A NORMAL FORM FOR PCSJ EXPRESSIONS

Sieger van Denneheuvel
Gerard R. Renardel de Lavalette

ITLI Prepublication Series
for Computation and Complexity Theory CT-90-02

A normal form for PCSJ expressions

Sieger van Denneheuvel
Department of Mathematics and Computer Science
University of Amsterdam

Gerard R. Renardel de Lavalette
Department of Philosophy
University of Utrecht

February 9, 1990

Abstract

In this paper we construct a normal form for a relational algebra, consisting of **P**rojection, **S**election and **J**oin, extended with **C**alculation and define a corresponding language **PCSJL**. An operator for renaming of attributes is also implicitly available in this algebra, since renaming can be defined directly in terms of calculation and projection. Moreover, the normal form can be extended easily to include the union operator. **PCSJL** plays a role in the implementation of the Rule Language **RL**; the normalization is to be used for query optimisation.

1 Introduction

PSJ expressions are relational algebra expressions containing only project, select and join operators. This restricted class of expressions is commonly used in relational databases. PSJ expressions are studied in [YAN87] and [LAR85], where it is mentioned without proof (which is not very difficult) that they can be transformed into a normal form where first the join operators are applied, then selection and finally projection. Such normalization procedures play an important role in query optimisation: see [ULL89] Ch. 11 and [YAN87]. Standard optimization techniques can be used to further optimize PSJ normal form expressions: e.g. in special circumstances the 'selection before join' heuristic can be applied to push selection down to the relational database tables ([ULL89]).

In this paper we add the relational operator *calculate* to the above mentioned relational operators, thus obtaining the language **PCSJL** of PCSJ expressions. The question arises whether PCSJ expressions also can be brought into a normal form. It is shown in Section 4 that a normal form exists where the joins are followed by selection, then calculate and finally projection; moreover, the proof (which is quite technical) yields an easy construction that has been converted into an implemented algorithm in Section 5. This PCSJ normal form procedure already provides optimization, and the normal form it yields can serve as the starting point for further optimization (just as for PSJ normal forms). The proof and the construction can be extended to include the union operator (see [DEN89]). In addition renaming for attributes can be incorporated in the normal form without extra effort since it can be defined directly in terms of the calculate and projection operators.

Our interest for PCSJ expressions lies in its role in the integration of relational databases and constraint solving. This integration is one of the aims of the declarative Rule Language **RL**. The potential for such an integration has been investigated in the context of the Rules Technology project led by Peter Lucas at the IBM San Jose Research Center: see e.g. [HANS89],[HANS88]. **RL** was defined by Peter van Emde Boas in [VEMD86a], where a relational semantic model is given to interpret **RL** (see also [VEMD86b],[VEMD86c]). A considerable part of this language has been implemented by the first author of this paper: see [DEN88a] and [DEN88b].

RL can be considered as an extension of **SQL** with existential quantification over variables occurring in constraints but not necessarily in relations (as is required in **SQL**, where all variables in the WHERE clause have to be present in the FROM clause; see [DATE87],[DATE89]). As a consequence, not all expressions in **RL** can be evaluated: imagine what happens when the existential quantifier ranges over an infinite domain. To be able to deal with these problems, the above-mentioned implementation of **RL** is equipped with a *constraint solver*. This constraint solver transforms evaluable **RL**-expressions into expressions of **PCSJL**, which corresponds to the fragment of **RL** without existential quantification.

Outline of the rest of this paper. In Section 2 we introduce the *constraint language* **CL** which serves as a parameter of the language **PCSJL**, defined in Section 3; Section 4 contains the normalization procedure, in Section 5 we shortly mention its implementation and finally in Section 6 we present conclusions and a perspective on future research.

Acknowledgements. The authors thank Peter van Emde Boas, Karen Kwast and Edith Spaan (all University of Amsterdam) for useful criticism and remarks.

