1 Joins on $Query(v)$ as a monoid

In order to justify some of our optimisations, we need to know certain properties of join denotation.

Firstly, we want to define establish closure and associatiavity of the *join* function defined in the denotational semantics.

We want to define these for queryable subsets of a view $v \in View_\Sigma$

$$Query(v) = \{s \mid \exists P, A, B. s = [\![P]\!](v) \wedge \Sigma \vdash P \colon A, B\} \tag{1}$$

1.1 Lemma: Join is closed on $Query(v)$

Proof: For $s, t \in Query(v)$ there exists P, Q, A, B, C, D such that $\Sigma \vdash P \colon A, B$ and $\Sigma \vdash Q \colon C, D$ hold and $s = [\![P]\!](v) \wedge t = [\![Q]\!](v)$

Case $B \neq C$ Then the join is empty, since no $b \in B = c \in C$, so

$$join([\![P]\!](v), [\![Q]\!](v)) = \emptyset = [\![Distinct(Id_T)]\!](v) \tag{2}$$

For some type $T \in \Sigma$ (We know Σ contains more than one type, due to the typing of P and Q

Case $B = C$ Then the join is not empty. By the correspondence of denotational and operational semantics, we have

$$join([\![P]\!](v), [\![Q]\!](v)) = [\![Chain(P, Q)]\!](v) \tag{3}$$

with the following typing

$$\Sigma \vdash Chain(P, Q) \colon A, D \tag{4}$$

1.2 Lemma: Join is associative on $Query(v)$

Proof: For $r, s, t \in Query(v)$ there exist queries P, Q, R and object types A, B, C, D, E, F such that $\Sigma \vdash P \colon A, B$, $\Sigma \vdash Q \colon C, D$, $\Sigma \vdash R \colon E, F$ and hold and $r = [\![P]\!](v) \wedge s = [\![Q]\!](v) \wedge t = [\![R]\!](v)$ We want to prove that $join(r, join(s, t)) = join(join(r, s), t)$

Case $B \neq C$ Then

$$\begin{aligned} join(r, join(s, t)) &= join(r, u) \text{ For some } u\text{, either } u = \emptyset \text{ or } u \text{ has a left type of } C \\ &= \emptyset \text{ hence equals the empty set} \\ &= join(\emptyset, t) \\ &= join(join(r, s), t) \end{aligned} \tag{5}$$

Case $D \neq E$ Then similarly

$$\begin{aligned} join(join(r, s), t) &= join(u, t) \text{ For some } u\text{, either } u = \emptyset \text{ or } u \text{ has a right type of } D \\ &= \emptyset \text{ hence equals the empty set} \\ &= join(r, \emptyset) \\ &= join(r, join(s, t)) \end{aligned} \tag{6}$$

Case $B = C$ and $D = E$ Then

$$
\begin{aligned}
(a,f) \in join(r, join(s,t)) &\Leftrightarrow (a,f) \in join([\![P]\!](v), join([\![Q]\!](v), [\![R]\!](v))) \\
&\Leftrightarrow a, f \lhd_{A,F,v} Chain(P, Chain(Q,R)) \text{ by deno-oper correspondence} \\
&\Leftrightarrow \exists b, d.(a,b) \lhd_{A,B,v} P \wedge (b,d) \lhd_{B,D,v} Q \wedge (d,f) \lhd_{D,F,v} Q \\
&\Leftrightarrow (a,f) \lhd_{A,F,v} Chain(Chain(P,Q), R) \\
&\Leftrightarrow (a,f) \in [\![Chain(Chain(P,Q),R)]\!](v) \\
&\Leftrightarrow (a,f) \in join(join([\![P]\!](v), [\![Q]\!](v)), [\![R]\!](v)) \\
&\Leftrightarrow (a,f) \in join(join(r,s),t)
\end{aligned}
\tag{7}
$$

Hence $Query(v)$ is a monoid with $join$. However, it doesn't particularly stick to a nice typescheme.

