Query optimization using rewrite rules

Sieger van Denneheuvel (UvA)
Karen Kwast (UvA)
Gerard R. Renardel de Lavalette (RUU)
Edith Spaan (UvA)

Logic Group
Preprint Series
No. 58
October 1990

Department of Philosophy
University of Utrecht
Heidelberglaan 2
3584 CS Utrecht
The Netherlands

Query optimization using rewrite rules

Sieger van Denneheuvel (UVA)
Karen Kwast (UVA)
Gerard R. Renardel de Lavalette (RUU)
Edith Spaan (UVA)

Department of Mathematics and Computer Science
University of Amsterdam (UVA) &
Applied Logic Group, Department of Philosophy
University of Utrecht (RUU)

Abstract

In this paper we present a normal form for a relational algebra, consisting of Projection, Selection and Join, extended with Calculation and Union and define a corresponding language **UPCSJL**. The construction of this normal form, using unconditional rewrite rules, already provides some optimization. Further optimization can be achieved efficiently by applying conditional rewrite rules that directly operate on the normal form. This approach is compared to more traditional query optimization techniques that do not apply normalization.

1 Introduction

PSJ expressions are relational algebra expressions containing only project, select and join operators. This restricted class of expressions, called **PSJL** in the sequel, is commonly used in relational databases. **PSJL** expressions are studied in [YAN87] and [LAR85], where it is mentioned without proof (which is not very difficult) that they can be reduced 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 optimization: see [ULL89] and [YAN87]. Standard optimization techniques can be used to further optimize **PSJL** 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]).

There are several reasons to apply normalization before optimization. Firstly, normalization reduces the number of relational operators in a relational expression. As a consequence, optimization after normalization can be more efficient since the number of reducible subexpressions (redexes) on which optimization rules are applied is also reduced. Moreover the optimization rules can benefit from the fixed structure of a normal form.

Secondly there is a functional difference between rules used for normalization and rules used for optimization: the former are unconditional whereas the latter are conditional. A normalization rule should always be applicable on a subexpression of the proper syntactical format or else a normal form could not be obtained. On the other hand optimization rules only rewrite if in addition to a proper format also a condition involving the subexpression is satisfied. Therefore optimization is in general more expensive than normalization, since applications of optimization rules may fail.

In this paper we add the relational operators *calculate* and *union* to the above mentioned relational operators, thus obtaining the language of **UPCSJL** expressions. The question arises whether **UPCSJL** expressions also can be reduced into a normal form. We prove the existence of a normal form, where the joins are followed by selection, calculation, projection and finally union; moreover, our proof yields a direct construction. This **UPCSJL** normal form already performs some optimization, but the normal form procedure can serve as the starting point for further optimization (just as for **PSJL** normal forms).

Our interest for **UPCSJL** 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: see [DEN90a] and [DEN90b].

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 [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*, which transforms evaluable **RL** expressions into expressions of **UPCSJL**.

Acknowledgements. The authors thank Peter van Emde Boas (University of Amsterdam) for useful criticism and remarks.

2 The data language DL

We begin with the definition of the *data language* **DL**: it will act as a parameter for the language **UPCSJL**. **DL** 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 E is any of the items defined above (or a collection of these), then $var(E)$ is the set of variables occurring in E. Furthermore, we assume some evaluation mechanism $eval(\_)$ for **DL** to be given, which evaluates closed terms (terms without variables) to constants and closed assertions to truth values.

We give an example language for **DL**, 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 Functions of the language UPCSJL

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

Definition 1 *A solution is an expression 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 $\{x_1 = c_1, \ldots, x_n = c_n\}$. 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)*
$\{name = \text{'bob'}, age = 55, dep = \text{'toy'}\}$ *(a tuple)*
$\{x = 1, y = 2, z = 3\}$ *(a tuple)*
$\{x = u + 2, y = v + 2\}$ *(a solution set)*

Assume that an instance of **DL** is given, i.e. some language with sorts, variables, constants, etc. We now present the functions used in the definition of the language **UPCSJL**. **UPCSJL** is a four-sorted language with expressions (thus named to distinguish them from **DL**-terms) and equations. The sorts are:

- $\mathcal{V}$ (finite sets of **DL**-variables)
- $\mathcal{C}$ (constraints, i.e. **DL**-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 $\mathcal{R}$. An expression of sort $\mathcal{R}$ is also called a *relational expression.* A base relation is sometimes followed by a bracketed list denoting all its attributes. In the sequel we use the notation E^X for renaming an expression E with respect to a variable set X:

Definition 3
$E^X = E'$ *where* E' *is obtained by renaming all variables* $var(E) - X$ *uniquely.*

To allow a brief notation, a renaming $(\_)^X$ can be applied at several places in an expression. In that case all occurrences of $(\_)^X$ denote the *same* renaming:

Example 2 *(renaming variables in expressions with* $X = \{x\}$, $Y = \{y\}$*)*
$\sigma((r(x,y))^X \bowtie s(y,z)^Y, (x > y)^X \wedge (y > z)^Y) \rightarrow \sigma(r(x,u_1) \bowtie s(y,u_2), x > u_1 \wedge y > u_2)$

Next we present the functions of **UPCSJL**. 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 4 *(attributes of a relational expression)*
$attr(\_) : \mathcal{R} \rightarrow \mathcal{V}$

Definition 5 *(head and tail variables of a solution set)*
$hvar(\_) : \mathcal{S} \rightarrow \mathcal{V}$
$tvar(\_) : \mathcal{S} \rightarrow \mathcal{V}$
$hvar(\{x_1 = t_1, \ldots, x_n = t_n\}) = \{x_1, \ldots, x_n\}$
$tvar(\{x_1 = t_1, \ldots, x_n = t_n\}) = var(\{t_1, \ldots, t_n\})$

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 6 $\oplus : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{C}$
$\Phi \oplus \Psi = \{s = t \mid x = s \in \Phi, x = t \in \Psi\}$

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

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

Definition 7 $\_(\_) : \mathcal{S} \times \mathcal{C} \to \mathcal{C}$
$\Phi(A) = A[x_1 := t_1, \ldots, x_n := t_n]$ *with* $\Phi = \{x_1 = t_1, \ldots, x_n = t_n\}$

Example 4 *(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 8 *(restricting and deleting solutions)*
1. $\_[\_] : \mathcal{S} \times \mathcal{V} \to \mathcal{S}$
$\Phi[X] = \{x = t \in \Phi \mid x \in X\}$
2. $\_\langle\_\rangle : \mathcal{S} \times \mathcal{V} \to \mathcal{S}$
$\Phi\langle X\rangle = \{x = t \in \Phi \mid x \notin X\}$

