Formalization of Multiway-Join Algorithms

Thibault Dardinier

December 14, 2021

Abstract

Worst-case optimal multiway-join algorithms are recent seminal achievement of the database community. These algorithms compute the natural join of multiple relational databases and improve in the worst case over traditional query plan optimizations of nested binary joins. In 2014, Ngo, Ré, and Rudra [1] gave a unified presentation of different multi-way join algorithms. We formalized and proved correct their "Generic Join" algorithm and extended it to support negative joins.

Contents

1 The algorithm

theory *Generic_Join*
imports *MFOTL_Monitor.Table*
begin

type_synonym *'a atable* = *nat set* × *'a table*
type_synonym *'a query* = *'a atable set*
type_synonym *vertices* = *nat set*

1.1 Generic algorithm

locale *getIJ* =
fixes *getIJ* :: *'a query* ⇒ *'a query* ⇒ *vertices* ⇒ *vertices* × *vertices*
assumes *coreProperties*: *card V* ≥ *2* ⟹ *getIJ Q_pos Q_neg V* = (*I*, *J*) ⟹
card I ≥ *1* ∧ *card J* ≥ *1* ∧ *V* = *I* ∪ *J* ∧ *I* ∩ *J* = {}
begin

lemma *getIJProperties*:
assumes *card V* ≥ *2*
assumes (*I*, *J*) = *getIJ Q_pos Q_neg V*
shows *card I* ≥ *1* **and** *card J* ≥ *1* **and** *card I* < *card V* **and** *card J* < *card V*

```
    and V = I ∪ J and I ∩ J = {}
⟨proof⟩

fun projectTable :: vertices ⇒ 'a atable ⇒ 'a atable where
  projectTable V (s, t) = (s ∩ V, Set.image (restrict V) t)

fun filterQuery :: vertices ⇒ 'a query ⇒ 'a query where
  filterQuery V Q = Set.filter (λ(s, _). ¬ Set.is_empty (s ∩ V)) Q

fun filterQueryNeg :: vertices ⇒ 'a query ⇒ 'a query where
  filterQueryNeg V Q = Set.filter (λ(A, _). A ⊆ V) Q

fun projectQuery :: vertices ⇒ 'a query ⇒ 'a query where
  projectQuery V s = Set.image (projectTable V) s

fun isSameIntersection :: 'a tuple ⇒ nat set ⇒ 'a tuple ⇒ bool where
  isSameIntersection t1 s t2 = (∀ i∈s. t1!i = t2!i)

fun semiJoin :: 'a atable ⇒ (nat set × 'a tuple) ⇒ 'a atable where
  semiJoin (s, tab) (st, tup) = (s, Set.filter (isSameIntersection tup (s ∩ st)) tab)

fun newQuery :: vertices ⇒ 'a query ⇒ (nat set × 'a tuple) ⇒ 'a query where
  newQuery V Q (st, t) = Set.image (λtab. projectTable V (semiJoin tab (st, t))) Q

fun merge_option :: 'a option × 'a option ⇒ 'a option where
  merge_option (None, None) = None
| merge_option (Some x, None) = Some x
| merge_option (None, Some x) = Some x
| merge_option (Some a, Some b) = Some a


fun merge :: 'a tuple ⇒ 'a tuple ⇒ 'a tuple where
  merge t1 t2 = map merge_option (zip t1 t2)

function (sequential) genericJoin :: vertices ⇒ 'a query ⇒ 'a query ⇒ 'a table where
  genericJoin V Q_pos Q_neg =
    (if card V ≤ 1 then
      (⋂(_, x) ∈ Q_pos. x) − (⋃(_, x) ∈ Q_neg. x)
    else
      let (I, J) = getIJ Q_pos Q_neg V in
      let Q_I_pos = projectQuery I (filterQuery I Q_pos) in
      let Q_I_neg = filterQueryNeg I Q_neg in
      let R_I = genericJoin I Q_I_pos Q_I_neg in
      let Q_J_neg = Q_neg − Q_I_neg in
      let Q_J_pos = filterQuery J Q_pos in
       let X = {(t, genericJoin J (newQuery J Q_J_pos (I, t)) (newQuery J Q_J_neg (I, t))) | t . t ∈
R_I} in
      (⋃(t, x) ∈ X. {merge xx t | xx . xx ∈ x}))
⟨proof⟩
termination
  ⟨proof⟩

fun wrapperGenericJoin :: 'a query ⇒ 'a query ⇒ 'a table where
  wrapperGenericJoin Q_pos Q_neg =
    (if ((∃(A, X)∈Q_pos. Set.is_empty X) ∨ (∃(A, X)∈Q_neg. Set.is_empty A ∧ ¬ Set.is_empty X))
then
      {}
    else
```

```
    let Q = Set.filter (λ(A, _). ¬ Set.is_empty A) Q_pos in
    if Set.is_empty Q then
      (⋂(A, X)∈Q_pos. X) −  (⋃(A, X)∈Q_neg. X)
    else
      let V = (⋃(A, X)∈Q. A) in
      let Qn = Set.filter (λ(A, _). A ⊆ V ∧ card A ≥ 1) Q_neg in
      genericJoin V Q Qn)
```