2 The constraint language CL

We begin with the definition of the *constraint language* **CL**: it will act as a parameter of the language **PCSJL** defined in Section 3. **CL** is a many-sorted language containing variables (denoted by the metavariables $x, y, \ldots$, also called *attributes*), constants ($c, d, \ldots$, also called *values*), functions ($f, g, \ldots$), $=$ (the equality predicate), predicates, propositional connectives ($\neg, \wedge, \vee, \rightarrow$) and the propositional constant **true**. Terms ($s, t, \ldots$) and assertions ($A, B, \ldots$, also called *constraints* or *conditions*) are defined as usual.

If I is any of the items defined above (or a collection of these), then $var(I)$ is the set of variables occurring in I. Furthermore, we assume some evaluation mechanism $|\_|$ for **CL** to be given, which evaluates closed terms (terms without variables) to constants and closed assertions to truth values.

We give an example language for **CL**, defined by the following sorts constants, functions and predicates:

sorts: **NUM** (natural numbers), **STR** (strings of characters)

constants: 0,1,2,...in **NUM**, all finite strings in **STR**

functions:
$*, + :$ **NUM** $\times$ **NUM** $\rightarrow$ **NUM**
cat: **STR** $\times$ **STR** $\rightarrow$ **STR** (concatenation)
length: **STR** $\rightarrow$ **NUM** (length of a string)
digits: **NUM** $\rightarrow$ **STR** (converts a number to its string representation)

predicates: $<, >, \leq, \geq, \neq$ (binary predicates, both on **NUM** and on **STR**)

3 The language PCSJL

Before we define the sorts of the language **PCSJL**, we introduce the following.

Definition 1 *A solution is a constraint of the form $x = t$ with $x \notin var(t)$.*

Definition 2 *A solution set is a finite set $\{x_1 = t_1, \ldots, x_n = t_n\}$, satisfying:*
1. $\|\{x_1, \ldots, x_n\}\| = n$, *(the variables are distinct)*
2. $\{x_1, \ldots, x_n\} \cap var(\{t_1, \ldots, t_n\}) = \emptyset$

A *tuple* (denoted by $\phi, \psi, \ldots$) is a solution set of the form $\{x_1 = c_n, \ldots, x_n = c_n\}$. For tuples ϕ, we often write $attr(\phi)$ instead of $var(\phi)$. Tuples are called *similar* if they have the same attributes. A *relation* R is a pair $< X, R' >$ of a finite collection of attributes X and a finite collection of similar tuples, satisfying:

$$\forall \phi \in R' \; attr(\phi) = X$$

If R' is non-empty then X can be obtained from R'; since most relations are nonempty, we shall allow ourselves to be a bit sloppy and identify R and R'; i.e. consider a base relation to be a collection of similar tuples.

Example 1 *(solution sets and tuples)*
$\{name =$ *'bob'*, $age = 55$, $dep =$ *'toy'*$\}$ *(a tuple)*
$\{x = 1, y = 2, z = 3\}$ *(a tuple)*
$\{x = u + 2, y = v + 2\}$ *(a solution set)*

Assume that an instance of **CL** is given, i.e. some language with sorts, variables, constants, etc. We now present the definition of the language **PCSJL** = **PCSJL(CL)**; the interpretation of the language is given together with its definition. **PCSJL** is a four-sorted language with expressions (thus named to distinguish them from **CL**-terms) and equations. The sorts are:

- $\mathcal{V}$ (finite sets of **CL**-variables)
- $\mathcal{C}$ (constraints, i.e. **CL**-assertions)
- $\mathcal{S}$ (solutions sets)
- $\mathcal{R}$ (relations)

We let X, Y, Z range over $\mathcal{V}$; A, B, C over $\mathcal{C}$; Φ, Ψ over $\mathcal{S}$; R, S over $\mathcal{R}$. Next we present the functions of **PCSJL**. They are grouped according to their range.