Example 5 *(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 $\_((\_)) : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$
$\Phi((\Psi)) = \{x = eval(\Phi(t)) \mid x = t \in \Psi\}$

Example 6 *(substitution on solution sets)*
$\{x = u + 2\}((\{x = u + 1\})) = \{x = u + 1\}$
$\{u = 2\}((\{x = u + 1\})) = \{x = 3\}$
$\{x = u + 2, y = v + 2\}((\{x = u + 1, 3 = y\})) = \{x = u + 1, 3 = v + 2\}$

3.4 Functions with range $\mathcal{R}$

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

Definition 10 *(primitive relational operators)*
1. $\pi : \mathcal{R} \times \mathcal{V} \to \mathcal{R}$
$\pi(R, X) = \{\phi[X] \mid \phi \in R\} \quad if \quad X \subset attr(R)$
2. $\sigma : \mathcal{R} \times \mathcal{C} \to \mathcal{R}$
$\sigma(R, A) = \{\phi \in R \mid eval(\phi(A)) = \mathbf{true}\} \quad if \quad var(A) \subset attr(R)$
3. $\kappa : \mathcal{R} \times \mathcal{S} \to \mathcal{R}$
$\kappa(R, \Phi) = \{\psi \cup \psi((\Phi)) \mid \psi \in R\} \quad if \quad tvar(\Phi) \subset attr(R)$ *and* $hvar(\Phi) \cap attr(R) = \emptyset$
4. $\_ \bowtie \_ : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$
$R \bowtie S = \{\phi \cup \psi \mid \phi \in R, \psi \in S, \forall x \in attr(\phi) \cap attr(\psi)\ (\phi[x] = \psi[x])\}$
5. $\_ \cup \_ : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$
$R \cup S = \{\phi \mid \phi \in R \vee \phi \in S\} \quad if \quad attr(R) = attr(S)$

One readily observes that the project, select, calculate and union operators are *partial* since they are only defined when certain conditions on the arguments are met. These conditions are referred to as *wellformedness* conditions. They are quite reasonable: the wellformedness condition for projection ensures that a relation is not projected on attributes that are not part of the relation; the wellformedness condition for selection takes care that the constraint A can indeed be evaluated to **true** or **false**; the first part of the wellformedness 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. The constraint set A in $\sigma(R, A)$ is the *condition* of the select operator and the solution set Φ in $\kappa(R, \Phi)$ is the *computation* of the calculate operator.

Example 7 *(extending tuples with the calculate operator)*
$r(x,y) := \{\{x=1, y=2\}\}$
$\kappa(r(x,y), \{u = x+y, v = \text{'bob'}\}) = \{\{x=1, y=2, u=3, v=\text{'bob'}\}\}$

4 Relational rewrite rules

In this section we describe a large number of rules handling the properties of the primitive relational operators. Since these operators involve wellformedness conditions we introduce a special notation to emphasize in what direction a rule can be applied:

Definition 11 *(notation for rewriting)*
1. $R \rightarrow S$ *iff* R *is wellformed* $\Rightarrow$ S *is wellformed* & $R = S$
2. $R \leftrightarrow S$ *iff* $R \rightarrow S$ & $S \rightarrow R$

For some rewrite rules $R \rightarrow S$, wellformedness of R does not imply wellformedness of all subexpressions of S and in this case the wellformedness conditions on the violating subexpressions are represented in the rewrite rule as a condition.

4.1 Projection, selection and calculation

The projection, selection and calculate operators have in common that they are applied on a single relational argument and therefore we discuss them together in this section. In the sequel we cluster rules that are similar in one proposition if possible.

Proposition 12 *(cascade rules)*
1. $\pi(\pi(R,X),Y) \rightarrow \pi(R,Y)$ ..... $[\pi\pi]$
2. $\sigma(\sigma(R,A),B) \leftrightarrow \sigma(R, A \wedge B)$ ..... $[\sigma\sigma]$
3. $\sigma(\sigma(R,A),B) \leftrightarrow \sigma(\sigma(R,B),A)$ ..... $[\sigma\sigma*]$
4. $\kappa(\kappa(R,\Phi),\Psi) \rightarrow \kappa(R, \Phi \cup \Phi((\Psi)))$ ..... $[\kappa\kappa]$
5. $\kappa(\kappa(R,\Phi),\Psi) \leftrightarrow \kappa(R, \Phi \cup \Psi)$ *if* $hvar(\Phi) \cap tvar(\Psi) = \emptyset$ ..... $[\kappa\kappa*]$
6. $\pi(R, attr(R)) \leftrightarrow R$ ..... $[\pi attr]$

Proposition 13 *(selection and projection)*
1. $\sigma(\pi(R,X),A) \rightarrow \pi(\sigma(R,A),X)$ ..... $[\sigma\pi]$
2. $\pi(\sigma(R,A),X) \rightarrow \pi(\sigma(\pi(R, attr(R) \cap (var(A) \cup X)),A),X)$ ..... $[\pi\sigma]$
3. $\pi(\sigma(R,A),X) \rightarrow \sigma(\pi(R,X),A)$ *if* $var(A) \subset X$ ..... $[\pi\sigma*]$

Proposition 14 *(selection and calculation)*
1. $\sigma(\kappa(R,\Phi),A) \rightarrow \kappa(\sigma(R, \Phi(A)),\Phi)$ ..... $[\sigma\kappa]$
2. $\kappa(\sigma(R,A),\Phi) \rightarrow \sigma(\kappa(R,\Phi),A)$ ..... $[\kappa\sigma]$

Proposition 15 *(calculation and projection)*
1. $\kappa(\pi(R,X),\Phi) \rightarrow \pi(\kappa(R^X,\Phi), hvar(\Phi) \cup X)$ ..... $[\kappa\pi]$
2. $\kappa(\pi(R,X),\Phi) \rightarrow \pi(\kappa(R,\Phi), hvar(\Phi) \cup X)$ *if* $attr(R) \cap hvar(\Phi) = \emptyset$ ..... $[\kappa\pi*]$
3. $\pi(\kappa(R,\Phi),X) \rightarrow \pi(\kappa(R,\Phi[X]),X)$ ..... $[\pi\kappa]$
4. $\pi(\kappa(R,\Phi),X) \rightarrow \pi(\kappa(\pi(R, tvar(\Phi) \cup (X \cap attr(R))),\Phi),X)$ ..... $[\pi\kappa*]$

Note that in the rule $(\kappa\pi)$ by the renaming in R^X and the wellformedness of $\kappa(\pi(R,X),\Phi)$ all variables $hvar(\Phi)$ are different from the variables $attr(R^X)$, as required.