end

1.2 An instantation

```
fun score :: 'a query ⇒ nat ⇒ nat where
  score Q i = (let relevant = Set.image (λ(_, x). card x) (Set.filter (λ(sign, _). i ∈ sign) Q) in
    let l = sorted_list_of_set relevant in
    foldl (+) 0 l
)

fun arg_max_list :: ('a ⇒ nat) ⇒ 'a list ⇒ 'a where
  arg_max_list f l = (let m = Max (set (map f l)) in arg_min_list (λx. m − f x) l)

lemma arg_max_list_element:
  assumes length l ≥ 1 shows arg_max_list f l ∈ set l
  ⟨proof⟩

fun max_getIJ :: 'a query ⇒ 'a query ⇒ vertices ⇒ vertices × vertices where
  max_getIJ Q_pos Q_neg V = (
  let l = sorted_list_of_set V in
  if Set.is_empty Q_neg then
    let x = arg_max_list (score Q_pos) l in
    ({x}, V − {x})
  else
    let x = arg_max_list (score Q_neg) l in
    (V − {x}, {x}))

lemma max_getIJ_coreProperties:
  assumes card V ≥ 2
  assumes (I, J) = max_getIJ Q_pos Q_neg V
  shows card I ≥ 1 ∧ card J ≥ 1 ∧ V = I ∪ J ∧ I ∩ J = {}
⟨proof⟩

global_interpretation New_max: getIJ max_getIJ
  defines New_max_getIJ_genericJoin = New_max.genericJoin
  and New_max_getIJ_wrapperGenericJoin = New_max.wrapperGenericJoin
  ⟨proof⟩
```

end

2 Correctness

2.1 Well-formed queries

```
theory Generic_Join_Correctness
  imports Generic_Join
begin

definition wf_set :: nat ⇒ vertices ⇒ bool where
  wf_set n V ⟷ (∀ x∈V. x < n)
```

definition *wf_atable* :: *nat* ⇒ *'a atable* ⇒ *bool* **where**
wf_atable n X ⟷ *table n* (*fst X*) (*snd X*) ∧ *finite* (*fst X*)

definition *wf_query* :: *nat* ⇒ *vertices* ⇒ *'a query* ⇒ *'a query* ⇒ *bool* **where**
wf_query n V Q_pos Q_neg ⟷ (∀ *X*∈(*Q_pos* ∪ *Q_neg*). *wf_atable n X*) ∧ (*wf_set n V*) ∧ (*card Q_pos* ≥ *1*)

definition *included* :: *vertices* ⇒ *'a query* ⇒ *bool* **where**
included V Q ⟷ (∀ (*S*, *X*)∈*Q*. *S* ⊆ *V*)

definition *covering* :: *vertices* ⇒ *'a query* ⇒ *bool* **where**
covering V Q ⟷ *V* ⊆ (⋃ (*S*, *X*)∈*Q*. *S*)

definition *non_empty_query* :: *'a query* ⇒ *bool* **where**
non_empty_query Q = (∀ *X*∈*Q*. *card* (*fst X*) ≥ *1*)

