Institute for Language, Logic and Information

EFFICIENT NORMALIZATION OF DATABASE AND CONSTRAINT EXPRESSIONS

Sieger van Denneheuvel
Karen Kwast

ITLI Prepublication Series
for Computation and Complexity Theory CT-90-05

University of Amsterdam

The ITLI Prepublication Series

1986
86-01 The Institute of Language, Logic and Information
86-02 Peter van Emde Boas — A Semantical Model for Integration and Modularization of Rules
86-03 Johan van Benthem — Categorial Grammar and Lambda Calculus
86-04 Reinhard Muskens — A Relational Formulation of the Theory of Types
86-05 Kenneth A. Bowen, Dick de Jongh — Some Complete Logics for Branched Time, Part I Well-founded Time, Forward looking Operators
86-06 Johan van Benthem — Logical Syntax

1987
87-01 Jeroen Groenendijk, Martin Stokhof — Type shifting Rules and the Semantics of Interrogatives
87-02 Renate Bartsch — Frame Representations and Discourse Representations
87-03 Jan Willem Klop, Roel de Vrijer — Unique Normal Forms for Lambda Calculus with Surjective Pairing
87-04 Johan van Benthem — Polyadic quantifiers
87-05 Víctor Sánchez Valencia — Traditional Logicians and de Morgan's Example
87-06 Eleonore Oversteegen — Temporal Adverbials in the Two Track Theory of Time
87-07 Johan van Benthem — Categorial Grammar and Type Theory
87-08 Renate Bartsch — The Construction of Properties under Perspectives
87-09 Herman Hendriks — Type Change in Semantics: The Scope of Quantification and Coordination

1988
Logic, Semantics and Philosophy of Language:
LP-88-01 Michiel van Lambalgen — Algorithmic Information Theory
LP-88-02 Yde Venema — Expressiveness and Completeness of an Interval Tense Logic
LP-88-03 — Year Report 1987
LP-88-04 Reinhard Muskens — Going partial in Montague Grammar
LP-88-05 Johan van Benthem — Logical Constants across Varying Types
LP-88-06 Johan van Benthem — Semantic Parallels in Natural Language and Computation
LP-88-07 Renate Bartsch — Tenses, Aspects, and their Scopes in Discourse
LP-88-08 Jeroen Groenendijk, Martin Stokhof — Context and Information in Dynamic Semantics
LP-88-09 Theo M.V. Janssen — A mathematical model for the CAT framework of Eurotra
LP-88-10 Anneke Kleppe — A Blissymbolics Translation Program

Mathematical Logic and Foundations:
ML-88-01 Jaap van Oosten — Lifschitz' Realizabiility
ML-88-02 M.D.G. Swaen — The Arithmetical Fragment of Martin Löf's Type Theories with weak Σ-elimination
ML-88-03 Dick de Jongh, Frank Veltman — Provability Logics for Relative Interpretability
ML-88-04 A.S. Troelstra — On the Early History of Intuitionistic Logic
ML-88-05 A.S. Troelstra — Remarks on Intuitionism and the Philosophy of Mathematics

Computation and Complexity Theory:
CT-88-01 Ming Li, Paul M.B.Vitanyi — Two Decades of Applied Kolmogorov Complexity
CT-88-02 Michiel H.M. Smid — General Lower Bounds for the Partitioning of Range Trees
CT-88-03 Michiel H.M. Smid, Mark H. Overmars, Leen Torenvliet, Peter van Emde Boas — Maintaining Multiple Representations of Dynamic Data Structures
CT-88-04 Dick de Jongh, Lex Hendriks, Gerard R. Renardel de Lavalette — Computations in Fragments of Intuitionistic Propositional Logic
CT-88-05 Peter van Emde Boas — Machine Models and Simulations (revised version)
CT-88-06 Michiel H.M. Smid — A Data Structure for the Union-find Problem having good Single-Operation Complexity
CT-88-07 Johan van Benthem — Time, Logic and Computation
CT-88-08 Michiel H.M. Smid, Mark H. Overmars, Leen Torenvliet, Peter van Emde Boas — Multiple Representations of Dynamic Data Structures
CT-88-09 Theo M.V. Janssen — Towards a Universal Parsing Algorithm for Functional Grammar
CT-88-10 Edith Spaan, Leen Torenvliet, Peter van Emde Boas — Nondeterminism, Fairness and a Fundamental Analogy
CT-88-11 Sieger van Denneheuvel, Peter van Emde Boas — Towards implementing RL

Other prepublications:
X-88-01 Marc Jumelet — On Solovay's Completeness Theorem

1989
Logic, Semantics and Philosophy of Language:
LP-89-01 Johan van Benthem — The Fine-Structure of Categorial Semantics
LP-89-02 Jeroen Groenendijk, Martin Stokhof — Dynamic Predicate Logic, towards a compositional, non-representational semantics of discourse
LP-89-03 Yde Venema — Two-dimensional Modal Logics for Relation Algebras and Temporal Logic of Intervals
LP-89-04 Johan van Benthem — Language in Action
LP-89-05 Johan van Benthem — Modal Logic as a Theory of Information
LP-89-06 Andreja Prijatelj — Intensional Lambek Calculi: Theory and Application
LP-89-07 Heinrich Wansing — The Adequacy Problem for Sequential Propositional Logic
LP-89-08 Víctor Sánchez Valencia — Peirce's Propositional Logic: From Algebra to Graphs
LP-89-09 Zhisheng Huang — Dependency of Belief in Distributed Systems

Mathematical Logic and Foundations:
ML-89-01 Dick de Jongh, Albert Visser — Explicit Fixed Points for Interpretability Logic
ML-89-02 Roel de Vrijer — Extending the Lambda Calculus with Surjective Pairing is conservative
ML-89-03 Dick de Jongh, Franco Montagna — Rosser Orderings and Free Variables
ML-89-04 Dick de Jongh, Marc Jumelet, Franco Montagna — On the Proof of Solovay's Theorem
ML-89-05 Rineke Verbrugge — Σ-completeness and Bounded Arithmetic
ML-89-06 Michiel van Lambalgen — The Axiomatization of Randomness
ML-89-07 Dirk Roorda — Elementary Inductive Definitions in HA: from Strictly Positive towards Monotone
ML-89-08 Dirk Roorda — Investigations into Classical Linear Logic
ML-89-09 Alessandra Carbone — Provable Fixed points in $I\Delta_0+\Omega_1$