3.1 Functions with range $\mathcal{V}$

Besides the usual set operations $\cup, \cap$ and $-$, we have:

Definition 3 *(variables and attributes)*
$var : \mathcal{C} \rightarrow \mathcal{V}$
$attr : \mathcal{R} \rightarrow \mathcal{V}$

Definition 4 *(head and tail variables)*
$hvar : \mathcal{S} \rightarrow \mathcal{V}$
$tvar : \mathcal{S} \rightarrow \mathcal{V}$
$hvar(\{x_1 = t_1, \ldots, x_n = t_n\}) = \{x_1, \ldots, x_n\}$
$tvar(\{x_1 = t_1, \ldots, x_n = t_n\}) = var(\{t_1, \ldots, t_n\})$

3.2 Functions with range $\mathcal{C}$

Besides $\wedge$ (conjunction), we have the *merge* of two solution sets, yielding a constraint. The merge function is defined as follows:

Definition 5 $\_ \oplus \_ : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{C}$
$\Phi \oplus \Psi = (\textit{the conjunction of}\ \{s = t \mid x = s \in \Phi, x = t \in \Psi\})$

Example 2 *(merging solution sets)*
$\{x = \text{'}bob\text{'}\} \oplus \{x = y\ cat\ z\} = \{\text{'}bob\text{'} = y\ cat\ z\}$
$\{x = u + 2, y = 3\} \oplus \{x = v + 2\} = \{u + 2 = v + 2\}$

Solution sets $\Phi = \{x_1 = t_1, \ldots, x_n = t_n\}$ can be interpreted as substitutions $[x_1 := t_n, \ldots, x_n := t_n]$ which can be applied to (collections of) items. So we have an operation *apply*:

Definition 6 $\_(\_) : \mathcal{S} \times \mathcal{C} \rightarrow \mathcal{C}$
$\Phi(A) = (\Phi$ *considered as a substitution, applied to* $A)$

Example 3 *(substitution on constraints)*
$\{x = u + 2\}(\{x = u + 1\}) = \{u + 2 = u + 1\}$
$\{x = u + 2, y = v + 2\}(\{x > y\}) = \{u + 2 > v + 2\}$

3.3 Functions with range $\mathcal{S}$

Here, too, we have the usual set operations $\cup, \cap$ and $-$; besides, we introduce the *restrict* and *delete* functions:

Definition 7 $\_[\_] : \mathcal{S} \times \mathcal{V} \rightarrow \mathcal{S}$
$\Phi[X] = \{x = t \in \Phi \mid x \in X\}$

Definition 8 $\_\langle\_\rangle : \mathcal{S} \times \mathcal{V} \rightarrow \mathcal{S}$
$\Phi\langle X\rangle = \{x = t \in \Phi \mid x \notin X\}$

Example 4 *(restriction and deletion)*
$\{x = z + 1, y = z + 2\}[\{x\}] = \{x = z + 1\}$
$\{x = z + 1, y = z + 2\}\langle\{x\}\rangle = \{y = z + 2\}$
$\{x = 1, y = 2, z = 3\}\langle\{x\}\rangle[\{y\}] = \{y = 2\}$

Further we also have substitution on solutions:

Definition 9 $\_(\!(\_)\!) : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$
$\Phi(\!(\Psi)\!) = \{x = \Phi(t) \mid x = t \in \Psi\}$

Example 5 *(substitution on solution sets)*
$\{x = u + 2\}(\!(\{x = u + 1\})\!) = \{x = u + 1\}$
$\{x = u + 2, y = v + 2\}(\!(\{x = u + 1, 3 = y\})\!) = \{x = u + 1, 3 = v + 2\}$

3.4 Functions with range $\mathcal{R}$

Here we find the usual projection, selection and join operators on relations, together with the calculate operator. The definitions are:

Definition 10 $\pi : \mathcal{R} \times \mathcal{V} \rightarrow \mathcal{R}$
$\pi(R, X) = \{\phi[X] \mid \phi \in R\}$ *if* $X \subset attr(R)$