definition *rwf_query* :: *nat* ⇒ *vertices* ⇒ *'a query* ⇒ *'a query* ⇒ *bool* **where**
rwf_query n V Qp Qn ⟷ *wf_query n V Qp Qn* ∧ *covering V Qp* ∧ *included V Qp* ∧ *included V Qn*
∧ *non_empty_query Qp* ∧ *non_empty_query Qn*

lemma *wf_tuple_empty*: *wf_tuple n* {} *v* ⟷ *v* = *replicate n None*
⟨*proof*⟩

lemma *table_empty*: *table n* {} *X* ⟷ (*X* = *empty_table* ∨ *X* = *unit_table n*)
⟨*proof*⟩

context *getIJ* **begin**

lemma *isSame_equi_dev*:
assumes *wf_set n V*
assumes *wf_tuple n A t1*
assumes *wf_tuple n B t2*
assumes *s* ⊆ *A*
assumes *s* ⊆ *B*
assumes *A* ⊆ *V*
assumes *B* ⊆ *V*
shows *isSameIntersection t1 s t2* = (*restrict s t1* = *restrict s t2*)
⟨*proof*⟩

lemma *wf_getIJ*:
assumes *card V* ≥ *2*
assumes *wf_set n V*
assumes (*I*, *J*) = *getIJ Q_pos Q_neg V*
shows *wf_set n I* **and** *wf_set n J*
⟨*proof*⟩

lemma *wf_projectTable*:
assumes *wf_atable n X*
shows *wf_atable n* (*projectTable I X*) ∧ (*fst* (*projectTable I X*) = (*fst X* ∩ *I*))
⟨*proof*⟩

lemma *set_filterQuery*:
assumes *QQ* = *filterQuery I Q*
assumes *non_empty_query Q*
shows ∀ *X*∈*Q*. (*card* (*fst X* ∩ *I*) ≥ *1* ⟷ *X* ∈ *QQ*)
⟨*proof*⟩

lemma *wf_filterQuery*:
assumes $I \subseteq V$
assumes *card* $I \geq 1$
assumes *rwf_query n V Qp Qn*
assumes $QQp =$ *filterQuery I Qp*
assumes $QQn =$ *filterQueryNeg I Qn*
shows *wf_query n I QQp QQn non_empty_query QQp covering I QQp*
⟨*proof*⟩

lemma *wf_set_subset*:
assumes $I \subseteq V$
assumes *card* $I \geq 1$
assumes *wf_set n V*
shows *wf_set n I*
⟨*proof*⟩

lemma *wf_projectQuery*:
assumes *card* $I \geq 1$
assumes *wf_query n I Q Qn*
assumes *non_empty_query Q*
assumes *covering I Q*
assumes $\forall X \in Q.$ *card* (*fst* $X \cap I$) ≥ 1
assumes $QQ =$ *projectQuery I Q*
assumes *included I Qn*
assumes *non_empty_query Qn*
shows *rwf_query n I QQ Qn*
⟨*proof*⟩

lemma *wf_firstRecursiveCall*:
assumes *rwf_query n V Qp Qn*
assumes *card* $V \geq 2$
assumes $(I, J) =$ *getIJ Qp Qn V*
assumes $Q\_I\_pos =$ *projectQuery I* (*filterQuery I Qp*)
assumes $Q\_I\_neg =$ *filterQueryNeg I Qn*
shows *rwf_query n I Q_I_pos Q_I_neg*
⟨*proof*⟩

lemma *wf_atable_subset*:
assumes *table n V X*
assumes $Y \subseteq X$
shows *table n V Y*
⟨*proof*⟩

lemma *same_set_semiJoin*:
fst (*semiJoin x other*) = *fst x*
⟨*proof*⟩

lemma *wf_semiJoin*:
assumes *card* $J \geq 1$
assumes *wf_query n J Q Qn*
assumes *non_empty_query Q*
assumes *covering J Q*
assumes $\forall X \in Q.$ *card* (*fst* $X \cap J$) ≥ 1
assumes $QQ =$ (*Set.image* (λ*tab.* *semiJoin tab* (*st*, *t*)) *Q*)
shows *wf_query n J QQ Qn non_empty_query QQ covering J QQ*
⟨*proof*⟩