Computation and Complexity Theory:
CT-89-01 Michiel H.M. Smid — Dynamic Deferred Data Structures
CT-89-02 Peter van Emde Boas — Machine Models and Simulations
CT-89-03 Ming Li, Herman Neuféglise, Leen Torenvliet, Peter van Emde Boas — On Space Efficient Simulations
CT-89-04 Harry Buhrman, Leen Torenvliet — A Comparison of Reductions on Nondeterministic Space
CT-89-05 Pieter H. Hartel, Michiel H.M. Smid, Leen Torenvliet, Willem G. Vree — A Parallel Functional Implementation of Range Queries
CT-89-06 H.W. Lenstra, Jr. — Finding Isomorphisms between Finite Fields
CT-89-07 Ming Li, Paul M.B. Vitanyi — A Theory of Learning Simple Concepts under Simple Distributions and Average Case Complexity for the Universal Distribution (Prel. Version)
CT-89-08 Harry Buhrman, Steven Homer, Leen Torenvliet — Honest Reductions, Completeness and Nondeterminstic Complexity Classes
CT-89-09 Harry Buhrman, Edith Spaan, Leen Torenvliet — On Adaptive Resource Bounded Computations
CT-89-10 Sieger van Denneheuvel — The Rule Language RL/1
CT-89-11 Zhisheng Huang, Sieger van Denneheuvel, Peter van Emde Boas — Towards Functional Classification of Recursive Query Processing

Other Prepublications:
X-89-01 Marianne Kalsbeek — An Orey Sentence for Predicative Arithmetic
X-89-02 G. Wagemakers — New Foundations: a Survey of Quine's Set Theory
X-89-03 A.S. Troelstra — Index of the Heyting Nachlass
X-89-04 Jeroen Groenendijk, Martin Stokhof — Dynamic Montague Grammar, a first sketch
X-89-05 Maarten de Rijke — The Modal Theory of Inequality
X-89-06 Peter van Emde Boas — Een Relationele Semantiek voor Conceptueel Modelleren: Het RL-project

1990 SEE INSIDE BACK COVER

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

EFFICIENT NORMALIZATION OF DATABASE AND CONSTRAINT EXPRESSIONS

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

ITLI Prepublication Series
for Computation and Complexity Theory
ISSN 0924-8374

Received October 1990

Efficient normalization of database and constraint expressions

Sieger van Denneheuvel

Karen Kwast

Department of Mathematics and Computer Science
University of Amsterdam

October 12, 1990

Abstract

In this paper we present a normal form for a relational algebra, consisting of **P**rojection, **S**election and **J**oin, extended with **C**alculation and **U**nion. 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.

Contents

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. We give the rules needed to derive the **PSJL** normal form in Section 5.1. 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 Section 5.2 we add the relational operator *calculate* to the above mentioned relational operators, thus obtaining the language of **PCSJL** expressions. The question arises whether **PCSJL** expressions also can be reduced into a normal form. We prove the existence of a normal form, where the joins are followed by selection, then calculate and finally projection; moreover, our proof yields a direct construction. This **PCSJL** 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). The proof and the construction for normal forms can be extended to include the union operator yielding the language **UPCSJL** in Section 5.3. 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.

The languages **PSJL**, **PCSJL** and **UPCSJL** have in common that they contain operators that can be combined efficiently into a single operator. Employing such a single operator leads to a more efficient normalization procedure. We define the corresponding languages **(PS)JL**, **(PCS)JL**, **U(PCS)JL** in Section 6.1, Section 6.2 and Section 6.3 respectively. In Section 7 we discuss optimization of normal forms.

Finally in Section 8 we introduce the language **CONSL** which allows variables that are not bound by a base relation. Our interest for **CONSL** 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 [DEN88].

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

Acknowledgements. The authors thank Gerard R. Renardel de Lavalette (University of Utrecht), Edith Spaan and Peter van Emde Boas (both University of Amsterdam) for useful criticism and remarks.

2 The data language DL

In the sequel we abbreviate the set of relational languages mentioned in the introduction as follows:

$$\mathbf{RELALG} = \{\mathbf{PSJL}, \mathbf{PCSJL}, \mathbf{UPCSJL}, \mathbf{(PS)JL}, \mathbf{(PCS)JL}, \mathbf{U(PCS)JL}, \mathbf{CONSL}\}$$

We begin with the definition of the *data language* **DL**: it will act as a parameter for each of the languages in **RELALG**. **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 languages in RELALG

Before we define the sorts of the languages in **RELALG**, 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 $\{x_1 = c_1, \ldots, x_n = c_n\}$ such that the sort of a variable x_i is the same as the sort of the constant c_i. 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. Also we have two relational constants **yes** and **no**. The former denotes the relation with no attributes consisting of a single tuple ($\{\emptyset\}$) and the latter the relation with no attributes and no tuples ($\emptyset$).

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 languages in **RELALG**. The languages in **RELALG** are four-sorted languages 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. This renaming will be used to avoid clashing of variables in application of rewrite rules.

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

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 used to define the languages in **RELALG**. 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 $var(\_) : \mathcal{C} \cup \mathcal{R} \cup \mathcal{S} \rightarrow \mathcal{V}$
$var(E) = \{x \mid x \textit{ is a variable in } E\}$

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

The attributes of relational expressions are defined inductively on the structure of relational expressions:

Definition 6 $attr(\_) : \mathcal{R} \rightarrow \mathcal{V}$
1. *$attr(R) = X$ if R is a base relation and X is the set of attributes of R.*
2. $attr(\pi(R, X)) = X$
3. $attr(\sigma(R, A)) = attr(R)$
4. $attr(\kappa(R, \Phi)) = attr(R) \cup hvar(\Phi)$
5. $attr(R \bowtie S) = attr(R) \cup attr(S)$
6. $attr(R \cup S) = attr(R)$

Put otherwise, the attributes of a relational expression are the *free variables.* The expression $attr(R)$ is only defined if R is a *wellformed relational expression*, which we will define later on.

Example 3 *(difference between variables and attributes)*
$var(\pi(\sigma(r(x,y), x > y), \{x\})) = \{x, y\}$
$attr(\pi(\sigma(r(x,y), x > y), \{x\})) = \{x\}$

The difference between $var(R)$ and $attr(R)$ will become relevant when we introduce the relational projection operator.