1.3 Joins on $Query_A(v)$ as a monoid

If we now take the more useful subset $Query_A(v)$ of $Query(v)$

$$Query_A(v) = \{s \subseteq A \times A \mid \exists P.\Sigma \vdash P\colon A, A \wedge s = [\![P]\!](v)\} \tag{8}$$

Trivially, from the above, $join$ over $Query_A(v)$ also forms a monoid.

This enables several optimisations.

Firstly, writing p for $[\![P]\!](v)$ and with the binary representation of $n = \sum_i b_i * 2^i$

$$
\begin{aligned}
[\![Exactly(n,P)]\!](v) &= p^n \text{ In } Query_A(v) \\
&= \prod_i p^{b_i * 2^i}
\end{aligned}
\tag{9}
$$

Todo: join distributes over and and or

1.4 Joins distribute over Or

It is useful to know how $join$ interacts with other queries. Specifically we can do some re-writing if it is the case that join distributes over Or

$$[\![Chain(P, Or(Q,R))]\!](v) = [\![Or(Chain(P,Q), Chain(P,R))]\!](v) \tag{10}$$

Firstly, we have by inversion of the type rules for $Chain$ and Or that:

$$
\begin{aligned}
\Sigma \vdash Chain(P, Or(Q,R))\colon A, C &\Leftrightarrow \Sigma \vdash P\colon A, B \wedge \Sigma \vdash Q, R\colon B, C \\
&\Leftrightarrow \Sigma \vdash Chain(P,Q)\colon A, C \wedge \Sigma \vdash Chain(P,R)\colon A, C \\
&\Leftrightarrow \Sigma \vdash Or(Chain(P,Q), Chain(P,R))\colon A, C
\end{aligned}
\tag{11}
$$

$$
\begin{aligned}
[\![Chain(P, Or(Q,R))]\!](v) &= \{(a,c) \mid \exists b.(a,b) \in [\![P]\!](v) \wedge (b,c) \in ([\![Q]\!](v) \cup [\![R]\!](v)\}) \\
&= \{(a,c) \mid \exists b.((a,b) \in [\![P]\!](v) \wedge (b,c) \in [\![Q]\!](v)) \vee ((a,b) \in [\![P]\!](v) \wedge (b,c) \in [\![R]\!](v))\} \\
&= [\![Chain(P,Q)]\!](v) \cup [\![Chain(P,R)]\!](v) \\
&= [\![Or(Chain(P,Q), Chain(P,R))]\!](v)
\end{aligned}
\tag{12}
$$

1.5 $Upto(n, P)$ expressed as $Exactly(n, P')$

Thanks to the previous section, we can now rewrite the denotation of *upto*

$$\begin{aligned}
[\![Upto(n, P)]\!](v) &= (\lambda pairs.join([\![P]\!](v), pairs) \cup pairs)^n [\![Id_A]\!](v) \\
&= (\lambda pairs.join([\![P]\!](v) \cup [\![Id_A]\!](v), pairs)^n [\![Id_A]\!](v) \text{ Since } Id_A \text{ is the identity of join} \\
&= [\![Exactly(n, Or(P, Id_A))]\!](v)
\end{aligned} \tag{13}$$

This means that we can evaluate *Upto* queries as *Exactly* queries, and apply the binary-representation driven construction above to evaluate queries using fewer unique joins.

1.6 Joins do not distribute over And

Consider the schema containing object types A, B, C and relations $R_1\colon A, B$, $R_2\colon B, C$, and $R_3\colon B, C$ and a view v in this schema, with relations $R_1 \mapsto \{(a, b1), (a, b2)\}$, $R_2 \mapsto \{(b1, c)\}$, and $R_3 \mapsto \{(b2, c)\}$.

Clearly

$$(a, c) \in [\![And(Chain(R_1, R_2), Chain(R_1, R_3))]\!](v) \tag{14}$$

But

$$[\![And(R_2, R_3)]\!](v) = \emptyset \tag{15}$$

So

$$[\![Chain(And(R_2, R_3))]\!](v) = \emptyset \tag{16}$$