4.2 Join and union

In this section we add the join and union operators in our list of rules. As before some rules have both conditional and unconditional versions.

Proposition 16 *(symmetry and associativity of* $\bowtie$ *and* $\cup$*)*
1. $R \bowtie S \leftrightarrow S \bowtie R$ ..... $[\text{SYM}\circ \bowtie]$
2. $(R \bowtie S) \bowtie T \leftrightarrow R \bowtie (S \bowtie T)$ ..... $[\text{ASS}\circ \bowtie]$
3. $R \cup S \leftrightarrow S \cup R$ ..... $[\text{SYM} \circ \cup]$
4. $(R \cup S) \cup T \leftrightarrow R \cup (S \cup T)$ ..... $[\text{ASS} \circ \cup]$

Proposition 17 *(join and calculation)*

1. $R \bowtie \kappa(S, \Phi) \rightarrow \kappa(\sigma(R \bowtie S, \Phi[attr(R)]), \Phi\langle attr(R)\rangle)$ ..... *[⋈κ]*
2. $R \bowtie \kappa(S, \Phi) \rightarrow \kappa(R \bowtie S, \Phi)$ *if* $hvar(\Phi) \cap attr(R) = \emptyset$ ..... *[⋈κ∗]*
3. $R \bowtie \kappa(S, \Phi) \rightarrow \sigma(R \bowtie S, \Phi)$ *if* $hvar(\Phi) \subset attr(R)$ ..... *[⋈κ+]*
4. $\kappa(R \bowtie S, \Phi) \rightarrow R \bowtie \kappa(S, \Phi)$ *if* $tvar(\Phi) \subset attr(S)$ ..... *[κ⋈]*

Proposition 18 *(join and projection)*

1. $R \bowtie \pi(S, X) \rightarrow \pi(R \bowtie S^X, attr(R) \cup X)$ ..... *[⋈π]*
2. $R \bowtie \pi(S, X) \rightarrow \pi(R \bowtie S, attr(R) \cup X)$ *if* $attr(R) \cap attr(S) \subset X$ ..... *[⋈π∗]*
3. $\pi(R \bowtie S, X) \rightarrow \pi(R \bowtie \pi(S, attr(S) \cap (attr(R) \cup X)), X)$ ..... *[π⋈]*
4. $\pi(R \bowtie S, X) \rightarrow \pi(R \bowtie \pi(S, attr(S) \cap X), X)$ *if* $attr(R) \cap attr(S) \subset X$ ..... *[π⋈∗]*

Proposition 19 *(join and selection)*

1. $R \bowtie \sigma(S, A) \rightarrow \sigma(R \bowtie S, A)$ ..... *[⋈σ]*
2. $\sigma(R \bowtie S, A) \rightarrow R \bowtie \sigma(S, A)$ *if* $var(A) \subset attr(R)$ ..... *[σ⋈]*

Proposition 20 *(distribution rules for union)*

1. $\pi(R \cup S, X) \rightarrow \pi(R, X) \cup \pi(S, X)$ ..... *[π∪]*
2. $\sigma(R \cup S, A) \rightarrow \sigma(R, A) \cup \sigma(S, A)$ ..... *[σ∪]*
3. $\kappa(R \cup S, \Phi) \rightarrow \kappa(R, \Phi) \cup \kappa(S, \Phi)$ ..... *[κ∪]*
4. $R \bowtie (S \cup T) \rightarrow R \bowtie S \cup R \bowtie T$ ..... *[⋈∪]*

Distribution rules for the join operator are described in the next section.

4.3 Generalized relational rules

Before we can proceed further we need rules which straightforwardly generalize rules of the previous section. However, the last generalized rule ($\kappa\bowtie\kappa$) is quite involved and crucial for the construction of a normal form that includes calculation.

Proposition 21 *(derived from* $(\bowtie\pi)$ *and* $(\bowtie\pi*)$*)*

1. $\pi(R, X) \bowtie \pi(S, Y) \rightarrow \pi(R^X \bowtie S^Y, X \cup Y)$ ..... *[π⋈π]*
2. $\pi(R, X) \bowtie \pi(S, Y) \rightarrow \pi(R \bowtie S, X \cup Y)$ *if* $attr(R) \cap attr(S) \subset X \cap Y$ ..... *[π⋈π∗]*

Proposition 22 *(derived from* $(\pi\bowtie)$ *and* $(\pi\bowtie*)$*)*

1. $\pi(R \bowtie S, X) \rightarrow \pi(\pi(R, attr(R) \cap (attr(S) \cup X)) \bowtie \pi(S, attr(S) \cap (attr(R) \cup X)), X)$ ..... *[π⋈+]*
2. $\pi(R \bowtie S, X) \rightarrow \pi(R, attr(R) \cap X) \bowtie \pi(S, attr(S) \cap X)$ *if* $attr(R) \cap attr(S) \subset X$ ..... *[π⋈#]*

Proposition 23 *(derived from* $(\bowtie\sigma)$*)* ..... *[σ⋈σ]*

$\sigma(R, A) \bowtie \sigma(S, B) \rightarrow \sigma(R \bowtie S, A \wedge B)$

The next proposition we need for derivation of ($\kappa\bowtie\kappa$). In general if both Φ and Ψ are solution sets, the expression $\Phi \cup \Psi$ might not be a solution set. One possible reason is that heads from Φ also occur as heads from Ψ. In this case the $\oplus$ operator can be used to merge the tails in Φ and the tails in Ψ together:

Proposition 24 *(introduction of the merge operator)* ..... *[κ⋈κ∗]*

$\kappa(R, \Phi) \bowtie \kappa(S, \Psi) \rightarrow \kappa(\sigma(R \bowtie S, \Phi \oplus \Psi), \Phi)$ *if* $hvar(\Phi) = hvar(\Psi)$

Proposition 25 *(generalization of* $(\bowtie\kappa)$*)* ..... *[κ⋈κ]*

$\kappa(R, \Phi) \bowtie \kappa(S, \Psi) \rightarrow$
$\kappa(\sigma(R \bowtie S,$
$\quad \Phi \oplus \Psi \wedge \Phi[attr(S)] \wedge \Psi\langle hvar(\Phi)\rangle[attr(R)]),$
$\quad \Phi\langle attr(S)\rangle \cup \Psi\langle hvar(\Phi)\rangle\langle attr(R)\rangle)$