3.2 Functions with range $\mathcal{C}$

Besides $\wedge$ (conjunction), we have the *merge* of two solution sets, yielding a constraint. In the sequel it will be used for the construction of new constraints from the tails of two solution sets. The merge function is defined as follows:

Definition 7 $\oplus : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{C}$
$\Phi \oplus \Psi = \{s = t \mid x = s \in \Phi, x = t \in \Psi\}$

Example 4 *(merging solution sets)*
$\{x = \text{'bob'}\} \oplus \{x = y \ cat \ z\} = \{\text{'bob'} = y \ cat \ z\}$
$\{x = u + 2, y = 3\} \oplus \{x = v + 2\} = \{u + 2 = v + 2\}$
$\{x = y + 2\} \oplus \{x = z, y = 2 * z\} = \{y + 2 = z\}$
$\{x = y + 2\} \oplus \{z = x, y = 2 * x\} = \emptyset$

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 8 $\_(\_) : \mathcal{S} \times \mathcal{C} \rightarrow \mathcal{C}$
$\Phi(A) = A[x_1 := t_1, \ldots, x_n := t_n]$ *with* $\Phi = \{x_1 = t_1, \ldots, x_n = t_n\}$

Example 5 *(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 define the *restrict* and *delete* functions:

Definition 9 *(restricting and deleting solutions)*
1. $\_[\_] : \mathcal{S} \times \mathcal{V} \rightarrow \mathcal{S}$
$\Phi[X] = \{x = t \in \Phi \mid x \in X\}$
2. $\_\langle\_\rangle : \mathcal{S} \times \mathcal{V} \rightarrow \mathcal{S}$
$\Phi\langle X\rangle = \{x = t \in \Phi \mid x \notin X\}$

Instead of $\Phi\langle X\rangle$ we could also write $\Phi[\overline{X}]$. Further we also have substitution on solutions:

Definition 10 $\_(\!(\_)\!) : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$
$\Phi(\!(\Psi)\!) = \{x = eval(\Phi(t)) \mid x = t \in \Psi\} \quad$ *if* $\quad hvar(\Psi) \cap tvar(\Phi) = \emptyset$

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

Note that there is actually no need for a merge operator since it can be defined in terms of the other operators (but nevertheless we will use it as an abbreviation in the sequel):

Proposition 11 $\Phi \oplus \Psi = \Phi[hvar(\Psi)](\Psi[hvar(\Phi)])$

The proposition can be explained in the following way. In the expression $\Phi(\Psi[hvar(\Phi)])$ the solutions in Ψ that have heads in Φ are subject to the substitution Φ. As a consequence the head variables in Ψ are replaced by the corresponding terms in Φ as required. This is allright except that now also the tail variables in Ψ may be subject to substitution which is not what we want. The problem is circumvented by replacing the substitution Φ by $\Phi[hvar(\Psi)]$ which yields the defining expression for the merge operator. We then have:

$$\begin{array}{l} hvar(\Phi[hvar(\Psi)]) \subset hvar(\Psi) \ \& \ tvar(\Psi) \cap hvar(\Psi) = \emptyset \\ \quad \Rightarrow \quad tvar(\Psi) \cap hvar(\Phi[hvar(\Psi)]) = \emptyset \end{array}$$

3.4 Functions with range $\mathcal{R}$

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

Definition 12 *(primitive relational operators)*
1. $\pi : \mathcal{R} \times \mathcal{V} \to \mathcal{R}$
$\pi(R, X) = \{\phi[X] \mid \phi \in R\} \quad \textit{if} \quad X \subset \textit{attr}(R)$
2. $\sigma : \mathcal{R} \times \mathcal{C} \to \mathcal{R}$
$\sigma(R, A) = \{\phi \in R \mid \textit{eval}(\phi(A)) = \textbf{true}\} \quad \textit{if} \quad \textit{var}(A) \subset \textit{attr}(R)$
3. $\kappa : \mathcal{R} \times \mathcal{S} \to \mathcal{R}$
$\kappa(R, \Phi) = \{\psi \cup \psi((\Phi)) \mid \psi \in R\} \quad \textit{if} \quad \textit{tvar}(\Phi) \subset \textit{attr}(R) \textit{ and } \textit{hvar}(\Phi) \cap \textit{attr}(R) = \emptyset$
4. $\_ \bowtie \_ : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$
$R \bowtie S = \{\phi \cup \psi \mid \phi \in R, \psi \in S, \forall x \in \textit{attr}(\phi) \cap \textit{attr}(\psi) \ (\phi[x] = \psi[x])\}$
5. $\_ \cup \_ : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$
$R \cup S = \{\phi \mid \phi \in R \vee \phi \in S\} \quad \textit{if} \quad \textit{attr}(R) = \textit{attr}(S)$

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 *instruction* of the calculate operator.

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

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

Definition 13 $\rho : \mathcal{R} \times \mathcal{S} \to \mathcal{R}$
$\rho(R, \Phi) = \pi(\kappa(R, \Phi), \textit{attr}(R) \cup \textit{hvar}(\Phi) - \textit{tvar}(\Phi))$
$\quad \textit{if} \quad \textit{tvar}(\Phi) \subset \textit{attr}(R) \textit{ and } \textit{hvar}(\Phi) \cap \textit{attr}(R) = \emptyset$

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

Another use of the calculate operator is in the creation of relations that have attributes but no tuples. The calculate operator can extend the empty set with arbitrary attributes:

Example 9 *(an empty relation with attributes 'x' and 'y')*
$\kappa(\emptyset, \{x = 1, y = \text{'bob'}\}) = \emptyset$

Cartesian product and intersection of relational expressions can be defined directly in terms of the join operator:

Definition 14 $\times : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$
$R \times S = R \bowtie S \quad \textit{if} \quad \textit{attr}(R) \cap \textit{attr}(S) = \emptyset$

Definition 15 $\cap : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$
$R \cap S = R \bowtie S \quad \textit{if} \quad \textit{attr}(R) = \textit{attr}(S)$

In the sequel we also need an operator that given a set of variables X, creates a relation consisting of the cartesian product of the domains of the variables in X:

Definition 16 $\mathcal{D}(\_) : \mathcal{V} \to \mathcal{R}$
$\mathcal{D}(X) = \{\phi \mid hvar(\phi) = X\}$

Since ϕ is a tuple, in the above definition it is ensured that variables of sort **NUM** are assigned all values of the numeric domain and variables of sort **STR** the values of the string domain.

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

For some rewrite rules $R \to S$, wellformedness of R does not imply wellformedness of S and as a consequence the rule is conditional. In that case the wellformedness conditions on the violating subexpressions are represented in the rewrite rule as a condition. Note that the list given below is incomplete, but this is unavoidable.

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 will cluster rules that are similar in one proposition if possible.