lemma *newQuery_equiv_def*:

newQuery V Q (st, t) = projectQuery V (Set.image (λtab. semiJoin tab (st, t)) Q)
⟨*proof*⟩

lemma *included_project*:
included V (projectQuery V Q)
⟨*proof*⟩

lemma *non_empty_newQuery*:
assumes *Q1 = filterQuery J Q0*
assumes *Q2 = newQuery J Q1 (I, t)*
assumes ∀ *X*∈*Q0. wf_atable n X*
shows *non_empty_query Q2*
⟨*proof*⟩

lemma *wf_newQuery*:
assumes *card J ≥ 1*
assumes *wf_query n J Q Qn0*
assumes *non_empty_query Q*
assumes *covering J Q*
assumes ∀ *X*∈*Q. card (fst X ∩ J) ≥ 1*
assumes *QQ = newQuery J Q t*
assumes *QQn = newQuery J Qn t*
assumes *non_empty_query Qn*
assumes *Qn = filterQuery J Qn0*
shows *rwf_query n J QQ QQn*
⟨*proof*⟩

lemma *subset_Q_neg*:
assumes *rwf_query n V Q Qn*
assumes *QQn ⊆ Qn*
shows *rwf_query n V Q QQn*
⟨*proof*⟩

lemma *wf_secondRecursiveCalls*:
assumes *card V ≥ 2*
assumes *rwf_query n V Q Qn*
assumes *(I, J) = getIJ Q Qn V*
assumes *Qns ⊆ Qn*
assumes *Q_J_neg = filterQuery J Qns*
assumes *Q_J_pos = filterQuery J Q*
shows *rwf_query n J (newQuery J Q_J_pos t) (newQuery J Q_J_neg t)*
⟨*proof*⟩

lemma *simple_merge_option*:
merge_option (a, b) = None ⟷ (a = None ∧ b = None)
⟨*proof*⟩

lemma *wf_merge*:
assumes *wf_tuple n I t1*
assumes *wf_tuple n J t2*
assumes *V = I ∪ J*
assumes *t = merge t1 t2*
shows *wf_tuple n V t*
⟨*proof*⟩

lemma *wf_inter*:
assumes *rwf_query n {i} Q Qn*
assumes *(sa, a) ∈ Q*

assumes $(sb,\ b) \in Q$
shows *table* n $\{i\}$ $(a \cap b)$
⟨*proof*⟩

lemma *table_subset*:
assumes *table* n V T
assumes $S \subseteq T$
shows *table* n V S
⟨*proof*⟩

lemma *wf_base_case*:
assumes *card* $V = 1$
assumes *rwf_query* n V Q Qn
assumes $R =$ *genericJoin* V Q Qn
shows *table* n V R
⟨*proof*⟩

lemma *filter_Q_J_neg_same*:
assumes *card* $V \geq 2$
assumes $(I,\ J) =$ *getIJ* Q Qn V
assumes *Q_I_neg* = *filterQueryNeg* I Qn
assumes *rwf_query* n V Q Qn
shows *filterQuery* J $(Qn - Q\_I\_neg) = Qn - Q\_I\_neg$ (**is** *?A* = *?B*)
⟨*proof*⟩

lemma *vars_genericJoin*:
assumes *card* $V \geq 2$
assumes $(I,\ J) =$ *getIJ* Q Qn V
assumes *Q_I_pos* = *projectQuery* I (*filterQuery* I Q)
assumes *Q_I_neg* = *filterQueryNeg* I Qn
assumes *R_I* = *genericJoin* I *Q_I_pos* *Q_I_neg*
assumes *Q_J_neg* = *filterQuery* J $(Qn - Q\_I\_neg)$
assumes *Q_J_pos* = *filterQuery* J Q
assumes $X = \{(t,$ *genericJoin* J (*newQuery* J *Q_J_pos* $(I,\ t))$ (*newQuery* J *Q_J_neg* $(I,\ t))) \mid t\ .\ t \in R\_I\}$
assumes $R = (\bigcup (t,\ x) \in X.\ \{merge\ xx\ t \mid xx\ .\ xx \in x\})$
assumes *rwf_query* n V Q Qn
shows $R =$ *genericJoin* V Q Qn
⟨*proof*⟩