Proof: Informally we first explain the construction. In the righthand side of ($\kappa\bowtie\kappa$) the merge operator $\oplus$ handles the case that there is a solution $x = t \in \Phi$ and a solution $y = s \in \Psi$ with $x = y$, in analogy to rule ($\kappa\bowtie\kappa*$). Also another case needs to be checked. Suppose there is a solution $x = t \in \Phi$ such that $x \in attr(S)$. If this solution were put in the computation of the calculate operator, then the resulting expression would be unwellformed. The problem can be handled by recognizing that $x = t$ now

satisfies the wellformedness conditions of the select operator, viz. $var(x = t) \subset attr(R) \cup attr(S)$. So the restriction operator inserts the solution $x = t$ in the condition of the select operator and the delete operator deletes it from the calculate computation. The symmetric case that there is a solution $x = t \in \Psi$ such that $x \in attr(R)$, is handled in the same way.
It should be noted that in the construction the expression

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

in the select condition can be replaced by the more simple expression $\Psi[attr(R)]$ since the following holds:

$$\Phi \oplus \Psi \wedge \Phi[attr(S)] \wedge \Psi\langle hvar(\Phi)\rangle[attr(R)] = \Phi \oplus \Psi \wedge \Phi[attr(S)] \wedge \Psi[attr(R)] \quad \ldots\ldots [**]$$

However this could lead to duplicate use of solutions from Ψ in the select condition and since the rule is to be used for query optimization we want to avoid this duplication.

For a formal derivation of $(\kappa\bowtie\kappa)$ we first observe:

$\Phi \oplus \Psi = \Phi[hvar(\Psi)] \oplus \Psi[hvar(\Phi)]$ ..... [*]

Now put:

$$\begin{array}{ll} \Phi_1 = \Phi[attr(S)] & \Psi_1 = \Psi[attr(R)] \\ \Phi_2 = \Phi[hvar(\Psi)] & \Psi_2 = \Psi[hvar(\Phi)] \\ \Phi_3 = \Phi\langle attr(S) \cup hvar(\Psi)\rangle & \Psi_3 = \Psi\langle attr(R) \cup hvar(\Phi)\rangle \end{array}$$

Both Φ and Ψ can be obtained as mutually disjoint unions of the above solution sets:

$$\Phi = \Phi_1 \cup \Phi_2 \cup \Phi_3, \Psi = \Psi_1 \cup \Psi_2 \cup \Psi_3$$

We have:

$\kappa(R, \Phi) \bowtie \kappa(S, \Psi)$
$\rightarrow \kappa(R, \Phi_1 \cup \Phi_2 \cup \Phi_3) \bowtie \kappa(S, \Psi_1 \cup \Psi_2 \cup \Psi_3)$
$\rightarrow \kappa(\kappa(\kappa(R, \Phi_1), \Phi_2), \Phi_3) \bowtie \kappa(\kappa(\kappa(S, \Psi_1), \Psi_2), \Psi_3)$ ..... $(\kappa\kappa*)$
$\rightarrow \kappa(\kappa(\kappa(\kappa(R, \Phi_1), \Phi_2) \bowtie \kappa(\kappa(S, \Psi_1), \Psi_2), \Psi_3), \Phi_3)$ ..... $(\bowtie\kappa*, \bowtie\kappa*)$
$\rightarrow \kappa(\kappa(\kappa(R, \Phi_1), \Phi_2) \bowtie \kappa(\kappa(S, \Psi_1), \Psi_2), \Phi_3 \cup \Psi_3)$ ..... $(\kappa\kappa*)$
$\rightarrow \kappa(\kappa(\sigma(\kappa(R, \Phi_1) \bowtie \kappa(S, \Psi_1), \Phi_2 \oplus \Psi_2), \Phi_2), \Phi_3 \cup \Psi_3)$ ..... $(\kappa\bowtie\kappa*)$
$\rightarrow \kappa(\kappa(\sigma(\kappa(R, \Phi_1) \bowtie \kappa(S, \Psi_1), \Phi \oplus \Psi), \Phi_2), \Phi_3 \cup \Psi_3)$ ..... $(*)$
$\rightarrow \kappa(\kappa(\sigma(\sigma(R \bowtie \kappa(S, \Psi_1), \Phi_1), \Phi \oplus \Psi), \Phi_2), \Phi_3 \cup \Psi_3)$ ..... $(\bowtie\kappa+)$
$\rightarrow \kappa(\kappa(\sigma(\sigma(\sigma(R \bowtie S, \Psi_1), \Phi_1), \Phi \oplus \Psi), \Phi_2), \Phi_3 \cup \Psi_3)$ ..... $(\bowtie\kappa+)$
$\rightarrow \kappa(\sigma(R \bowtie S, \Phi_1 \wedge \Psi_1 \wedge \Phi \oplus \Psi), \Phi_2 \cup \Phi_3 \cup \Psi_3)$ ..... $(\sigma\sigma, \sigma\sigma, \kappa\kappa*)$

The last expression yields $(\kappa\bowtie\kappa)$ by backsubstitution of Φ_1, Φ_2, Φ_3, Ψ_1 and Ψ_3 and application of $(**)$.
■

5 The language PCSJL

In this section we define a sublanguage **PCSJL** of **UPCSJL** together with a normal form for **PCSJL**. This normal form will be used in the next section for the construction of the **UPCSJL** normal form.

Definition 26 *The language* **PCSJL** *consists of expressions constructed from the functions:* $\bowtie, \pi, \kappa, \sigma$

Definition 27 *A* **PCSJL** *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 base relations.*

Another feasible normal form (i.e. format to which all expressions are reducible) exchanges the positions of the select and calculate operators:

$$\pi(\sigma(\kappa(R_1 \bowtie \ldots \bowtie R_n, \Phi), A), X)$$

However the disadvantage of this normal form is that unnecessary computations are performed for tuples for which the select condition evaluates to false. Derivation of our normal form below is achieved by applying four unconditional rules of the form $R \rightarrow S$ for each of the primitive four relational operators.