Definition 11 $\sigma : \mathcal{R} \times \mathcal{C} \rightarrow \mathcal{R}$
$\sigma(R, X) = \{\phi \in R \mid |\phi(A)| = \mathbf{true}\}$ *if* $var(A) \subset attr(R)$

Definition 12 $\kappa : \mathcal{R} \times \mathcal{S} \rightarrow \mathcal{R}$
$\kappa(R, \Phi) = \{\psi \cup \psi(\!(\Phi)\!) \mid \psi \in R\}$ *if* $tvar(\Phi) \subset attr(R)$ *and* $hvar(\Phi) \cap attr(R) = \emptyset$

Definition 13 $\_ \bowtie \_ : \mathcal{R} \times \mathcal{R} \rightarrow \mathcal{R}$
$R \bowtie S = \{\phi \cup \psi \mid \phi \in R, \psi \in S, \forall x \in attr(\phi) \cap attr(\psi)\ (\phi[x] = \psi[x])\}$

One readily observes that the project, select and calculate operators are *partial* since they are only defined when certain conditions on the arguments are met. These condition are referred to as *definedness* conditions. They are quite reasonable: the definedness condition for projection ensures that a relation is not projected on attributes that are not part of the relation; the definedness condition for selection takes care that the constraint A can indeed be evaluated to **true** or **false**; the first part of the definedness condition for the calculate operator ensures that the tails of solutions in Φ can be evaluated, the second part rules out the possibility that the head of a solution is also determined directly by an attribute of the relation R.

Example 6 *(extending tuples with the calculate operator)*
$r(x,y) = \{\{x = 1, y = 2\}\}$
$\kappa(r(x,y), \{u = x + y, v = \text{'}bob\text{'}\}) = \{\{x = 1, y = 2, u = 3, v = \text{'}bob\text{'}\}\}$

An operator for attribute renaming can be defined with use of the project and calculate operators in the following way:

Definition 14 $\rho : \mathcal{R} \times \mathcal{S} \rightarrow \mathcal{R}$
$\rho(R, \Phi) = \pi(\kappa(R, \Phi), attr(R) \cup hvar(\Phi) - tvar(\Phi))$
if $tvar(\Phi) \subset attr(R)$ *and* $hvar(\Phi) \cap attr(R) = \emptyset$

Note that the defined renaming operator is slightly more general than usual attribute renaming, since in the tails of Φ terms are allowed. The renaming operator is invoked with the renaming given in the solution set Φ. The solution set contains elements of the form $x = y$ such that y is among the attributes of R and x is not:

Example 7 *(renaming an attribute)*
$r(x,y) = \{\{x = 1, y = 2\}\}$
$\rho(r(x,y), \{z = y\}) = \{\{x = 1, z = 2\}\}$

The functions defined in this section can be represented as in Figure 1. Functions not listed in the diagram are $\_(\_)$ and $\_(\!(\_)\!)$.

4 Normal forms

In this section, we define normal forms and show that every expression of sort $\mathcal{R}$ is equivalent to an expression in normal form. We call two expressions equivalent if they refer to the same object (for expressions of sort $\mathcal{R}$ this means that they refer to the same relation).

Definition 15 *A normal form is an expression of the form*
$\pi(\kappa(\sigma(R_1 \bowtie \ldots \bowtie R_n, A), \Phi), X)$ *where* $R_1, \ldots, R_n$ *are atomic expressions.*

NF is the collection of normal forms.

