_1, \cdots, q_n : F_n}(R)) = \{q_1, \cdots, q_n\}$ *and* $Pop(\pi_{q_1 : F_1, \cdots, q_n : F_n}(R)) = \{s | t \in Pop(R), |s| = n, \forall_{1 \leq i \leq n} : s[q_i] = F_i(t)\}$.

Table 1b shows an example in which the operators $\pi_{\{a,time,r\}}(\sigma_{a=a_1}(R))$ are applied to R from Table 1a

In event logs, the directly follows relation plays an important role. For that reason, we introduce the directly follows relation as it is previously defined [1] into the relational algebra.

Definition 3. *Let R be a tabular event log with* $Schema(R) = \{c, time, p_1, p_2, \ldots, p_n\}$*, where c is the column with case identifiers, and time the column with completion timestamps. Applying the directly follows operator, denoted* $>_{c,time} R$ *to the event log returns the relation of events that follow each other in some case:* $\{\{\downarrow c \mapsto t[c], \downarrow time \mapsto t[time], \downarrow p_1 \mapsto t[p_1], \ldots, \uparrow c \mapsto u[c], \uparrow time \mapsto u[time], \uparrow p_1 \mapsto u[p_1], \ldots\}, t \in R, u \in R, t[c] = u[c], t[time] < u[time], \neg\exists v \in R \colon t[c] = u[c] \wedge t[time] < v[time] \wedge v[time] < u[time]\}$.

Table 1c illustrates the use of the directly follows operator. The table presents an event log in which there are two cases labelled c_1 and c_2 in which events happen at time *time*. Each event represents that an activity a was performed by a resource r. The result is the table that contains all pairs of events that directly follow each other in some case. For example, the event that activity a_0 is performed in case c_1 is directly followed by the event that activity a_1 is performed in case c_1. In the remainder of this paper, we assume that a directly precedes operator $<_{c,time} R$ exists that is defined analogously to the directly follows operator.

Proof of Equivalence

We show by rewriting that the proposed algorithm returns a result that is equivalent to the result created by the definition of the generalised group-by operator:

$$_{GC(t_1,t_2)}\mathcal{G}_{p_1 \mapsto F_1, p_2 \mapsto F_2, \ldots p_n \mapsto F_n} R$$

The algorithm to compute the generalised group-by using vanilla relational algebra operators is as follows.

result $= \emptyset$
for $t \in Pop(R)$:

$R' = \sigma_{GC(t,\cdot)} R$
if $R' = \{t\}$:
$\{q_1, \ldots, q_m\} = \{p_1, \ldots, p_n\} \cup \mathit{Schema}(R)$
$\{s_1, \ldots, s_k\} = \{p_1, \ldots, p_n\} - \mathit{Schema}(R)$
$\mathit{result} = \mathit{result} \cup \pi_{q_1,\ldots,q_m} R' \times \{s_1 \mapsto \perp, \ldots, s_k \mapsto \perp\}$
else:
$\mathit{result} = \mathit{result} \cup \{\{p_i \mapsto F_i(R') \mid i \in \{1, \ldots, n\}\}\}$
return *result*

This can be rewritten by distributing the for statement over the if statement as:

$\mathit{result} = \emptyset$
for $t \in \mathit{Pop}(R)$:
$R' = \sigma_{GC(t,\cdot)} R$
if $R' = \{t\}$:
$\{q_1, \ldots, q_m\} = \{p_1, \ldots, p_n\} \cup \mathit{Schema}(R)$
$\{s_1, \ldots, s_k\} = \{p_1, \ldots, p_n\} - \mathit{Schema}(R)$
$\mathit{result} = \mathit{result} \cup \pi_{q_1,\ldots,q_m} R' \times \{s_1 \mapsto \perp, \ldots, s_k \mapsto \perp\}$
for $t \in \mathit{Pop}(R)$:
$R' = \sigma_{GC(t,\cdot)} R$
if $R' \neq \{t\}$:
$\mathit{result} = \mathit{result} \cup \{\{p_i \mapsto F_i(R') \mid i \in \{1, \ldots, n\}\}\}$
return *result*

This can be rewritten by rewriting the relational algebraic expression as:

$\mathit{result} = \emptyset$
for $t \in \mathit{Pop}(R)$:
$R' = \sigma_{GC(t,\cdot)} R$
if $R' = \{t\}$:
$\mathit{result} = \mathit{result} \cup \{p_i \mapsto (t[p_i] \text{ if } p_i \in \mathit{Schema}(R) \text{ else } \perp) \mid$
$i \in \{1, 2, \ldots, n\}\}$
for $t \in \mathit{Pop}(R)$:
$R' = \sigma_{GC(t,\cdot)} R$
if $R' \neq \{t\}$:
$\mathit{result} = \mathit{result} \cup \{\{p_i \mapsto F_i(R') \mid i \in \{1, \ldots, n\}\}\}$
return *result*

This can be rewritten by turning the for and if statements into set comprehensions as:

$$\mathit{result} = \{p_i \mapsto (t[p_i] \text{ if } p_i \in \mathit{Schema}(R) \text{ else } \perp) \mid i \in \{1, 2, \ldots, n\} \wedge t \in \mathit{Pop}(R) \wedge \sigma_{GC(t,\cdot)} R = \{t\}\}$$
$$\mathit{result} = \mathit{result} \cup \big\{\{p_i \mapsto F_i(R') \mid i \in \{1, \ldots, n\}\} \mid t \in \mathit{Pop}(R) \wedge R' = \sigma_{GC(t,\cdot)} R \wedge R' \neq \{t\}\big\}$$

return *result*

This can be rewritten by rewriting the selection statements as:

$$
\begin{aligned}
\mathit{result} = \{ & p_i \mapsto (t[p_i] \text{ if } p_i \in \mathit{Schema}(R) \text{ else } \perp) \mid \\
& i \in \{1, 2, \ldots, n\} \wedge t \in \mathit{Pop}(R) \wedge \\
& \nexists t' \in \mathit{Pop}(R) \colon t' \neq t \wedge GC(t, t')\}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{result} = \mathit{result} \cup \big\{ & \{p_i \mapsto F_i(R') \mid i \in \{1, \ldots, n\}\} \; \big| \\
& t \in \mathit{Pop}(R) \wedge R' \subseteq R \wedge R' \neq \{t\} \wedge \forall v \in R' \colon GC(t, v) \wedge \\
& \nexists z \in R \backslash R' \colon \exists u' \in R' \colon GC(u', z) \big\}
\end{aligned}
$$

return *result*

It is easy to see that this is equivalent to the definition of the population of the generalised group-by statement and that it has the schema $\{p_1, p_2, \ldots, p_n\}$.

References

1. Dijkman, R., et al.: Enabling efficient process mining on large data sets: Realizing an in-database process mining operator. Distrib. Parallel Databases **38**(1), 227–253 (2020)
2. Grefen, P., de By, R.: A multi-set extended relational algebra: A formal approach to a practical issue. In: Int. Conf. Data Eng. pp. 80–88. IEEE, USA (1994)
3. Maier, D.: The theory of relational databases. Computer Science Press (1983)