Proposition 18 *(constant reductions)*
1. $\sigma(R, \mathbf{true}) \leftrightarrow R$ .... *[*σ**true***]*
2. $\kappa(R, \emptyset) \leftrightarrow R$ .... *[*$\kappa\emptyset$*]*
3. $\pi(R, attr(R)) \leftrightarrow R$ .... *[*$\pi attr$*]*
4. $R \bowtie \mathbf{yes} \leftrightarrow R$ .... *[*$\bowtie$**yes***]*

Proposition 19 *(cascade rules)*
1. $\pi(\pi(R, X), Y) \to \pi(R, Y)$ .... *[*$\pi\pi$*]*
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. $\kappa(\kappa(R, \Phi), \Psi) \to \kappa(R, \Phi \cup \Phi((\Psi)))$ .... *[*$\kappa\kappa$*]*
5. $\kappa(\kappa(R, \Phi), \Psi) \leftrightarrow \kappa(R, \Phi \cup \Psi)$ *if* $hvar(\Phi) \cap tvar(\Psi) = \emptyset$ .... *[*$\kappa\kappa*$*]*

Proposition 20 *(selection and projection)*
1. $\sigma(\pi(R, X), A) \to \pi(\sigma(R, A), X)$ .... *[*$\sigma\pi$*]*
2. $\pi(\sigma(R, A), X) \to \pi(\sigma(\pi(R, attr(R) \cap (var(A) \cup X)), A), X)$ .... *[*$\pi\sigma$*]*
3. $\pi(\sigma(R, A), X) \to \sigma(\pi(R, X), A)$ *if* $var(A) \subset X$ .... *[*$\pi\sigma*$*]*

Proposition 21 *(selection and calculation)*
1. $\sigma(\kappa(R, \Phi), A) \to \kappa(\sigma(R, \Phi(A)), \Phi)$ .... *[*$\sigma\kappa$*]*
2. $\kappa(\sigma(R, A), \Phi) \to \sigma(\kappa(R, \Phi), A)$ .... *[*$\kappa\sigma$*]*

Proposition 22 *(calculation and projection)*
1. $\kappa(\pi(R, X), \Phi) \to \pi(\kappa(R^X, \Phi), hvar(\Phi) \cup X)$ .... *[*$\kappa\pi$*]*
2. $\kappa(\pi(R, X), \Phi) \to \pi(\kappa(R, \Phi), hvar(\Phi) \cup X)$ *if* $attr(R) \cap hvar(\Phi) = \emptyset$ .... *[*$\kappa\pi*$*]*
3. $\pi(\kappa(R, \Phi), X) \to \pi(\kappa(R, \Phi[X]), X)$ .... *[*$\pi\kappa$*]*
4. $\pi(\kappa(R, \Phi), X) \to \pi(\kappa(\pi(R, tvar(\Phi) \cup (X \cap attr(R))), \Phi), X)$ .... *[*$\pi\kappa*$*]*
5. $\pi(\kappa(R, \Phi), X) \to \kappa(\pi(R, X), \Phi[X])$ *if* $tvar(\Phi[X]) \subset X$ .... *[*$\pi\kappa+$*]*

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. To illustrate how the rules for the join operator subsume the rules for cartesian product, we have also listed the latter ([ULL89]), if appropriate.

Proposition 23 *(symmetry and associativity of $\bowtie$)*
1. $R \bowtie S \leftrightarrow S \bowtie R$ .... [SYMo $\bowtie$]
2. $(R \bowtie S) \bowtie T \leftrightarrow R \bowtie (S \bowtie T)$ .... [ASSo$\bowtie$]

Proposition 24 *(symmetry and associativity of $\cup$)*
1. $R \cup S \leftrightarrow S \cup R$ .... [SYM $\circ \cup$]
2. $(R \cup S) \cup T \leftrightarrow R \cup (S \cup T)$ .... [ASS $\circ\cup$]

Proposition 25 *(join and calculation)*
1. $R \bowtie \kappa(S, \Phi) \rightarrow \kappa(\sigma(R \bowtie S, \Phi[attr(R)]), \Phi\langle attr(R)\rangle)$ .... [$\bowtie\kappa$]
2. $R \bowtie \kappa(S, \Phi) \rightarrow \kappa(R \bowtie S, \Phi)$ *if* $hvar(\Phi) \cap attr(R) = \emptyset$ .... [$\bowtie\kappa*$]
3. $R \bowtie \kappa(S, \Phi) \rightarrow \sigma(R \bowtie S, \Phi)$ *if* $hvar(\Phi) \subset attr(R)$ .... [$\bowtie\kappa+$]
4. $\kappa(R \bowtie S, \Phi) \rightarrow R \bowtie \kappa(S, \Phi)$ *if* $tvar(\Phi) \subset attr(S)$ .... [$\kappa\bowtie$]

Proposition 26 *(join and projection)*
1. $R \bowtie \pi(S, X) \rightarrow \pi(R \bowtie S^X, attr(R) \cup X)$ .... [$\bowtie\pi$]
2. $R \bowtie \pi(S, X) \rightarrow \pi(R \bowtie S, attr(R) \cup X)$ *if* $attr(R) \cap attr(S) \subset X$ .... [$\bowtie\pi*$]
3. $R \times \pi(S, X) \rightarrow \pi(R \times S, attr(R) \cup X)$ .... [$\times\pi*$]
4. $\pi(R \bowtie S, X) \rightarrow \pi(R \bowtie \pi(S, attr(S) \cap (attr(R) \cup X)), X)$ .... [$\pi\bowtie$]
5. $\pi(R \bowtie S, X) \rightarrow \pi(R \bowtie \pi(S, attr(S) \cap X), X)$ *if* $attr(R) \cap attr(S) \subset X$ .... [$\pi\bowtie*$]
6. $\pi(R \times S, X) \rightarrow \pi(R \times \pi(S, attr(S) \cap X), X)$ .... [$\pi\times*$]

Proposition 27 *(join and selection)*
1. $R \bowtie \sigma(S, A) \rightarrow \sigma(R \bowtie S, A)$ .... [$\bowtie\sigma$]
2. $\sigma(R \bowtie S, A) \rightarrow R \bowtie \sigma(S, A)$ *if* $var(A) \subset attr(R)$ .... [$\sigma\bowtie$]

Proposition 28 *(union)*
1. $\pi(R \cup S, X) \rightarrow \pi(R, X) \cup \pi(S, X)$ .... [$\pi\cup$]
2. $\sigma(R \cup S, A) \rightarrow \sigma(R, A) \cup \sigma(S, A)$ .... [$\sigma\cup$]
3. $\kappa(R \cup S, \Phi) \rightarrow \kappa(R, \Phi) \cup \kappa(S, \Phi)$ .... [$\kappa\cup$]
4. $R \bowtie (S \cup T) \rightarrow R \bowtie S \cup R \bowtie T$ .... [$\bowtie\cup$]
5. $\pi(R, X) \cup \pi(S, X) \rightarrow \pi(R \cup S, X)$ .... [$\pi\cup\pi$]
6. $\sigma(R, A) \cup \sigma(S, A) \rightarrow \sigma(R \cup S, A)$ .... [$\sigma\cup\sigma$]
7. $\kappa(R, \Phi) \cup \kappa(S, \Phi) \rightarrow \kappa(R \cup S, \Phi)$ .... [$\kappa\cup\kappa$]
8. $R \bowtie S \cup R \bowtie T \rightarrow R \bowtie (S \cup T)$ .... [$\bowtie\cup\bowtie$]

In the above union rules we deliberately maintained two versions for each of the operators π, σ, κ and $\bowtie$, so that from the rule label the structure of the rule can be easily inferred.