Proposition 16 *The normal form* $\pi(\kappa(\sigma(R_1 \bowtie \ldots \bowtie R_n, A), \Phi), X)$ *is defined iff:*
1. $var(A) \subset attr(R_1 \bowtie \ldots \bowtie R_n)$
2. $tvar(\Phi) \subset attr(R_1 \bowtie \ldots \bowtie R_n)$
3. $X \subset attr(R_1 \bowtie \ldots \bowtie R_n) \cup hvar(\Phi)$
4. $attr(R_1 \bowtie \ldots \bowtie R_n) \cup hvar(\Phi) = \emptyset$

Proof: Follows directly from the definedness conditions.
□

Proposition 17 *Consequences of definedness of* $\pi(\kappa(\sigma(R_1 \bowtie \ldots \bowtie R_n, A), \Phi), X)$*:*
1. $tvar(\Phi) \cap hvar(\Phi) = \emptyset$
2. $var(A) \cap hvar(\Phi) = \emptyset$
3. $hvar(\Phi) = (var(A) \cup var(\Phi)) - attr(R_1 \bowtie \ldots \bowtie R_n)$

Proof: Follows directly from the definedness conditions and Proposition 16.
□

Theorem 18 *Every defined expression in* **PCSJL** *of sort* $\mathcal{R}$ *can be transformed into an equivalent defined normal form.*

Proof: Let $\underline{\text{NF}}$ be the collection of all expressions equivalent to a normal form (in other words, $\underline{\text{NF}}$ is the closure of NF under equivalence). It suffices to show the following statements:

1. All atomic expressions of sort $\mathcal{R}$ are in $\underline{\text{NF}}$ (case (1))

2. $\underline{\text{NF}}$ is closed under projection, selection, calculation and join.
   (cases $(2.\pi), (2.\sigma), (2.\kappa)$ and $(2.\bowtie)$ respectively)

This is done as follows:
• (1): An atomic expression R can be rewritten in normal form by:

$$R = \pi(\kappa(\sigma(R, \mathbf{true}), \emptyset), attr(R))$$

In the remaining cases we use the following abbreviations:

$$R := R_1 \bowtie \ldots \bowtie R_n, \; S := S_1 \bowtie \ldots \bowtie S_m$$

• $(2.\pi)$: This is easy: we have to show

$$\pi(\pi(\kappa(\sigma(R, A), \Phi), X), Y) \in \underline{NF}$$

and this follows from

$$\pi(\pi(\kappa(\sigma(R, A), \Phi), X), Y) = \pi(\kappa(\sigma(R, A), \Phi), Y)$$

for it follows from the definedness conditions that $Y \subset X$.

- $(2.\sigma)$: This case is slightly more complex. We must show:

$$\sigma(\pi(\kappa(\sigma(R,A),\Phi),X),B) \in \underline{NF}$$

To obtain a defined normal form for this expression, we have to use substitution. The critical point in the construction is that variables from $hvar(\Phi)$ may be present in the constraint B. If B were added straightaway to the solution part of the new normal form, an undefined normal form would be constructed. Therefore the variables in $hvar(\Phi)$ are substituted by the substitution in $\Phi(B)$, resulting in a defined normal form:

$$\sigma(\pi(\kappa(\sigma(R,A),\Phi),X),B) = \pi(\kappa(\sigma(R,A \wedge \Phi(B)),\Phi),X)$$

- $(2.\kappa)$: Here we want:

$$\kappa(\pi(\kappa(\sigma(R,A),\Phi),X),\Psi) \in \underline{NF}$$

We begin with assuming that the following condition holds:

$$hvar(\Psi) \cap (attr(R) \cup hvar(\Phi)) - X = \emptyset \tag{1}$$

For this case we can show that under assumption of (1) the next condition is true so that $\Phi \cup \Psi$ (and consequently also $\Phi \cup \Phi((\Psi))$) is a solution set:

$$hvar(\Phi) \cap hvar(\Psi) = \emptyset \tag{2}$$

For suppose $x \in hvar(\Phi) \cap hvar(\Psi)$. Then either $x \in X$ or $x \notin X$. If $x \in X$, the definedness conditions are violated since x is both determined by a head of a solution in Ψ and the relation $\pi(\kappa(\sigma(R,A),\Phi),X)$. On the other hand, if $x \notin X$ then (1) does not hold, contrary to our assumption. So it must be that the assumption $x \in hvar(\Phi) \cap hvar(\Psi)$ was false and hence (2) holds.