Proposition 28 *The expression* $\pi(\kappa(\sigma(R, A), \Phi), X)$ *is wellformed iff:*
1. $var(A) \subset attr(R)$
2. $tvar(\Phi) \subset attr(R)$
3. $hvar(\Phi) \cap attr(R) = \emptyset$
4. $X \subset attr(R) \cup hvar(\Phi)$

Proposition 29 *Consequences of wellformedness of* $\pi(\kappa(\sigma(R, A), \Phi), X)$*:*
1. $tvar(\Phi) \cap hvar(\Phi) = \emptyset$
2. $var(A) \cap hvar(\Phi) = \emptyset$
3. $hvar(\Phi) = (var(A) \cup var(\Phi)) - attr(R)$

Proposition 30 *(projection rule derived from* $(\pi\pi)$*)* ..... $[\pi\pi\kappa\sigma]$
$\pi(\pi(\kappa(\sigma(R, A), \Phi), X), Y) \rightarrow \pi(\kappa(\sigma(R, A), \Phi), Y)$

Proposition 31 *(selection rule derived from* $(\sigma\pi)$*,* $(\sigma\kappa)$ *and* $(\sigma\sigma)$*)* ..... $[\sigma\pi\kappa\sigma]$
$\sigma(\pi(\kappa(\sigma(R, A), \Phi), X), B) \rightarrow \pi(\kappa(\sigma(R, A \wedge \Phi(B)), \Phi), X)$

Proposition 32 *(calculation rule derived from* $(\kappa\pi)$ *and* $(\kappa\kappa)$*)* ..... $[\kappa\pi\kappa\sigma]$
$\kappa(\pi(\kappa(\sigma(R, A), \Phi), X), \Psi) \rightarrow \pi(\kappa(\sigma(R^X, A^X), \Phi^X \cup \Phi^X(\!(\Psi)\!)), hvar(\Psi) \cup X)$

Proposition 33 *(join rule)* ..... $[\pi\kappa\sigma\bowtie\pi\kappa\sigma]$
$\pi(\kappa(\sigma(R, A), \Phi), X) \bowtie \pi(\kappa(\sigma(S, B), \Psi), Y) \rightarrow$
$\pi(\kappa(\sigma(R^X \bowtie S^Y,$
$\quad A^X \wedge B^Y \wedge \Phi^X \oplus \Psi^Y \wedge \Phi^X[attr(S^Y)] \wedge \Psi^Y\langle hvar(\Phi^X)\rangle[attr(R^X)]),$
$\quad (\Phi^X\langle attr(S^Y)\rangle \cup \Psi^Y\langle hvar(\Phi^X)\rangle\langle attr(R^X)\rangle)[X \cup Y]\ ),$
$\quad X \cup Y)$

Proof: $\pi(\kappa(\sigma(R, A), \Phi), X) \bowtie \pi(\kappa(\sigma(S, B), \Psi), Y)$
$\rightarrow \pi(\kappa(\sigma(R^X, A^X), \Phi^X) \bowtie \kappa(\sigma(S^Y, B^Y), \Psi^Y), X \cup Y)$ ..... $(\pi\bowtie\pi)$
$\rightarrow \pi(\kappa(\sigma(\sigma(R^X, A^X) \bowtie \sigma(S^Y, B^Y),$
$\quad \Phi^X \oplus \Psi^Y \wedge \Phi^X[attr(S^Y)] \wedge \Psi^Y\langle hvar(\Phi^X)\rangle[attr(R^X)]),$
$\quad \Phi^X\langle attr(S^Y)\rangle \cup \Psi^Y\langle hvar(\Phi^X)\rangle\langle attr(R^X)\rangle), X \cup Y)$ ..... $(\kappa\bowtie\kappa)$
$\rightarrow \pi(\kappa(\sigma(\sigma(R^X \bowtie S^Y, A^X \wedge B^Y),$
$\quad \Phi^X \oplus \Psi^Y \wedge \Phi^X[attr(S^Y)] \wedge \Psi^Y\langle hvar(\Phi^X)\rangle[attr(R^X)]),$
$\quad \Phi^X\langle attr(S^Y)\rangle \cup \Psi^Y\langle hvar(\Phi^X)\rangle\langle attr(R^X)\rangle), X \cup Y)$ ..... $(\sigma\bowtie\sigma)$
$\rightarrow \pi(\kappa(\sigma(R^X \bowtie S^Y,$
$\quad A^X \wedge B^Y \wedge \Phi^X \oplus \Psi^Y \wedge \Phi^X[attr(S^Y)] \wedge \Psi^Y\langle hvar(\Phi^X)\rangle[attr(R^X)]),$
$\quad (\Phi^X\langle attr(S^Y)\rangle \cup \Psi^Y\langle hvar(\Phi^X)\rangle\langle attr(R^X)\rangle)[X \cup Y]\ ), X \cup Y)$ ..... $(\sigma\sigma,\pi\kappa)$
■

Proposition 34 *Every wellformed relational expression in* **PCSJL** *can be transformed into an equivalent wellformed* **PCSJL** *normal form.*

Proof: Basis: $R \rightarrow \pi(\kappa(\sigma(R, \mathbf{true}), \emptyset), attr(R))$.
Induction: rules $(\pi\pi\kappa\sigma)$, $(\sigma\pi\kappa\sigma)$, $(\kappa\pi\kappa\sigma)$ and $(\pi\kappa\sigma\bowtie\pi\kappa\sigma)$.
■

Note that the normal form is not unique. Especially the renaming of clashing variables is a rich source of equivalent expressions. We conclude this section with some examples of normalization.

Example 8
$\kappa(\sigma(\pi(\kappa(\sigma(r(v, w), v > w), \{x = v + w\}), \{w, x\}), x > 0), \{v = x + 2\})$
$\rightarrow \pi(\kappa(\sigma(r(u_1, w), u_1 + w > 0 \wedge u_1 > w), \{v = u_1 + w + 2\}), \{v, w, x\})$

Example 9
$\pi(\kappa(\sigma(r(w, y, v), w > y), \{x = w + y\}), \{w, x\}) \bowtie \pi(\kappa(\sigma(s(x, z, v), x > z), \{y = x + z\}), \{x, y, z\})$
$\rightarrow \pi(\kappa(\sigma(r(w, u_1, u_2) \bowtie s(x, z, u_3), w > u_1 \wedge x > z \wedge x = w + u_1), \{y = x + z\}), \{w, x, y, z\})$

Example 10
$\pi(\kappa(\sigma(r(v, w), v > w), \{x = v + w\}), \{v, w, x\}) \bowtie \pi(\kappa(\sigma(s(y, z), y > z), \{x = y + z\}), \{x, y, z\})$
$\rightarrow \pi(\kappa(\sigma(r(v, w) \bowtie s(y, z), v > w \wedge y > z \wedge v + w = y + z), \{x = v + w\}), \{v, w, x, y, z\})$

6 The language UPCSJL

The next step is the addition of the union operator, yielding the language **UPCSJL**. The normal form procedure presented in this section heavily relies on the normal form construction for **PCSJL** expressions as discussed before. By the availability of this construction, derivation of a **UPCSJL** normal form is more or less straightforward.