4.3 Generalized relational rules

Before we can proceed further we need rules which straightforwardly generalize rules of the previous section. However, the last rule ($\kappa\bowtie\kappa$) in this section is quite involved but crucial for the proofs of rules to come.

Proposition 29 *(derived from ($\bowtie\pi$) and ($\bowtie\pi*$))*
1. $\pi(R, X) \bowtie \pi(S, Y) \rightarrow \pi(R^X \bowtie S^Y, X \cup Y)$ .... [$\pi\bowtie\pi$]
2. $\pi(R, X) \bowtie \pi(S, Y) \rightarrow \pi(R \bowtie S, X \cup Y)$ *if* $attr(R) \cap attr(S) \subset X \cap Y$ .... [$\pi\bowtie\pi*$]

Proposition 30 *(derived from ($\pi\bowtie$) and ($\pi\bowtie*$))*
1. $\pi(R \bowtie S, X) \rightarrow \pi(\pi(R, attr(R) \cap (attr(S) \cup X)) \bowtie \pi(S, attr(S) \cap (attr(R) \cup X)), X)$ .... [$\pi\bowtie+$]
2. $\pi(R \bowtie S, X) \rightarrow \pi(R, attr(R) \cap X) \bowtie \pi(S, attr(S) \cap X)$ *if* $attr(R) \cap attr(S) \subset X$ .... [$\pi\bowtie\#$]
3. $\pi(R \times S, X) \rightarrow \pi(R, attr(R) \cap X) \times \pi(S, attr(S) \cap X)$ .... [$\pi\times\#$]

Proposition 31 *(derived from ($\bowtie\sigma$))* .... *[$\sigma\bowtie\sigma$]*
$\sigma(R, A) \bowtie \sigma(S, B) \rightarrow \sigma(R \bowtie S, A \wedge B)$

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 32 *(introduction of the merge operator)* .... *[$\kappa\bowtie\kappa*$]*
$\kappa(R, \Phi) \bowtie \kappa(S, \Psi) \rightarrow \kappa(\sigma(R \bowtie S, \Phi \oplus \Psi), \Phi) \quad \textit{if} \quad hvar(\Phi) = hvar(\Psi)$

Proposition 33 *(generalization of ($\bowtie\kappa$))* .... *[$\kappa\bowtie\kappa$]*
$\kappa(R, \Phi) \bowtie \kappa(S, \Psi) \rightarrow$
$\kappa(\sigma(R \bowtie S,$
$\quad \Phi \oplus \Psi \wedge \Phi[attr(S)] \wedge \Psi\langle hvar(\Phi)\rangle[attr(R)]),$
$\quad \Phi\langle attr(S)\rangle \cup \Psi\langle hvar(\Phi)\rangle\langle attr(R)\rangle)$

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 instruction 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 instruction. The symmetric case that there is a solution $x = t \in \Psi$ such that $x \in attr(R)$, is handled in the same way.

More precisely the 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 condition of the select operator; 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 instruction of the calculate operator.
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:

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

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 languages PSJL, PCSJL and UPCSJL

In the next three sections we give normal forms for each of the languages **PSJL**, **PCSJL** and **UPCSJL**. The language **PSJL** is the first in this series of increasingly more expressive languages. The second and the third add the calculate operator and the union operator respectively to **PSJL**. Both **PSJL** and **PCSJL** are just intermediate steps towards the language **UPCSJL**.

5.1 The language PSJL

In the construction of a normal form for **PSJL** we employ the rules ($\pi\pi\sigma$), ($\sigma\pi\sigma$) and ($\pi\sigma\bowtie\pi\sigma$) presented in this section. For a more detailed description of the construction we refer to Section 5.2 which contains a similar construction.

Definition 34 *The language* **PSJL** *consists of expressions constructed from the functions:*
$\bowtie,\pi,\sigma$

Proposition 35 *The expression* $\pi(\sigma(R,A),X)$ *is wellformed iff:*
1. $var(A) \subset attr(R)$
2. $X \subset attr(R)$

Proof: Directly from the wellformedness conditions of the individual operators. ■

Proposition 36 *(projection rule)* .... [$\pi\pi\sigma$]
$\pi(\pi(\sigma(R,A),X),Y) \rightarrow \pi(\sigma(R,A),Y)$

Proof: Immediate from ($\pi\pi$). ■

Proposition 37 *(selection rule)* .... [$\sigma\pi\sigma$]
$\sigma(\pi(\sigma(R,A),X),B) \rightarrow \pi(\sigma(R,A \wedge B),X)$

Proof: Immediate from ($\sigma\pi$) and ($\sigma\sigma$). ■

Proposition 38 *(join rule)* .... [$\pi\sigma\bowtie\pi\sigma$]
$\pi(\sigma(R,A),X) \bowtie \pi(\sigma(S,B),Y) \rightarrow \pi(\sigma(R^X \bowtie S^Y, A^X \wedge B^Y), X \cup Y)$

Proof: $\pi(\sigma(R,A),X) \bowtie \pi(\sigma(S,B),Y)$
$\rightarrow \pi(\sigma(R^X,A^X) \bowtie \sigma(S^Y,B^Y),X \cup Y)$ .... ($\pi\bowtie\pi$)
$\rightarrow \pi(\sigma(R^X \bowtie S^Y, A^X \wedge B^Y),X \cup Y)$ .... ($\sigma\bowtie\sigma$)
■

Now we describe the normal form for **PSJL** expressions. The normal form is quite simple, so we will postpone a more extensive discussion of normal forms and their construction to Section 5.2.

Definition 39 *(* **PSJL** *normal form)*
If $R_1,\ldots,R_n$ *are base relations then the following is a* **PSJL** *normal form:*

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

Proposition 40 *Every wellformed relational expression in* **PSJL** *can be transformed into an equivalent wellformed* **PSJL** *normal form.*

Proof: Directly from ($\pi\pi\sigma$), ($\sigma\pi\sigma$), ($\pi\sigma\bowtie\pi\sigma$) and $R \rightarrow \pi(\sigma(R,\mathbf{true}),attr(R))$. ■

5.2 Adding calculation: PCSJL

In this section we expand the previous propositions by adding the calculate operator. The results that are presented in this section subsume all results of the previous section. The calculate operator adds some complexity to the rules $(\pi\pi\sigma)$, $(\sigma\pi\sigma)$ and $(\pi\sigma\bowtie\pi\sigma)$, resulting in the modified rules $(\pi\pi\kappa\sigma)$, $(\sigma\pi\kappa\sigma)$ and $(\pi\kappa\sigma\bowtie\pi\kappa\sigma)$ respectively. Moreover we need a completely new rule $(\kappa\pi\kappa\sigma)$ to be used for induction over the calculate operator.

