IVC

July 28, 2011

1 Definitions

In this draft we want to explain the notion of domain. The domain of a variable is the set of values that a variable can take. The domain of all the variables in a query expression can easily be created recursively if some rules will be respected. We will use through out the entire paper the notation of $\mathsf{dom}_{\vec{x}}(q)$ for the domain of a set of variables, where $\vec{x}$ is a vector representing the variables present in the expression q. Taking into account the expressions from the AGCA (Aggregate Calculus[**?**]) the definition of an expression will be:

$$q ::\text{-}\ q \cdot q | q + q | v \leftarrow q | v_1 \theta v_2 | R(\vec{y}) | c | v$$

We will start with the definition of $\mathsf{dom}_{\vec{x}}(R(\vec{y}))$:

$$\mathsf{dom}_{\vec{x}}(R(\vec{y})) = \left\{ \vec{c} \,|\, (\prod_{\forall i : x_i \in \vec{y}} (x_i \leftarrow c_i) \cdot R(\vec{y})) \neq 0 \right\}$$

where $\vec{x} =< x_1, x_2, x_3, \cdots, x_i, \cdots >$, $\vec{c} =< c_1, c_2, c_3, \cdots, c_i, \cdots >$, will be a vector of constants and the schema of $\vec{x}$ will match the schema of $\vec{c}$.

For the comparison operator($v_1 \theta v_2$), where v_1 and v_2 are variables, we can compute the domain as follows:

$$\mathsf{dom}_{\vec{x}}(v_1 \theta v_2) = \left\{ \vec{c} \,|\, \forall i : (x_i = v_1 \vee x_i = v_2) \Rightarrow ((\prod_i (x_i \leftarrow c_i))(v_1 \theta v_2)) \neq 0 \right\}$$

In fact using implication operator in the above definition allows us to extend the $\vec{x}$ to whatever vector we want. It is not necessary that $\vec{x}$ has the same schema as the given expression. Thus, we can evaluate $\mathsf{dom}_{\vec{x}}$ for a broader range of $\vec{x}$ and it is not delimited by the schema of the expression. For the join operator we can write:

$$\mathsf{dom}_{\vec{x}}(q_1 \cdot q_2) = \{\vec{c}|\vec{c} \in \mathsf{dom}_{\vec{x}}(q_1) \wedge \vec{c} \in \mathsf{dom}_{\vec{x}}(q_2)\}$$

while for the union operator the domain definition is very similar:

$$\mathsf{dom}_{\vec{x}}(q_1 + q_2) = \{\vec{c}|\vec{c} \in \mathsf{dom}_{\vec{x}}(q_1) \vee \vec{c} \in \mathsf{dom}_{\vec{x}}(q_2)\}$$

The dom of constants is infinite. The dom of an expression, that is represented by a variable (q ::- var), will be bound to other expressions in which that variable will appear. . Finally, we can give a formalism for expressing the domain of a variable that will participate in an assignment operation:

$$\mathsf{dom}_{\vec{x}}(v \leftarrow q_1) = \left\{\vec{c}|\vec{c} \in \mathsf{dom}_{\vec{x}}(q_1) \wedge \forall i : \big((x_i = v) \Rightarrow (q_1 \cdot \prod_{j:x_j \neq v} (x_j = c_j)) = c_i\big)\right\}$$

Having the definitions for the domains, we will try to give some insight regarding to the notion of arity. We will start with an example relation:

$$q = R(a, b) \cdot S(b, c)$$

the schema for q will have three variables (a, b, c) The arity of a tuple will be the number of occurrences in the relation, in other words the order of multiplicity of that tuple. The arity of a tuple will increase or decrease if insertions or respectively deletions will be made to a relation. For example:

R	a	b	arity
	<1	2>	1
	<1	3>	2
	<3	4>	1

Table 1: Relation R

We need to maintain the maps for certain values as long as the arity of a tuple is greater then 0. If the arity of the tuple drops to 0 then the tuple will not be taken into consideration and therefore it can be eliminated from the domains of the maps.

We need to store the arities inside each map. We need a way to compute the arity of an AGCA expression. Since we substitute the subexpressions with maps, we can easily consider the maps without input variables as some relations. Also a map with input variables can be seen as a relation with a group-by clause. Input variables of a map bind some variables.

S	b	c	arity
	<1	1>	1
	<2	2>	2
	<3	5>	2

Table 2: Relation S

$R \cdot S$	a	b	c	arity
	<1	2	2>	2
	<1	3	5>	4
	<3	4	$*>$	0
	$<*$	1	1>	0

Table 3: Relation $R \cdot S$

2 Computing the arities of the AGCA expressions

We define the function arity which will be used to compute the arity of a tuple in a certain relation. The function will be defined on the relation and the tuple for which the multiplicity order is desired to be computed.

$$arity(Relation \text{ q}, Tuple \text{ t}) = \text{multiplicity order of tuple } t \text{ in the relation } q$$

$$arity(q,t) = \prod_t (q)$$

where $\prod_t (q)$ means the projection of relation q for the tuple t. This function can be used for the computation of the arity of the expressions from the AGCA: q ::- $q \cdot q$ | $q + q$ | $q \theta t$ | $t \leftarrow q$ | *constant* | *variable*, however constants and variables can be eliminated from the computation because relations are of interest.

$$arity(q1 \cdot q2, t) = \sum_{\{t1\} \bowtie \{t2\} = t} arity(q1, t1) * arity(q2, t2)$$

$$arity(q1 + q2, t) = arity(q1, t) + arity(q2, t), \text{ where Sch(q1)=Sch(q2)}$$

$$arity(t1\ \theta\ t2) = \begin{cases} 0, & \text{if } t1 \theta t2 \text{ is false} \\ 1, & \text{if } t1 \theta t2 \text{ is false} \end{cases}$$

$$arity(t \leftarrow q) = 1$$