lemma *base_genericJoin*:
assumes *card* $V \leq 1$
shows *genericJoin* V Q $Qn = (\bigcap (\_,\ x) \in Q.\ x) - (\bigcup (\_,\ x) \in Qn.\ x)$
⟨*proof*⟩

lemma *wf_genericJoin*:
$[\![$*rwf_query* n V Q Qn; *card* $V \geq 1]\!] \Longrightarrow$ *table* n V (*genericJoin* V Q Qn)
⟨*proof*⟩

2.2 Correctness

lemma *base_correctness*:
assumes *card* $V = 1$
assumes *rwf_query* n V Q Qn
assumes $R =$ *genericJoin* V Q Qn
shows $z \in$ *genericJoin* V Q $Qn \longleftrightarrow$ *wf_tuple* n V $z \wedge (\forall (A,\ X) \in Q.$ *restrict* A $z \in X) \wedge (\forall (A,\ X) \in Qn.$ *restrict* A $z \notin X)$
⟨*proof*⟩

lemma *simple_list_index_equality*:
assumes *length a* = *n*
assumes *length b* = *n*
assumes ∀ *i* < *n*. *a*!*i* = *b*!*i*
shows *a* = *b*
⟨*proof*⟩

lemma *simple_restrict_none*:
assumes *i* < *length X*
assumes *i* ∉ *A*
shows (*restrict A X*)!*i* = *None*
⟨*proof*⟩

lemma *simple_restrict_some*:
assumes *i* < *length X*
assumes *i* ∈ *A*
shows (*restrict A X*)!*i* = *X*!*i*
⟨*proof*⟩

lemma *merge_restrict*:
assumes *A* ∩ *J* = {}
assumes *A* ⊆ *I*
assumes *length xx* = *n*
assumes *length t* = *n*
assumes *restrict J xx* = *xx*
shows *restrict A* (*merge xx t*) = *restrict A t*
⟨*proof*⟩

lemma *restrict_idle_include*:
assumes *wf_tuple n A v*
assumes *A* ⊆ *I*
shows *restrict I v* = *v*
⟨*proof*⟩

lemma *merge_index*:
assumes *I* ∩ *J* = {}
assumes *wf_tuple n I tI*
assumes *wf_tuple n J tJ*
assumes *t* = *merge tI tJ*
assumes *i* < *n*
shows (*i* ∈ *I* ∧ *t*!*i* = *tI*!*i*) ∨ (*i* ∈ *J* ∧ *t*!*i* = *tJ*!*i*) ∨ (*i* ∉ *I* ∧ *i* ∉ *J* ∧ *t*!*i* = *None*)
⟨*proof*⟩

lemma *restrict_index_in*:
assumes *i* < *length X*
assumes *i* ∈ *I*
shows (*restrict I X*)!*i* = *X*!*i*
⟨*proof*⟩

lemma *restrict_index_out*:
assumes *i* < *length X*
assumes *i* ∉ *I*
shows (*restrict I X*)!*i* = *None*
⟨*proof*⟩

lemma *merge_length*:
assumes *length a* = *n*

assumes *length b = n*
shows *length (merge a b) = n*
⟨*proof*⟩

lemma *real_restrict_merge*:
assumes *I ∩ J = {}*
assumes *wf_tuple n I tI*
assumes *wf_tuple n J tJ*
shows *restrict I (merge tI tJ) = restrict I tI ∧ restrict J (merge tI tJ) = restrict J tJ*
⟨*proof*⟩

lemma *simple_set_image_id*:
assumes *∀ x∈X. f x = x*
shows *Set.image f X = X*
⟨*proof*⟩

lemma *nested_include_restrict*:
assumes *restrict I z = t*
assumes *A ⊆ I*
shows *restrict A z = restrict A t*
⟨*proof*⟩

lemma *restrict_nested*:
restrict A (restrict B x) = restrict (A ∩ B) x (**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *newQuery_equi_dev*:
newQuery V Q (I, t) = Set.image (projectTable V) (Set.image (λtab. semiJoin tab (I, t)) Q)
⟨*proof*⟩