Definition 41 *The language* **PCSJL** *consists of expressions constructed from the functions:* $\bowtie,\pi,\kappa,\sigma$

Definition 42 *(* **PCSJL** *normal form)*
If $R_1,\ldots,R_n$ *are base relations then the following is a* **PCSJL** *normal form:*

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

Proposition 43 *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. $X \subset attr(R) \cup hvar(\Phi)$

Proof: Directly from the wellformedness conditions of the individual operators. ■

Proposition 44 *Consequences of wellformedness of* $\pi(\kappa(\sigma(R, A), \Phi), X)$*:*
1. $tvar(\Phi) \cap hvar(\Phi) = \emptyset$
2. $var(A) \cap hvar(\Phi) = \emptyset$

Proof: Follows directly from Proposition 43.
■

Another feasible normal form 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. Since these rules are unconditional they are suitable for obtaining a normal form. We now derive the four rules:

Proposition 45 *(projection rule)* .... *[ππκσ]*
$\pi(\pi(\kappa(\sigma(R, A), \Phi), X), Y) \rightarrow \pi(\kappa(\sigma(R, A), \Phi), Y)$

Proof: Immediate from $(\pi\pi)$. ■

Proposition 46 *(selection rule)* .... *[σπκσ]*
$\sigma(\pi(\kappa(\sigma(R, A), \Phi), X), B) \rightarrow \pi(\kappa(\sigma(R, A \wedge \Phi(B)), \Phi), X)$

Proof: $\sigma(\pi(\kappa(\sigma(R, A), \Phi), X), B)$
$\rightarrow \pi(\sigma(\kappa(\sigma(R, A), \Phi), B), X)$ .... $(\sigma\pi)$
$\rightarrow \pi(\kappa(\sigma(\sigma(R, A), \Phi(B)), \Phi), X)$ .... $(\sigma\kappa)$
$\rightarrow \pi(\kappa(\sigma(R, A \wedge \Phi(B)), \Phi), X)$ .... $(\sigma\sigma)$
■

Proposition 47 *(calculation rule)* .... *[κπκσ]*
$\kappa(\pi(\kappa(\sigma(R, A), \Phi), X), \Psi) \rightarrow \pi(\kappa(\sigma(R^X, A^X), \Phi^X \cup \Phi^X((\Psi))), hvar(\Psi) \cup X)$

Proof: $\kappa(\pi(\kappa(\sigma(R, A), \Phi), X), \Psi)$
$\rightarrow \pi(\kappa(\kappa(\sigma(R^X, A^X), \Phi^X), \Psi), hvar(\Psi) \cup X)$ .... $(\kappa\pi)$
$\rightarrow \pi(\kappa(\sigma(R^X, A^X), \Phi^X \cup \Phi^X((\Psi))), hvar(\Psi) \cup X)$ .... $(\kappa\kappa)$
■

Proposition 48 *(join rule)* .... *[$\pi\kappa\sigma\bowtie\pi\kappa\sigma$]*
$\pi(\kappa(\sigma(R, A), \Phi), X) \bowtie \pi(\kappa(\sigma(S, B), \Psi), Y) \rightarrow$
$\pi(\kappa(\sigma(R^X \bowtie S^Y,$
$\quad A^X \wedge B^Y \wedge \Phi^X \oplus \Psi^Y \wedge \Phi^X[attr(S^Y)] \wedge \Psi^Y\langle hvar(\Phi^X)\rangle[attr(R^X)]),$
$\quad \Phi^X\langle attr(S^Y)\rangle \cup \Psi^Y\langle hvar(\Phi^X)\rangle\langle attr(R^X)\rangle\ ),$
$\quad X \cup Y)$

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

Proposition 49 *Every wellformed relational expression in* **PCSJL** *can be transformed into an equivalent wellformed* **PCSJL** *normal form.*

Proof: It suffices to show the following five statements:
1. Basis: $R \rightarrow \pi(\kappa(\sigma(R, \mathbf{true}), \emptyset), attr(R))$
In the remaining cases we use the following abbreviations:

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

For each case we first list the relational expression to be normalized into **PCSJL**.
2. Projection: $\pi(\pi(\kappa(\sigma(R, A), \Phi), X), Y)$
Application of ($\pi\pi\kappa\sigma$).
3. Selection: $\sigma(\pi(\kappa(\sigma(R, A), \Phi), X), B)$
Application of ($\sigma\pi\kappa\sigma$).
4. Calculation: $\kappa(\pi(\kappa(\sigma(R, A), \Phi), X), \Psi)$
Application of ($\kappa\pi\kappa\sigma$).
5. Join: $\pi(\kappa(\sigma(R, A), \Phi), X) \bowtie \pi(\kappa(\sigma(S, B), \Psi), Y)$
Application of ($\pi\kappa\sigma\bowtie\pi\kappa\sigma$).
■

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.

Example 10
$\kappa(\sigma(\pi(\kappa(\sigma(r(v, w), v > w), \{x = v + w\}), \{w, x\}), x > 0), \{v = x + 2\})$
$\rightarrow \pi(\kappa(\sigma(r(u_1, w), u_1 + w > 0 \wedge u_1 > w), \{v = u_1 + w + 2\}), \{v, w, x\})$

Example 11
$\pi(\kappa(\sigma(r(v, w), v > w), \{x = v + w\}), \{v, w, x\})$
$\quad \bowtie \pi(\kappa(\sigma(s(x, z), x > z), \{y = x + z\}), \{x, z, y\})$
$\rightarrow \pi(\kappa(\sigma(r(v, w) \bowtie s(x, z), v > w \wedge x > z \wedge x = v + v), \{y = x + z\}), \{v, w, x, y, z\})$

Example 12
$\pi(\kappa(\sigma(r(w, y, v), w > y), \{x = w + y\}), \{w, x\})$
$\quad \bowtie \pi(\kappa(\sigma(s(x, z, v), x > z), \{y = x + z\}), \{x, y, z\})$
$\rightarrow \pi(\kappa(\sigma(r(w, u_1, u_2) \bowtie s(x, z, u_3), w > u_1 \wedge x > z \wedge x = w + u_1), \{y = x + z\}), \{w, x, y, z\})$

Example 13
$\pi(\kappa(\sigma(r(v,w), v > w), \{x = v + w\}), \{v, w, x\})$
$\quad \bowtie \pi(\kappa(\sigma(s(z), z > 0), \{y = z + 1\}), \{z, y\})$
$\rightarrow \pi(\kappa(\sigma(r(v,w) \bowtie s(z), v > w \wedge z > 0), \{x = v + w, y = z + 1\}), \{v, w, x, y, z\})$