Definition 35 *The language* **UPCSJL** *consists of expressions constructed from the functions:* $\bowtie, \cup, \pi, \kappa, \sigma$

Definition 36 *(* **UPCSJL** *normal form)*
If $R_1, \ldots, R_n$ *are* **PCSJL** *normal forms then the following is a* **UPCSJL** *normal form:*

$$R_1 \cup \ldots \cup R_n$$

Proposition 37 *Every wellformed relational expression in* **UPCSJL** *can be transformed into an equivalent wellformed* **UPCSJL** *normal form.*

Proof: It suffices to distinguish the following six cases:
1. Basis: $R \rightarrow \pi(\kappa(\sigma(R, \mathbf{true}), \emptyset), attr(R))$.
Induction: in the remaining cases, using the construction of Proposition 34, it is assumed that R_i and S_i are reduced to **PCSJL** normal form. The expressions T_i are the resulting **PCSJL** normal forms:
2. $\pi(R_1 \cup \ldots \cup R_n, X) \rightarrow \pi(R_1, X) \cup \ldots \cup \pi(R_n, X) \rightarrow T_1 \cup \ldots \cup T_n$ ..... $(\pi\cup,\ \pi\pi\kappa\sigma)$
3. $\sigma(R_1 \cup \ldots \cup R_n, X) \rightarrow \sigma(R_1, X) \cup \ldots \cup \sigma(R_n, X) \rightarrow T_1 \cup \ldots \cup T_n$ ..... $(\sigma\cup,\ \sigma\pi\kappa\sigma)$
4. $\kappa(R_1 \cup \ldots \cup R_n, X) \rightarrow \kappa(R_1, X) \cup \ldots \cup \kappa(R_n, X) \rightarrow T_1 \cup \ldots \cup T_n$ ..... $(\kappa\cup,\ \kappa\pi\kappa\sigma)$
5. $(R_1 \cup \ldots \cup R_n) \bowtie (S_1 \cup \ldots \cup S_m) \rightarrow R_1 \bowtie S_1 \cup \ldots \cup R_n \bowtie S_m \rightarrow T_1 \cup \ldots \cup T_{n*m}$ ... $(\bowtie\cup,\ \pi\kappa\sigma\bowtie\pi\kappa\sigma)$
6. $(R_1 \cup \ldots \cup R_n) \cup (S_1 \cup \ldots \cup S_m) \rightarrow R_1 \cup \ldots \cup R_n \cup S_1 \cup \ldots \cup S_m$ ..... $(\textsc{ass}\ \circ\cup)$
Note that the rules $(\pi\cup)$, $(\sigma\cup)$, $(\kappa\cup)$, $(\bowtie\cup)$ and $(\textsc{ass}\ \circ\cup)$ were given in Section 4.2. ■

7 Query optimization of UPCSJL normal forms

Optimization of **PSJL** expressions (i.e. expressions involving projection, selection and join) is rather well understood (see the literature on query transformations, e.g. [YAN87], [ULL89]). The heuristic of performing selections and projections before joins is effective because the number and size of tuples to be joined can be reduced. If also calculations are applied before joins duplicate computations are avoided. For instance in the next example the variable v is calculated twice instead of four times if the indicated rewriting takes place:

Example 11
$r(x, y) := \{\{x = 100, y = 3\}, \{x = 200, y = 3\}\}$
$s(y, z) := \{\{y = 3, z = 10\}, \{y = 3, z = 20\}\}$
$\kappa(r(x, y) \bowtie s(y, z), v = x + 1) \rightarrow \kappa(r(x, y), v = x + 1) \bowtie s(y, z)$

A procedure for direct optimization of relational expressions, excluding the calculate operator, consists of the following steps (adapted from [ULL89]):

Algorithm 38 *(direct optimization)*

1. *Apply* $(\sigma\sigma)$ *to separate all compound select conditions:*

$$\sigma(R, A_1 \wedge \ldots \wedge A_n) \rightarrow \sigma(\ldots\sigma(R, A_1)\ldots, A_n)$$

2. *Apply* $(\sigma\sigma*)$, $(\sigma\pi)$, $(\sigma\bowtie)$ *and* $(\sigma\cup)$ *to move selection down.*
3. *Apply* $(\pi\pi)$, $(\pi\bowtie+)$, $(\pi\cup)$, $(\pi\sigma)$ *and* $(\pi attr)$ *to move projection down. Rules* $(\pi\pi)$ *and* $(\pi attr)$ *cause some projections to disappear, while rule* $(\pi\sigma)$ *splits a projection into two projections, one of which can be migrated downwards if possible.*
4. *Apply* $(\sigma\sigma)$, $(\pi\pi)$ *and* $(\sigma\pi)$ *to combine cascades of selections and projections into a single selection, a single projection or a single projection followed by a single projection.*

A disadvantage of the above algorithm is that the optimization rules are applied on the entire relational expression. An appealing prospect would therefore be first to reduce the number of relational operators, by bringing the expression in **UPCSJL** normal form. Subsequently it should be possible to apply standard optimization techniques for **PSJL** expressions on the normal form. Our aim is to give an optimization algorithm for **UPCSJL** expressions using the heuristic rules of Algorithm 38.

A necessary rewrite rule to accomplish this goal transforms a **PCSJL** normal form to an expression that contains a 'maximal' **PSJL** normal form:

Proposition 39 *(derived from $(\pi\kappa*)$)* ..... $[\pi\kappa\sigma]$
$\pi(\kappa(\sigma(R, A), \Phi), X) \rightarrow \pi(\kappa(\pi(\sigma(R, A), tvar(\Phi) \cup (attr(R) \cap X)), \Phi), X)$

In $(\pi\kappa\sigma)$ a projection operator is inserted between calculation and selection such that: 1) the computation Φ can still be performed, 2) attributes of R that are in X remain available for the outermost projection.

