

## 11 ALP Proof of Correctness

### 11.0.1 Equality: Algorithm 2 is correct.

**PROOF.** We prove that  $l \leq \text{col}_1 = \text{col}_2 = \dots = \text{col}_n \leq r$  is extracted correctly. Let us refer to the attributes together as equality set  $C_n$ . Assuming the contradiction, let Algorithm 2 extract some other lower and upper s-value bounds  $l_k$  and  $r_k$  for a partition of size  $k$  (i.e.  $C_k = \{\text{col}_1, \text{col}_2, \dots, \text{col}_k\}$ ), for some  $k < n$ ,  $l_k \neq l$ ,  $r_k \neq r$ . Therefore, a mutation where  $\text{col}_n$  has one value and the attributes in  $C_k$  have a different value, gets a FIT-result from  $Q_{\mathcal{H}}$ . Either of  $l$  and  $r$  can serve as this value, when  $C_k$  is mutated within interval  $[l_k, r_k]$ . This is possible only when  $\text{col}_i \neq \text{col}_n$  satisfies  $Q_{\mathcal{H}}$  for any  $1 \leq i \leq k$ . This violates the fact the  $Q_{\mathcal{H}}$  has equality within  $C_n$ . So, the algorithm does not extract a wrong inequality.

**11.0.2 Inequality: Algorithm 3 is correct.** Combining Lemmas 10-13, we prove the correctness.

**LEMMA 10.** *If  $Q_{\mathcal{H}}$  has a hidden algebraic predicate chain of the form  $l \rightarrow \text{col}_1 \rightarrow \text{col}_2 \rightarrow \dots \rightarrow \text{col}_n \rightarrow r$ , where  $i_{\min} \leq l \leq r \leq i_{\max}$ , and  $\rightarrow$  is either  $\leq$  or  $<$ , and no other predicate involving  $\text{col}_i$   $\forall i, 1 < i < n$  exists in  $Q_{\mathcal{H}}$ , Algorithm 3 extracts it correctly.*

**PROOF.** Apart from  $\text{col}_i \rightarrow \text{col}_{i+1}$ , the only predicates in  $Q_{\mathcal{H}}$  involving  $\text{col}_i$  is  $l + \delta_2 * \Delta \rightarrow \text{col}_i$ , and involving  $\text{col}_{i+1}$  is  $\text{col}_{i+1} \rightarrow r - \delta_3 * \Delta$ , where  $\delta_2$  is the number of preceding  $<$  operators of  $\text{col}_i$  in -chain, and  $\delta_3$  is the number of succeeding  $<$  operators of  $\text{col}_{i+1}$  in -chain. Therefore,  $S_E^{\text{LB}}_{\text{col}_i} = l + \delta_2 * \Delta$ ,  $S_E^{\text{UB}}_{\text{col}_{i+1}} = r - \delta_3 * \Delta$ . Assume the contradiction,  $\text{col}_i \rightarrow \text{col}_{i+1} \notin E$  when Algorithm 3 terminated. Let  $D^1.\text{col}_i = v_i$ ,  $D^1.\text{col}_{i+1} = v_{i+1}$  for some  $l + \delta_2 * \Delta \leq v_i \leq v_{i+1} \leq r - \delta_3 * \Delta$ , which is bound to be true for the given  $Q_{\mathcal{H}}$ . Since no other predicate involves  $\text{col}_i$  and  $\text{col}_{i+1}$ ,  $S_E^{\text{UB}}_{\text{col}_i} = v_{i+1}$ ,  $S_E^{\text{LB}}_{\text{col}_{i+1}} = v_i$ , which ensures that Algorithm 3(pre-processing) includes  $\text{col}_i \rightarrow \text{col}_{i+1}$  in  $E$ . It can only be removed from  $E$  if mutation of  $\text{col}_i$  does not impact the LB of  $\text{col}_{i+1}$ . Thus, when  $\text{col}_i$  is mutated with  $v_{i+1}$ , LB of  $\text{col}_{i+1}$  still remains  $v_i$ . The same holds when the mutation value is  $l + \delta_2 * \Delta$ . So,  $l + \delta_2 * \Delta \leq v_{i+1} \leq v_i$ , which is possible only if  $l + \delta_2 * \Delta = v_i = v_{i+1}$ . It obtains the same LB and UB for  $\text{col}_i$ . The UB must be static to match the static LB of  $l + \delta_2 * \Delta$ . It contradicts  $\text{col}_i$  not having any more predicates.