Example 14
$\pi(\kappa(\sigma(r(v,w), v > w), \{x = v + w\}), \{v, w, x\})$
$\quad \bowtie \pi(\kappa(\sigma(s(y,z), y > z), \{x = y + z\}), \{x, y, z\})$
$\rightarrow \pi(\kappa(\sigma(r(v,w) \bowtie s(y,z), v > w \wedge y > z \wedge v + w = y + z), \{x = v + w\}), \{v, w, x, y, z\})$

5.3 Adding union: 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 50 *The language* **UPCSJL** *consists of expressions constructed from the functions:* $\bowtie, \cup, \pi, \kappa, \sigma$

Definition 51 *(* **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 52 *Every wellformed relational expression in* **UPCSJL** *can be transformed into an equivalent wellformed* **UPCSJL** *normal form.*

Proof: It suffices to show the following six statements:
1. Basis: $R \rightarrow \pi(\kappa(\sigma(R, \mathbf{true}), \emptyset), attr(R))$.
In the remaining cases, using the construction of Proposition 49, it may be assumed that:

$$R_i \text{ is in } \mathbf{PCSJL} \text{ normal form for } i = 1, \ldots, n$$
$$S_j \text{ is in } \mathbf{PCSJL} \text{ normal form for } j = 1, \ldots, m$$

The expressions T_i are the resulting **PCSJL** normal forms:
2. Projection: By repeated application of the rules ($\pi\cup$) and ($\pi\pi\kappa\sigma$) we have:
$\pi(R_1 \cup \ldots \cup R_n, X) \rightarrow \pi(R_1, X) \cup \ldots \cup \pi(R_n, X) \rightarrow T_1 \cup \ldots \cup T_n$
3. Selection: By repeated application of the rules ($\sigma\cup$) and ($\sigma\pi\kappa\sigma$) we have:
$\sigma(R_1 \cup \ldots \cup R_n, X) \rightarrow \sigma(R_1, X) \cup \ldots \cup \sigma(R_n, X) \rightarrow T_1 \cup \ldots \cup T_n$
4. Calculation: By repeated application of the rules ($\kappa\cup$) and ($\kappa\pi\kappa\sigma$) we have:
$\kappa(R_1 \cup \ldots \cup R_n, X) \rightarrow \kappa(R_1, X) \cup \ldots \cup \kappa(R_n, X) \rightarrow T_1 \cup \ldots \cup T_n$
5. Join: By repeated application of the rule ($\bowtie\cup$) and ($\pi\kappa\sigma\bowtie\pi\kappa\sigma$) we have:
$(R_1 \cup \ldots \cup R_n) \bowtie (S_1 \cup \ldots \cup S_m) \rightarrow R_1 \bowtie S_1 \cup \ldots \cup R_n \bowtie S_m \rightarrow T_1 \cup \ldots \cup T_{n*m}$
6. Union: By repeated application of (ASS $\circ\cup$) we have:
$(R_1 \cup \ldots \cup R_n) \cup (S_1 \cup \ldots \cup S_m) \rightarrow R_1 \cup \ldots \cup R_n \cup S_1 \cup \ldots \cup S_m$
■

6 The languages (PS)JL, (PCS)JL and U(PCS)JL

Our approach in this section is similar to the one in Section 5 except that in the following sections we consider combinations of relational operators. The aim of combining relational operators is that the normalization process becomes more efficient.

6.1 Combining projection and selection: (PS)JL

A natural question that arises in the context of relational expressions is whether it is fruitful to combine two or more operators into a single relational operator. In this section we merge the projection and the selection operator and as we shall see such a combination is not merely a convenient abbreviation but also yields a more efficient normal form procedure. More precisely the number of applications of the induction hypothesis is reduced, since a combination of a projection and a selection now only takes one induction step instead of two.

Definition 53 $\tau : \mathcal{R} \times \mathcal{C} \times \mathcal{V} \rightarrow \mathcal{R}$
$\tau(R, A, X) := \pi(\sigma(R, A), X)$

Proposition 54 *The expression $\tau(R, A, X)$ is wellformed iff:*
1. $var(A) \subset attr(R)$
2. $X \subset attr(R)$

Proof: Directly from the wellformedness conditions of the defining operators. ■

Definition 55 *The language* **(PS)JL** *consists of expressions constructed from the functions:*
$\bowtie, \tau$

Now that we have combined projection and selection, we also need new rules that only refer to the τ and join operators. Using these rules the normal form for **(PS)JL** can be constructed by straightforward induction.

Proposition 56 *(join rule)*.. *[$\tau\bowtie\tau$]*
$\tau(R, A, X) \bowtie \tau(S, B, Y) \rightarrow \tau(R^X \bowtie S^Y, A^X \wedge B^Y, X \cup Y)$

Proof: Immediate from $(\pi\sigma\bowtie\pi\sigma)$. ■

Proposition 57 *(cascade rule)*.. *[$\tau\tau$]*
$\tau(\tau(R, A, X), B, Y) \rightarrow \tau(R, A \wedge B, Y)$

Proof: $\tau(\tau(R, A, X), B, Y)$
$\rightarrow \pi(\sigma(\tau(R, A, X), B), Y)$
$\rightarrow \pi(\tau(R, A \wedge B, X), Y)$ .... $(\sigma\pi\sigma)$
$\rightarrow \tau(R, A \wedge B, Y)$ .... $(\pi\pi\sigma)$
■

Definition 58 *(* **(PS)JL** *normal form)*
If $R_1, \ldots, R_n$ are base relations then the following is a **(PS)JL** *normal form:*

$$\tau(R_1 \bowtie \ldots \bowtie R_n, A, X)$$

Proposition 59 *Every wellformed relational expression in* **(PS)JL** *can be transformed into an equivalent wellformed* **(PS)JL** *normal form.*

Proof: Induction using $(\tau\tau)$, $(\tau\bowtie\tau)$ and $R \rightarrow \tau(R, \mathbf{true}, attr(R))$.
■

Although the previous construction showed some improvement on the number of invocations of the induction, we still have that each join operator takes a separate induction step. This can be avoided if induction is applied on the combination of the τ and the join operator instead of induction on τ and join individually. Since some order in the application of the join and the τ operators is assumed we need the following definition:

Definition 60 *(* **(PS)JL** *tree)*
1. If R is a base relation then R is a **(PS)JL** *tree.*
2. If $R_1, \ldots, R_n$ are **(PS)JL** *trees then $\tau(R_1 \bowtie \ldots \bowtie R_n, A, X)$ is a* **(PS)JL** *tree.*

We are now in a position to state the next slightly modified normal form proposition:

Proposition 61 *Every wellformed* **(PS)JL** *tree can be transformed into an equivalent wellformed* **(PS)JL** *normal form.*

Proof:
Basis: $R \rightarrow \tau(R, \mathbf{true}, attr(R))$.
Induction over τ,$\bowtie$ combinations:
Suppose R is of the following form:

$$\tau(R_1 \bowtie \ldots \bowtie R_n, A, X)$$

By induction we know that the R_i are in **(PS)JL** normal form. Now we are faced with the problem of bringing this particular expression into **(PS)JL** normal form.
Normalization is achieved by taking the following steps:

1. $(\tau\bowtie\tau)$:
   Merge $R_1 \bowtie \ldots \bowtie R_n$ into one **(PS)JL** normal form using the rule $(\tau\bowtie\tau)$ various times. This yields an expression
   $$\tau(\tau(R'_1 \bowtie \ldots \bowtie R'_p, B, Y), A, X)$$
   where the R'_i are base relations and B and Y are newly created.