lemma *projectTable_idle*:
assumes *table n A X*
assumes *A ⊆ I*
shows *projectTable I (A, X) = (A, X)*
⟨*proof*⟩

lemma *restrict_partition_merge*:
assumes *I ∪ J = V*
assumes *wf_tuple n V z*
assumes *xx = restrict J z*
assumes *t = restrict I z*
assumes *Set.is_empty (I ∩ J)*
shows *z = merge xx t*
⟨*proof*⟩

lemma *restrict_merge*:
assumes *zI = restrict I z*
assumes *zJ = restrict J z*
assumes *restrict (A ∩ I) zI ∈ Set.image (restrict I) X*
assumes *restrict (A ∩ J) zJ ∈ Set.image (restrict J) (Set.filter (isSameIntersection zI (A ∩ I)) X)*
assumes *z = merge zJ zI*
assumes *table n A X*
assumes *A ⊆ I ∪ J*
assumes *card (A ∩ I) ≥ 1*
assumes *wf_set n (I ∪ J)*
assumes *wf_tuple n (I ∪ J) z*
shows *restrict A z ∈ X*
⟨*proof*⟩

lemma *partial_correctness*:
assumes $V = I \cup J$
assumes *Set.is_empty* $(I \cap J)$
assumes *card* $I \geq 1$
assumes *card* $J \geq 1$
assumes *Q_I_pos* = *projectQuery* I (*filterQuery* I Q)
assumes *Q_J_pos* = *filterQuery* J Q
assumes *Q_I_neg* = *filterQueryNeg* I Qn
assumes *Q_J_neg* = *filterQuery* J $(Qn - Q\_I\_neg)$
assumes *NQ_pos* = *newQuery* J *Q_J_pos* (I, t)
assumes *NQ_neg* = *newQuery* J *Q_J_neg* (I, t)
assumes *R_NQ* = *genericJoin* J *NQ_pos NQ_neg*
assumes $\forall x.\ (x \in R\_I \longleftrightarrow wf\_tuple\ n\ I\ x \wedge (\forall (A,\ X) \in Q\_I\_pos.\ restrict\ A\ x \in X) \wedge (\forall (A, X) \in Q\_I\_neg.\ restrict\ A\ x \notin X))$
assumes $\forall y.\ (y \in R\_NQ \longleftrightarrow wf\_tuple\ n\ J\ y \wedge (\forall (A,\ X) \in NQ\_pos.\ restrict\ A\ y \in X) \wedge (\forall (A, X) \in NQ\_neg.\ restrict\ A\ y \notin X))$
assumes $z = merge\ xx\ t$
assumes $t \in R\_I$
assumes $xx \in R\_NQ$
assumes *rwf_query* n V Q Qn
shows *wf_tuple* n V $z \wedge (\forall (A,\ X) \in Q.\ restrict\ A\ z \in X) \wedge (\forall (A,\ X) \in Qn.\ restrict\ A\ z \notin X)$
⟨*proof*⟩

lemma *simple_set_inter*:
assumes $I \subseteq (\bigcup X \in A.\ f\ X)$
shows $I \subseteq (\bigcup X \in A.\ (f\ X) \cap I)$
⟨*proof*⟩

lemma *union_restrict*:
assumes *restrict* I *z1* = *restrict* I *z2*
assumes *restrict* J *z1* = *restrict* J *z2*
shows *restrict* $(I \cup J)$ *z1* = *restrict* $(I \cup J)$ *z2*
⟨*proof*⟩