**LEMMA 11.** *If  $Q_{\mathcal{H}}$  has a hidden algebraic predicate chain of the form  $l \rightarrow \text{col}_1 \rightarrow \text{col}_2 \rightarrow \dots \rightarrow \text{col}_n \rightarrow r$ , where  $i_{\min} \leq l \leq r \leq i_{\max}$ , and  $\rightarrow$  is either  $\leq$  or  $<$ , and for any  $\text{col}_i \rightarrow \text{col}_{i+1}$  pair in the predicate chain, at least one of the static bounds  $\text{LB}_{i+1} \rightarrow \text{col}_{i+1}$  and  $\text{col}_i \rightarrow \text{UB}_i$  exists, Algorithm 3 extracts the predicate chain correctly.*

**PROOF.**  $UB_i < LB_{i+1}$  implies  $\text{col}_i \rightarrow \text{col}_{i+1}$  is a redundant predicate. Therefore, we prove that Algorithm 3 extracts  $\text{col}_i \rightarrow \text{col}_{i+1}$  when  $LB_{i+1} \rightarrow UB_i$ . Now, it is given that  $S_E^{\text{LB}}_{\text{col}_{i+1}} = LB_{i+1}$ , and  $S_E^{\text{UB}}_{\text{col}_i} = UB_i$ . Let  $D^1.\text{col}_i = v_i$ ,  $D^1.\text{col}_{i+1} = v_{i+1}$ .

(1) Let  $v_i, v_{i+1} \in [LB_{i+1}, UB_i]$ . We also have  $v_i \rightarrow v_{i+1}$  due to -chain. The given static LB of  $\text{col}_{i+1}$  can only happen in the presence of -chain if  $v_i \rightarrow LB_{i+1}$ . Due to our initial assumption,  $v_i = LB_{i+1}$ . Therefore, when  $D^1.\text{col}_i$  is mutated with a higher value  $UB_i$ ,  $S_E^{\text{LB}}_{\text{col}_{i+1}}$  becomes  $UB_i$  (increases), i.e. gets impacted. Therefore, the algorithm extracts  $\text{col}_i \rightarrow \text{col}_{i+1}$ .

(2) Let  $v_i \in [i_{\min}, LB_{i+1}]$ ,  $v_{i+1} \in (UB_i, i_{\max}]$ . So, when  $\text{col}_i$  gets mutated with a higher value  $UB_i$ , following the construction in the earlier case,  $S_E^{\text{LB}}_{\text{col}_{i+1}}$  gets impacted, extracting  $\text{col}_i \rightarrow \text{col}_{i+1}$ .

**LEMMA 12.** *Algorithm 3 does not extract a predicate  $\text{col}_x \rightarrow \text{col}_y$  that is absent in  $Q_{\mathcal{H}}$ . (The proof is skipped due to its triviality.)*

**LEMMA 13.** *If  $Q_{\mathcal{H}}$  has a hidden algebraic predicate chain of the form  $l \rightarrow \text{col}_1 \rightarrow \text{col}_2 \rightarrow \dots \rightarrow \text{col}_n \rightarrow r$ , where  $i_{\min} \leq l \leq r \leq i_{\max}$ , and  $\rightarrow$  is either  $\leq$  or  $<$ , Algorithm 3 extracts all the arithmetic predicates ( $\text{col}_i, \leq, UB_i$ ) and ( $\text{col}_i, \geq, LB_i$ ) correctly,  $\forall i, 1 \leq i \leq n$ .*

**PROOF.** Assuming a contradiction, the algorithm obtained  $\text{col}_i \leq \text{col}_i$ , and  $lb_i \leq \text{col}_i$ , such that  $lb_i \neq v_i$ .  $v_i \leq \text{col}_i$  is given to be in  $Q_{\mathcal{H}}$ . Since  $Q_{\mathcal{H}}$  produces UNFIT-result on  $lb_i < v_i$ , our assumption leads to  $v_i < lb_i$ . This implies  $S_E^{\text{LB}}_{\text{col}_i} = lb_i$  when  $\text{col}_{i-1}$  has mutated value of its own lower bound. So,  $\text{col}_{i-1}$  has a minimum possible value satisfying  $Q_{\mathcal{H}}$  is  $lb_i$ , i.e.  $lb_i \leq \text{col}_{i-1}$ . This is a contradiction, given that  $v_i \leq \text{col}_i$ . Similarly, the  $\geq$  can also be proved.