A second reason for adopting (1) is to ensure that the variables projected away by the projection on X, should be different from $hvar(\Psi)$, the new variables introduced by Ψ. But this is already the case since we have assumed (1) to hold. The resulting normal form is:

$$\kappa(\pi(\kappa(\sigma(R,A),\Phi),X),\Psi) = \pi(\kappa(\sigma(R,A),\Phi \cup \Phi((\Psi))),X \cup hvar(\Psi))$$

If (1) does not hold, then in R, A and Φ the variables in $(attr(R) \cup hvar(\Phi)) - X$ have to be renamed in order to make them different from those in $hvar(\Psi)$. After the renaming (1) holds and the rest of the argument runs analogously.

- $(2.\bowtie)$: This last case is the most complex. What we want is:

$$\pi(\kappa(\sigma(R,A),\Phi),X) \bowtie \pi(\kappa(\sigma(S,B),\Psi),Y) \in \underline{NF}$$

The condition to prevent an undesired clash of variables now reads:

$$(attr(R) \cup hvar(\Phi)) \cap (attr(S) \cup hvar(\Psi)) - (X \cap Y) = \emptyset \tag{3}$$

We start with assuming that (3) holds. Then $\Phi \cup \Psi$ is, in general, not a solution set. The merge function $\oplus$ in the construction below handles this case. Also another case needs to be checked. Suppose there is a solution $x = t \in \Phi$ with $x \in attr(S)$. If this solution were put in the calculate part of the new normal form, then the resulting expression would be undefined. The problem can be handled by recognizing that $x = t$ now satisfies the definedness conditions of the select operator, viz. $var(x = t) \subset attr(R) \cup attr(S)$. So the restriction operator below inserts the solution $x = t$ in the select condition C of the new normal form below and the delete operator deletes it from the calculate part Θ. The symmetric case that there is a solution $x = t \in \Psi$ so that $x \in attr(R)$, is handled in the same way. The resulting normal form, which is defined, now reads:

$$\pi(\kappa(\sigma(R, A), \Phi), X) \bowtie \pi(\kappa(\sigma(S, B), \Psi), Y) = \pi(\kappa(\sigma(R \bowtie S, C), \Theta), X \cup Y)$$
with
$$C = A \wedge B \wedge \Phi \oplus \Psi \wedge \Phi[attr(S)] \wedge \Psi\langle hvar(\Phi)\rangle[attr(R)]$$
and
$$\Theta = \Phi\langle attr(S)\rangle \cup \Psi\langle hvar(\Phi)\rangle\langle attr(R)\rangle$$

The above construction can be explained in the following way. The solution sets Φ and Ψ are first transformed into the constraints $\Phi \oplus \Psi$ and the solution sets $\Psi\langle hvar(\Phi)\rangle$ and Φ. Next $\Phi \oplus \Psi$ is put directly into the select condition C; the remaining pair of solution sets $\Psi\langle hvar(\Phi)\rangle$ and Φ needs to be processed further. On both solution sets restrictions are applied to see whether more solutions can be turned into select conditions. The applied restrictions are compensated by the delete operators in the calculate part.

It should be noted that in the above construction, the expression

$$\Psi\langle hvar(\Phi)\rangle[attr(R)]$$

can be replaced by the more simple expression $\Psi[attr(R)]$. However this could lead to duplicate use of solutions from Ψ in the condition of the resulting normal form and since normal forms are to be used for query optimisation we want to avoid this duplication.