lemma *partial_correctness_direct*:
assumes $V = I \cup J$
assumes *Set.is_empty* $(I \cap J)$
assumes *card* $I \geq 1$
assumes *card* $J \geq 1$
assumes *Q_I_pos* = *projectQuery* I (*filterQuery* I Q)
assumes *Q_J_pos* = *filterQuery* J Q
assumes *Q_I_neg* = *filterQueryNeg* I Qn
assumes *Q_J_neg* = *filterQuery* J $(Qn - Q\_I\_neg)$
assumes *R_I* = *genericJoin* I *Q_I_pos Q_I_neg*
assumes $X = \{(t,\ genericJoin\ J\ (newQuery\ J\ Q\_J\_pos\ (I,\ t))\ (newQuery\ J\ Q\_J\_neg\ (I,\ t))) \mid t\ .\ t \in R\_I\}$
assumes $R = (\bigcup (t,\ x) \in X.\ \{merge\ xx\ t \mid xx\ .\ xx \in x\})$
assumes *R_NQ* = *genericJoin* J *NQ_pos NQ_neg*
assumes $\forall x.\ (x \in R\_I \longleftrightarrow wf\_tuple\ n\ I\ x \wedge (\forall (A,\ X) \in Q\_I\_pos.\ restrict\ A\ x \in X) \wedge (\forall (A, X) \in Q\_I\_neg.\ restrict\ A\ x \notin X))$
assumes $\forall t \in R\_I.\ (\forall y.\ (y \in genericJoin\ J\ (newQuery\ J\ Q\_J\_pos\ (I,\ t))\ (newQuery\ J\ Q\_J\_neg\ (I, t)) \longleftrightarrow wf\_tuple\ n\ J\ y \wedge$
$(\forall (A,\ X) \in (newQuery\ J\ Q\_J\_pos\ (I,\ t)).\ restrict\ A\ y \in X) \wedge (\forall (A,\ X) \in (newQuery\ J\ Q\_J\_neg\ (I, t)).\ restrict\ A\ y \notin X)))$
assumes *wf_tuple* n V $z \wedge (\forall (A,\ X) \in Q.\ restrict\ A\ z \in X) \wedge (\forall (A,\ X) \in Qn.\ restrict\ A\ z \notin X)$
assumes *rwf_query* n V Q Qn
shows $z \in R$

⟨*proof*⟩

lemma *obvious_forall*:
assumes ∀ *x*∈*X*. *P x*
assumes *x*∈*X*
shows *P x*
⟨*proof*⟩

lemma *correctness*:
⟦*rwf_query n V Q Qn*; *card V* ≥ *1*⟧ ⟹ (*z* ∈ *genericJoin V Q Qn* ⟷ *wf_tuple n V z* ∧
(∀ (*A*, *X*)∈*Q*. *restrict A z* ∈ *X*) ∧ (∀ (*A*, *X*)∈*Qn*. *restrict A z* ∉ *X*))
⟨*proof*⟩

lemma *wf_set_finite*:
assumes *wf_set n A*
shows *finite A*
⟨*proof*⟩

lemma *vars_wrapperGenericJoin*:
fixes *Q* :: *'a query* **and** *Q_pos* :: *'a query* **and** *Q_neg* :: *'a query*
and *V* :: *nat set* **and** *Qn* :: *'a query*
assumes *Q* = *Set.filter* (λ(*A*, _). ¬ *Set.is_empty A*) *Q_pos*
and *V* = (⋃ (*A*, *X*)∈*Q*. *A*)
and *Qn* = *Set.filter* (λ(*A*, _). *A* ⊆ *V* ∧ *card A* ≥ *1*) *Q_neg*
and ¬ *Set.is_empty Q*
and ¬((∃ (*A*, *X*)∈*Q_pos*. *Set.is_empty X*) ∨ (∃ (*A*, *X*)∈*Q_neg*. *Set.is_empty A* ∧ ¬ *Set.is_empty X*))
shows *wrapperGenericJoin Q_pos Q_neg* = *genericJoin V Q Qn*
⟨*proof*⟩

lemma *wrapper_correctness*:
assumes *card Q_pos* ≥*1*
assumes ∀ (*A*, *X*)∈(*Q_pos* ∪ *Q_neg*). *table n A X* ∧ *wf_set n A*
shows *z* ∈ *wrapperGenericJoin Q_pos Q_neg* ⟷ *wf_tuple n* (⋃ (*A*, *X*)∈*Q_pos*. *A*) *z* ∧ (∀ (*A*, *X*)∈*Q_pos*. *restrict A z* ∈ *X*) ∧ (∀ (*A*, *X*)∈*Q_neg*. *restrict A z* ∉ *X*)
⟨*proof*⟩