After application of $(\pi\kappa\sigma)$ we need to optimize the newly created **PSJL** normal form. Algorithm 38 contains the rules $(\pi\sigma)$ and $(\pi\bowtie)$ to move projection down in the relational expression. To avoid that projection is first pushed down over selection and subsequently distributed over the join operator we combine these two steps into a single rule:

Proposition 40 *(derived from $(\pi\sigma)$ and $(\pi\bowtie)$)* ..... $[\pi\sigma\bowtie]$
$\pi(\sigma(R \bowtie S, A), X) \rightarrow \pi(\sigma(R \bowtie \pi(S, attr(S) \cap (attr(R) \cup var(A) \cup X)), A), X)$

In the above rule $(\pi\sigma\bowtie)$ the expression S is projected on the union of: 1) $attr(R)$ to make sure that the joined attributes remain after projection, 2) $var(A)$ so that the condition A can be evaluated, 3) X to enforce that necessary attributes of S that are not in A remain after projection.

Once the **PSJL** expression has been processed also calculation should take part in the optimization. Calculation can be pushed down over projection, selection and join with a single rule:

Proposition 41 *(derived from $(\kappa\pi)$, $(\kappa\sigma)$ and $(\kappa\bowtie)$)* ..... $[\kappa\pi\sigma\bowtie]$
$\kappa(\pi(\sigma(R \bowtie S, A), X), \Phi) \rightarrow \pi(\sigma(R^X \bowtie \kappa(S^X, \Phi), A^X), hvar(\Phi) \cup X) \quad if \quad tvar(\Phi) \subset attr(S)$

Efficiency can be gained by generalizing $(\sigma\bowtie)$, $(\pi\sigma\bowtie)$ and $(\kappa\pi\sigma\bowtie)$ to an arbitrary number of joins, so that processing of nested joins is avoided:

Proposition 42 *(generalization of $\sigma\bowtie$)* ..... $[\sigma\bowtie\bowtie]$
$\sigma(R_1 \bowtie \ldots \bowtie R_i \bowtie \ldots \bowtie R_n, A_1 \wedge \ldots \wedge A_j \wedge \ldots \wedge A_p)$
$\rightarrow \sigma(R_1 \bowtie \ldots \bowtie \sigma(R_i, A_j) \bowtie \ldots \bowtie R_n, A_1 \wedge \ldots \wedge A_{j-1} \wedge A_{j+1} \wedge \ldots \wedge A_p)$
$\quad if \quad var(A_j) \subset attr(R_i)$

Proposition 43 *(generalization of $\pi\sigma\bowtie$)* ..... $[\pi\sigma\bowtie\bowtie]$
$\pi(\sigma(R_1 \bowtie \ldots \bowtie R_i \bowtie \ldots \bowtie R_n, A), X)$
$\rightarrow \pi(\sigma(R_1 \bowtie \ldots \bowtie$
$\quad \pi(R_i, attr(R_i) \cap (attr(R_1 \bowtie \ldots \bowtie R_{i-1} \bowtie R_{i+1} \bowtie \ldots \bowtie R_n) \cup var(A) \cup X)) \bowtie \ldots \bowtie R_n, A), X)$

Proposition 44 *(generalization of $\kappa\pi\sigma\bowtie$)* ..... $[\kappa\pi\sigma\bowtie\bowtie]$
$\kappa(\pi(\sigma(R_1 \bowtie \ldots \bowtie R_i \bowtie \ldots \bowtie R_n, A), X), \Phi_1 \cup \ldots \cup \Phi_j \cup \ldots \cup \Phi_q)$
$\rightarrow \kappa(\pi(\sigma(R_1^X \bowtie \ldots \bowtie \kappa(R_i^X, \Phi_j) \bowtie \ldots \bowtie R_n^X, A^X), hvar(\Phi_j) \cup X), \Phi_1 \cup \ldots \cup \Phi_{j-1} \cup \Phi_{j+1} \cup \ldots \cup \Phi_q)$
$\quad if \quad tvar(\Phi_j) \subset attr(R_i)$

Now we combine the above generalized rules into an optimization algorithm for **UPCSJL** (i.e. **PCSJL** expressions extended with union) that uses normalization:

Algorithm 45 *(normalization before optimization)*

1. *Bring the expression into* **UPCSJL** *normal form: $S_1 \cup \ldots \cup S_m$*
2. *Apply the following steps (a),(b),(c) on each S_j for $j = 1, \ldots m$.*
   S_j has the form $\pi(\kappa(\sigma(R_1 \bowtie \ldots \bowtie R_n, A), \Phi), X)$:
   (a) Apply $(\pi\kappa\sigma)$ yielding: $\pi(\kappa(\pi(\sigma(R_1 \bowtie \ldots \bowtie R_n, A), Y), \Phi), X)$
   (b) Optimize the subexpression $\pi(\sigma(R_1 \bowtie \ldots \bowtie R_n, A), Y)$:

i. Apply ($\sigma\sigma$) to separate select conditions yielding:

$$\pi(\sigma(R_1 \bowtie \ldots \bowtie R_n, A_1 \wedge \ldots \wedge A_p), Y)$$

ii. Apply ($\sigma\bowtie\bowtie$) at most p times to move selection down.

iii. Apply ($\pi\sigma\bowtie\bowtie$) exactly n times to move projection down.

iv. Apply ($\sigma\sigma$) to combine cascades of selections and ($\pi attr$) to eliminate superfluous projections.

(c) Optimize the subexpression $\kappa(\pi(\sigma(R'_1 \bowtie \ldots \bowtie R'_n, A'), Y), \Phi)$ resulting from the previous step:

i. Apply ($\kappa\kappa*$) to separate calculate computations yielding:

$$\kappa(\pi(\sigma(R'_1 \bowtie \ldots \bowtie R'_n, A'), Y), \Phi_1 \cup \ldots \cup \Phi_q)$$

ii. Apply ($\kappa\pi\sigma\bowtie\bowtie$) at most q times to move calculation down.

iii. Apply ($\kappa\kappa*$) to combine cascades of calculations.

8 Conclusions

In this paper we have defined **UPCSJL**, a language with expressions built up using the operations union, projection, selection, join and calculate. We have shown that a Normal Form Theorem for this language exists, using *unconditional* rules to reduce arbitrary relational expressions into normal form. In addition we showed that after normalization *conditional* optimization rules can benefit from the fixed structure of the normal form.

There are several directions for further research in this area. First of all the translation of **RL** expressions into **UPCSJL** 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

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

[DEN90a] van Denneheuvel, S. & van Emde Boas, P., *The rule language RL/1,* A. M. Tjoa & R. Wagner (eds.), Database and Expert Systems Applications, Proc. of the International Conference, Vienna, Austria, Aug 1990, Springer Verlag Wien, 381-387, (1990).

[DEN90b] van Denneheuvel, S., Kwast K. & Renardel de Lavalette G. R., *A normal form for PCSJ expressions,* Computing Science in the Netherlands, nov 1990, (to appear).