If (3) does not hold then we are going to rename variables. The condition (3) is equivalent with the conjunction of the following two conditions:

$$((attr(R) \cup hvar(\Phi)) - X) \cap (attr(S) \cup hvar(\Psi)) = \emptyset \tag{4}$$
$$(attr(R) \cup hvar(\Phi)) \cap ((attr(S) \cup hvar(\Psi)) - Y) = \emptyset \tag{5}$$

First we rename in R, A and Φ the variables in $(attr(R) \cup hvar(\Phi)) - X$ in order to make them different from the variables $attr(S) \cup hvar(\Psi)$. After renaming (4)

holds. In a similar way the offending variables for condition (5) are renamed in S, B and Ψ. After these renamings both (4) and (5) hold, so also (3) holds. The rest of the argument runs analogously.
This ends the proof of Theorem 18.

□

The standard normalization construction for PSJ expressions can be obtained from the above normalization construction for PCSJ expressions by taking both $\Phi = \emptyset$ and $\Psi = \emptyset$. The calculate rule is dropped from the normalization construction.

Union can be included in the normalization process (see [DEN89]). In the normal form the union operator is added as the outermost operator, having as arguments the PCSJ normal forms described in this paper.

5 Implementation

The normalization procedure of the previous section has been implemented in Prolog. The sample problems listed below were run on this prototype. Output was adapted to the notation of this paper and the listed problems illustrate the use of the join rule (2.$\bowtie$).

Example 8 *(Conditions and solutions are combined directly)*
$\pi(\kappa(\sigma(r(a,b), a > b), \{x = a + b\}), \{a, b, x\})$
$\bowtie \pi(\kappa(\sigma(s(c,d), c > d), \{y = c + d\}), \{c, d, y\})$
$\rightarrow \pi(\kappa(\sigma(r(a,b) \bowtie s(c,d), a > b \wedge c > d), \{x = a + b, y = c + d\}), \{a, b, x, c, d, y\})$

Example 9 *(A solution is used as a condition in the new normal form)*
$\pi(\kappa(\sigma(r(a,b), a > b), \{x = a + b\}), \{a, b, x\})$
$\bowtie \pi(\kappa(\sigma(s(x,d), x > d), \{y = x + d\}), \{x, d, y\})$
$\rightarrow \pi(\kappa(\sigma(r(a,b) \bowtie s(x,d), a > b \wedge x > d \wedge x = a + b), \{y = x + d\}), \{a, b, x, d, y\})$

Example 10 *(A variable needs to be renamed)*
$\pi(\kappa(\sigma(r(a,b), a > b), \{x = a + b\}), \{a, b, x\})$
$\bowtie \pi(\kappa(\sigma(s(x,d), x > d), \{y = x + d\}), \{d, y\})$
$\rightarrow \pi(\kappa(\sigma(r(a,b) \bowtie s(u_1,d), a > b \wedge u_1 > d), \{x = a + b, y = u_1 + d\}), \{a, b, x, d, y\})$

Example 11 *(A variable occurs as a head in both solution sets)*
$\pi(\kappa(\sigma(r(a,b), a > b), \{x = a - b\}), \{a, b, x\})$
$\bowtie \pi(\kappa(\sigma(s(c,d), c > d), \{x = c - d\}), \{c, d, x\})$
$\rightarrow \pi(\kappa(\sigma(r(a,b) \bowtie s(c,d), a > b \wedge c > d \wedge a - b = c - d), \{x = a - b\}), \{a, b, x, c, d\})$

6 Conclusions

In this paper we have defined **PCSJL**, a language with expressions built up using the operations projection, selection, join and calculate. We have shown that a Normal Form Theorem for this language exists, by giving a construction to transform arbitrary relational expressions into normal form. Since renaming for attributes can be directly defined in terms of the above relational operators, it is included in the normalization construction. Also the union operator can be added to this framework without difficulty, as we outlined before. Finally we mentioned a prototype system to demonstrate that the construction can be practically implemented.

There are several directions for further research in this area. First of all the translation of **RL**-expressions into **PCSJL**-expressions is to be worked out, making use of a constraint solver: this is currently investigated by the authors. Another interesting point is the existence of a normal form that includes the difference operator. However there seems to be no easy normal form for this case since in general projection does not commute with set difference.