end

end

3 Example instantiations and queries

theory *Examples_Join*
imports *Generic_Join*
begin

3.1 Instantiations

global_interpretation *Max_getIJ*: *getIJ* λ_ _ *V*. (*V* − {*Max V*}, {*Max V*})
defines *Max_getIJ_genericJoin* = *Max_getIJ.genericJoin*
and *Max_getIJ_wrapperGenericJoin* = *Max_getIJ.wrapperGenericJoin*
⟨*proof*⟩

global_interpretation *Min_getIJ*: *getIJ* λ_ _ *V*. ({*Min V*}, *V* − {*Min V*})
defines *Min_getIJ_genericJoin* = *Min_getIJ.genericJoin*
and *Min_getIJ_wrapperGenericJoin* = *Min_getIJ.wrapperGenericJoin*

⟨*proof*⟩

3.2 Queries

value *Max_getIJ.genericJoin* {*0*, *1*} {({*0* , *1*}, {[*Some* (*0* :: *nat*), *Some 0*], [*Some 1*, *Some 1*]}), ({*0* , *1*}, {[*Some 0*, *Some 0*], [*Some 0*, *Some 1*]})} {} :: *nat table*
value *Min_getIJ.genericJoin* {*0*, *1*} {({*0* , *1*}, {[*Some* (*0* :: *nat*), *Some 0*], [*Some 1*, *Some 1*]}), ({*0* , *1*}, {[*Some 0*, *Some 0*], [*Some 0*, *Some 1*]})} {} :: *nat table*

fun *protoTableTriangle* :: *nat* ⇒ *nat table* **where**
protoTableTriangle 0 = {[*Some 0*, *Some 0*]}
| *protoTableTriangle* (*Suc n*) = (*protoTableTriangle n*) ∪ {[*Some* (*Suc n*), *Some 0*], [*Some 0*, *Some* (*Suc n*)]}

fun *auxInsertNoneTriangle* :: *nat tuple* ⇒ *nat* ⇒ *nat tuple* **where**
auxInsertNoneTriangle l 0 = *None* # *l*
| *auxInsertNoneTriangle* (*x* # *q*) (*Suc n*) = *x* # (*auxInsertNoneTriangle q n*)
| *auxInsertNoneTriangle* [] (*Suc v*) = *undefined*

fun *insertNoneTriangle* :: *nat table* ⇒ *nat* ⇒ *nat table* **where**
insertNoneTriangle t n = {*auxInsertNoneTriangle x n* | *x* . *x* ∈ *t*}

value *set* [*0* ..< *5*]

fun *getTableTriangle* :: *nat* ⇒ *nat* ⇒ *nat atable* **where**
getTableTriangle n i = ({*0*, *1*, *2*} − {*i*}, *insertNoneTriangle* (*protoTableTriangle n*) *i*)

fun *getQueryTriangle* :: *nat* ⇒ *nat query* **where**
getQueryTriangle n = {*getTableTriangle n 0*, *getTableTriangle n 1*, *getTableTriangle n 2*}

definition *verticesTriangle* :: *vertices* **where** *verticesTriangle* = {*0*, *1*, *2*}

value *getQueryTriangle 2*

value *Max_getIJ.genericJoin verticesTriangle* (*getQueryTriangle 2*) {({*0*, *2*}, {[*Some 0*, *None*, *Some 0*]})}

value *let n = 2 in let* ((_, *A*), (_, *B*), (_, *C*)) = (*getTableTriangle n 0*, *getTableTriangle n 1*, *getTableTriangle n 2*) *in*
let AB = join A True B in join AB True C
value *Min_getIJ.wrapperGenericJoin* (*getQueryTriangle 2*) {}
value *Max_getIJ.wrapperGenericJoin* (*getQueryTriangle 2*) {}
value *New_max.wrapperGenericJoin* (*getQueryTriangle 2*) {}

end

References

[1] H. Q. Ngo, C. Ré, and A. Rudra. Skew strikes back: New developments in the theory of join algorithms. *SIGMOD Rec.*, 42(4):5–16, Feb. 2014.