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

[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), (1986).

[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**, 78-92, (1986).

[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), 80-92, (1986).

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

Logic Group Preprint Series

Department of Philosophy
University of Utrecht
Heidelberglaan 2
3584 CS Utrecht
The Netherlands

1 C.P.J. Koymans, J.L.M. Vrancken, *Extending Process Algebra with the empty process*, September 1985
2 J.A. Bergstra, *A process creation mechanism in Process Algebra*, September 1985
3 J.A. Bergstra, *Put and get, primitives for synchronous unreliable message passing*, October 1985
4 A. Visser, *Evaluation, provably deductive equivalence in Heyting's arithmetic of substitution instances of propositional formulas*, November 1985
5 G.R. Renardel de Lavalette, *Interpolation in a fragment of intuitionistic propositional logic*, January 1986
6 C.P.J. Koymans, J.C. Mulder, *A modular approach to protocol verification using Process Algebra*, April 1986
7 D. van Dalen, F.J. de Vries, *Intuitionistic free abelian groups*, April 1986
8 F. Voorbraak, *A simplification of the completeness proofs for Guaspari and Solovay's R*, May 1986
9 H.B.M. Jonkers, C.P.J. Koymans & G.R. Renardel de Lavalette, *A semantic framework for the COLD-family of languages*, May 1986
10 G.R. Renardel de Lavalette, *Strictheidsanalyse*, May 1986
11 A. Visser, *Kunnen wij elke machine verslaan? Beschouwingen rondom Lucas' argument*, July 1986
12 E.C.W. Krabbe, *Naess's dichotomy of tenability and relevance*, June 1986
13 H. van Ditmarsch, *Abstractie in wiskunde, expertsystemen en argumentatie*, Augustus 1986
14 A. Visser, *Peano's Smart Children, a provability logical study of systems with built-in consistency* , October 1986
15 G.R. Renardel de Lavalette, *Interpolation in natural fragments of intuitionistic propositional logic*, October 1986
16 J.A. Bergstra, *Module Algebra for relational specifications*, November 1986
17 F.P.J.M. Voorbraak, *Tensed Intuitionistic Logic*, January 1987
18 J.A. Bergstra, J. Tiuryn, *Process Algebra semantics for queues*, January 1987
19 F.J. de Vries, *A functional program for the fast Fourier transform*, March 1987
20 A. Visser, *A course in bimodal provability logic*, May 1987
21 F.P.J.M. Voorbraak, *The logic of actual obligation, an alternative approach to deontic logic*, May 1987
22 E.C.W. Krabbe, *Creative reasoning in formal discussion*, June 1987
23 F.J. de Vries, *A functional program for Gaussian elimination*, September 1987
24 G.R. Renardel de Lavalette, *Interpolation in fragments of intuitionistic propositional logic*, October 1987 (revised version of no. 15)
25 F.J. de Vries, *Applications of constructive logic to sheaf constructions in toposes*, October 1987
26 F.P.J.M. Voorbraak, *Redeneren met onzekerheid in expertsystemen*, November 1987
27 P.H. Rodenburg, D.J. Hoekzema, *Specification of the fast Fourier transform algorithm as a term rewriting system*, December 1987

28 D. van Dalen, *The war of the frogs and the mice, or the crisis of the Mathematische Annalen,* December 1987
29 A. Visser, *Preliminary Notes on Interpretability Logic,* January 1988
30 D.J. Hoekzema, P.H. Rodenburg, *Gauß elimination as a term rewriting system*, January 1988
31 C. Smoryński, *Hilbert's Programme,* January 1988
32 G.R. Renardel de Lavalette, *Modularisation, Parameterisation, Interpolation,* January 1988
33 G.R. Renardel de Lavalette, *Strictness analysis for POLYREC, a language with polymorphic and recursive types*, March 1988
34 A. Visser, *A Descending Hierarchy of Reflection Principles,* April 1988
35 F.P.J.M. Voorbraak, *A computationally efficient approximation of Dempster-Shafer theory*, April 1988
36 C. Smoryński, *Arithmetic Analogues of McAloon's Unique Rosser Sentences,* April 1988
37 P.H. Rodenburg, F.J. van der Linden, *Manufacturing a cartesian closed category with exactly two objects*, May 1988
38 P.H. Rodenburg, J.L.M.Vrancken, *Parallel object-oriented term rewriting : The Booleans*, July 1988
39 D. de Jongh, L. Hendriks, G.R. Renardel de Lavalette, *Computations in fragments of intuitionistic propositional logic*, July 1988
40 A. Visser, *Interpretability Logic,* September 1988
41 M. Doorman, *The existence property in the presence of function symbols,* October 1988
42 F. Voorbraak, *On the justification of Dempster's rule of combination*, December 1988
43 A. Visser, *An inside view of EXP, or: The closed fragment of the provability logic of* $I\Delta_0+\Omega_1$, February 1989
44 D.H.J. de Jongh & A. Visser, *Explicit Fixed Points in Interpretability Logic,* March 1989
45 S. van Denneheuvel & G.R. Renardel de Lavalette, *Normalisation of database expressions involving calculations*, March 1989
46 M.F.J. Drossaers, *A Perceptron Network Theorem Prover for the Propositional Calculus*, July 1989
47 A. Visser, *The Formalization of Interpretability,* August 1989
48 J.L.M. Vrancken, *Parallel Object Oriented Term Rewriting : a first implementation in Pool2*, September 1989
49 G.R. Renardel de Lavalette, *Choice in applicative theories*, September 1989
50 C.P.J. Koymans & G.R. Renardel de Lavalette, *Inductive definitions in COLD-K*, September 1989
51 F. Voorbraak, *Conditionals, probability, and belief revision (preliminary version)*, October 1989
52 A. Visser, *On the* Σ^0_1*-Conservativity of* Σ^0_1*-Completeness,* October 1989
53 G.R. Renardel de Lavalette, *Counterexamples in applicative theories with choice*, January 1990
54 D. van Dalen, *L.E.J. Brouwer. Wiskundige en Mysticus*, June 1990
55 F. Voorbraak, *The logic of objective knowledge and rational belief,* September 1990
56 J.L.M. Vrancken, *Reflections on Parallel and Functional Languages,* September 1990
57 A. Visser, *An inside view of EXP, or: The closed fragment of the provability logic of* $I\Delta_0+\Omega_1$, revised version with new appendices, October 1990
58 S. van Denneheuvel, K. Kwast, G.R. Renardel de Lavalette, E. Spaan, *Query optimization using rewrite rules*, October 1990