References

[DATE87] Date, C.J., *A Guide to the SQL Standard,* Addison-Wesley Publishing Company 1987.

[DATE89] Date, C.J. & White, C.J., *A Guide to DB2,* (Third Edition), Addison-Wesley Publishing Company 1989.

[DEN88a] van Denneheuvel, S. & van Emde Boas, P., *Constraint solving for databases,* Proc. of NAIC **1**, Apr. 1988

[DEN88b] van Denneheuvel, S. & van Emde Boas, P., *Towards implementing RL,* Preprint CT-88-11, Institute for Language, Logic and Information, University of Amsterdam, 1988

[DEN89] van Denneheuvel, S. & Renardel de Lavalette, G. R., *Normalisation of Database expressions involving Calculations,* Logic Group Preprint Series No.45, Department of Philosophy, University of Utrecht, 1989

[HANS88] Hansen, M.R., *Algebraic Optimization of Recursive Database Queries,* Information Systems and Operations Research 26 (1988) 286-298

[HANS89] Hansen, M.R., Hansen, B.S., Lucas, P. & van Emde Boas, P., *Integrating Relational Databases and Constraint Languages,* in Comput. Lang. Vol. **14**, No. 2, 63-82, 1989.

[LAR85] Larson, P.A., Yang, H.Z., *Computing Queries from Derived Relations,* Proc. of the 11th Intl. Conf. on VLDB, 259-269, (1985).

[ULL89] Ullman, J.D., *Principles of Data and Knowledge - Base Systems,* Volume II: The New Technologies, Computer Science Press, 1989.

[VEMD86a] van Emde Boas, P., *RL, a Language for Enhanced Rule Bases Database Processing,* Working Document, Rep IBM Research, RJ 4869 (51299)

[VEMD86b] van Emde Boas, P., *A semantical model for the integration and modularization of rules,* Proceedings MFCS 12, Bratislava, August 1986, Springer Lecture Notes in Computer Science **233** (1986), 78-92

[VEMD86c] van Emde Boas, H. & van Emde Boas, P., *Storing and Evaluating Horn-Clause Rules in a Relational Database,* IBM J. Res. Develop. **30** (1), (1986), 80-92

[YAN87] Yang, H. Z., Larson, P. A., *Query Transformations for PSJ-queries,* Proc. of the 13th Int. Conf. on VLDB, Brighton, 245-254, (1987)

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\mathrm{REL} \arrow[loop above, "\bowtie"] \arrow[loop right, "\pi"] \arrow[loop below, "\sigma"'] \arrow[loop below, "\kappa\ \rho"] \arrow[r, bend left, "\mathrm{attr}"] & \mathrm{VAR} \arrow[l, bend left] \arrow[d, bend left, "{\_[\_]}\ \ {\_\langle\_\rangle}"] \\
\mathrm{CON} \arrow[u, bend left] \arrow[ur, "\mathrm{var}"] & \mathrm{SOL} \arrow[loop above] \arrow[u, bend left, "\mathrm{tvar}\ \ \mathrm{hvar}"] \arrow[ul] \arrow[l, bend left, "\oplus"]
\end{tikzcd}$$

Fig. 1

Instituut voor Taal, Logica en Informatie
Institute for Language, Logic and Information

Faculteit der Wiskunde en Informatica
(Department of Mathematics and Computer Science)
Plantage Muidergracht 24
1018TV Amsterdam

Faculteit der Wijsbegeerte
(Department of Philosophy)
Nieuwe Doelenstraat 15
1012CP Amsterdam

A NORMAL FORM FOR PCSJ EXPRESSIONS

Sieger van Denneheuvel
Department of Mathematics and Computer Science
University of Amsterdam

Gerard R. Renardel de Lavalette
Department of Philosophy
University of Utrecht

Received April 1990

The ITLI Prepublication Series