## 12 Performance Enhancements

Extraction modules that operate solely on the minimized database  $D^1$  – e.g. algebraic predicates – have negligible cost due to its minuscule size. However, modules that need to work with the original database  $D_I$  – e.g. NEP extraction – could incur significant overheads if  $D_I$  is large. Therefore, PRISM employs the following techniques to reduce performance bottlenecks in these modules.

### 12.1 Correlated Sampling [1]

Correlated sampling is a technique that makes use of the schema join graph in the sampling process. This results in a higher probability of the sampled data satisfying the join predicates. It is used before the database minimization step to obtain a smaller  $D_I$  that produces a FIT-result, thereby reducing the iterations in the minimization. This is also useful in outer join queries, where random sampling is susceptible to producing tuples with mismatched keys.

### 12.2 View-based Database Minimization

We employ a minimization technique based on *virtual views*, which does not require copying the records of a table during the binary halving process. The views are created on the base table by utilizing system-generated tuple identifiers, which give the physical location of a row in the table – for instance, in PostgreSQL, this identifier is called *ctid* and consists of a block number and a record number within that block. The *ctid* of the first record of a table is  $(0, 1)$ . The number of records present in a block,  $n_b$ , is table-width dependent but computable from the schema. Based on  $n_b$ , we can estimate the *ctid* of the middle row of the table. For example, the following queries create a view containing roughly the upper half of table  $T$ :

```
Alter Table T Rename to Tdummy;
Create View T as
Select * From Tdummy
Where ctid between '(0,1)' and '(|Tdummy|/2n_b,1)';
```

If a FIT-result is obtained, the view creation continues recursively with the upper half; if not, it shifts to a virtual view on the lower

half. This reduction continues until a  $D^1$  is achieved. The key improvement over the explicit halving approach is the avoidance of the time and space costs for materializing the intermediate tables.

### 12.3 Hash-Based Result Comparators

This component aims to efficiently verify the equality of the results of  $Q_H$  and  $Q_E$  over  $D_I$ . The direct but slow technique is to explicitly compute, using the EXCEPT command, the difference between the results in both directions – i.e.  $\mathcal{R}_H$  EXCEPT  $\mathcal{R}_E$  and  $\mathcal{R}_E$  EXCEPT  $\mathcal{R}_H$ , and verify that both are zero. This especially becomes a performance bottleneck in NEP extraction. Therefore, XPOSE employs the following hash-based result comparators instead. While PostgreSQL has several options for hash functions, we use the HashText hash function since it works across datatypes.

**12.3.1 Global Hash Method:** This method is used if the query has ORDER BY ( $\vec{O}$ ) or GROUP BY attributes. It sorts  $\mathcal{R}_H$  and  $\mathcal{R}_E$  on remaining projection attributes wrt  $\vec{O}$ , then computes hashes on each result table. If they are equal, the result tables are the same.

**12.3.2 Rolling Hash Method:** This method is used if the query has no physical ordering. Here, we calculate a hash value for a relation by applying a hash function to each tuple in the relation and then aggregating the results. In the PostgreSQL database, to get the rolling hash of the tuples present in the result set  $\mathcal{R}_H$ , we use:

```
Select SUM(rh_hashes.hashtext)
```

```
From (Select hashtext( $\mathcal{R}_H$ ::TEXT) From  $\mathcal{R}_H$ ) as rh_hashes;
```

The same evaluation is done for  $\mathcal{R}_E$ . Comparing the two hashes confirms or denies the unordered set equality of  $\mathcal{R}_H$  and  $\mathcal{R}_E$ .

## References

- [1] F. Yu, W.-C. Hou, C. Luo, D. Che, and M. Zhu, “CS2: A New Database Synopsis for Query Estimation,” in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2013)*, New York, NY, USA, June 22–27, 2013, pp. 469–480. ACM, 2013. doi: 10.1145/2463676.2463701.