2. $(\tau\tau)$:
   Applying the rule $(\tau\tau)$ yields the following expression:
   $$\tau(R'_1 \bowtie \ldots \bowtie R'_p, A \wedge B, X)$$
   This expression is a **(PS)JL** normal form.

■
Note that any wellformed relational **(PS)JL** expression can be easily transformed to a **(PS)JL** tree.

6.2 Adding calculation: (PCS)JL

Addition of the calculate operator to the language **(PS)JL** yields the language **(PCS)JL** discussed in this section. The rules $(\tau\tau)$ and $(\tau\bowtie\tau)$ are generalized to $(\delta\delta)$ and $(\delta\bowtie\delta)$. With these rules, similar efficient normal form constructions can be obtained as in Section 6.1.

Definition 62 $\delta : \mathcal{R} \times \mathcal{C} \times \mathcal{S} \times \mathcal{V} \rightarrow \mathcal{R}$
$\delta(R, A, \Phi, X) := \pi(\kappa(\sigma(R, A), \Phi), X)$

Proposition 63 *The expression* $\delta(R, A, \Phi, X)$ *is wellformed iff:*
1. $var(A) \subset attr(R)$
2. $tvar(\Phi) \subset attr(R)$
3. $hvar(\Phi) \cap attr(R) = \emptyset$
4. $X \subset attr(R) \cup hvar(\Phi)$

Proof: Directly from the wellformedness conditions of the defining operators. ■

Definition 64 *The language* **(PCS)JL** *consists of expressions constructed from the functions:* $\bowtie$,δ

Proposition 65 *(join rule)* .... *[$\delta\bowtie\delta$]*
$\delta(R, A, \Phi, X) \bowtie \delta(S, B, \Psi, Y) \rightarrow$
$\delta(R^X \bowtie S^Y,$
$\quad A^X \wedge B^Y \wedge \Phi^X \oplus \Psi^Y \wedge \Phi^X[attr(S^Y)] \wedge \Psi^Y\langle hvar(\Phi^X)\rangle[attr(R^X)],$
$\quad \Phi^X\langle attr(S^Y)\rangle \cup \Psi^Y\langle hvar(\Phi^X)\rangle\langle attr(R^X)\rangle,$
$\quad X \cup Y)$

Proof: Immediate from $(\pi\kappa\sigma\bowtie\pi\kappa\sigma)$. ■

Proposition 66 *(cascade rule)* .... *[$\delta\delta$]*
$\delta(\delta(R, A, \Phi, X), B, \Psi, Y) \rightarrow \delta(R^X, A^X \wedge \Phi^X(B), \Phi^X \cup \Phi^X(\Psi), Y)$

Proof: $\delta(\delta(R, A, \Phi, X), B, \Psi, Y)$
$\rightarrow \pi(\kappa(\sigma(\delta(R, A, \Phi, X), B), \Psi), Y)$
$\rightarrow \pi(\kappa(\delta(R, A \wedge \Phi(B), \Phi, X), \Psi), Y)$ .... $(\sigma\pi\kappa\sigma)$
$\rightarrow \pi(\delta(R^X, A^X \wedge \Phi^X(B), \Phi^X \cup \Phi^X(\Psi), hvar(\Psi) \cup X), Y)$ .... $(\kappa\pi\kappa\sigma)$
$\rightarrow \delta(R^X, A^X \wedge \Phi^X(B), \Phi^X \cup \Phi^X(\Psi), Y)$ .... $(\pi\pi\kappa\sigma)$
Note that in the above construction the condition B is not renamed. From the wellformedness conditions we know that $var(B) \subset X$. Therefore renaming B with respect to X in the application of the $(\kappa\pi\kappa\sigma)$ would not introduce any changes and the renaming is omitted. ■

Definition 67 *(* **(PCS)JL** *normal form)*
If $R_1, \ldots, R_n$ are base relations then the following is a **(PCS)JL** *normal form:*

$$\delta(R_1 \bowtie \ldots \bowtie R_n, A, \Phi, X)$$

Proposition 68 *Every wellformed relational expression in* **(PCS)JL** *can be transformed into an equivalent wellformed* **(PCS)JL** *normal form.*

Proof: Induction using $(\delta\delta)$, $(\delta\bowtie\delta)$ and $R \rightarrow \delta(R, \mathbf{true}, \emptyset, attr(R))$. ■

Definition 69 *(* **(PCS)JL** *tree)*
1. If R is a base relation then R is a **(PCS)JL** *tree.*
2. If $R_1, \ldots, R_n$ are **(PCS)JL** *trees then $\delta(R_1 \bowtie \ldots \bowtie R_n, A, X,)$ is a* **(PCS)JL** *tree.*

Proposition 70 *Every wellformed* **(PCS)JL** *tree can be transformed into an equivalent wellformed* **(PCS)JL** *normal form.*

Proof:
Basis: $R \rightarrow \delta(R, \mathbf{true}, \emptyset, attr(R))$.
Induction: Suppose R is of the following form:

$$\delta(R_1 \bowtie \ldots \bowtie R_n, B, \Psi, Y)$$

By induction we know that the R_i are in **(PCS)JL** normal form. Now we are faced with the problem of bringing this particular expression into **(PCS)JL** normal form.
Normalization is achieved by taking the following steps:

1. $(\delta\bowtie\delta)$:
   Merge $R_1 \bowtie \ldots \bowtie R_n$ into one **(PCS)JL** normal form using the rule $(\delta\bowtie\delta)$ various times. This yields an expression
   $$\delta(\delta(R'_1 \bowtie \ldots \bowtie R'_p, A, \Phi, X), B, \Psi, Y)$$
   where R'_i are base relations and A, Φ and X are newly created.

2. $(\delta\delta)$:
   Applying the rule $(\delta\delta)$ yields the following:
   $$\delta(R''_1 \bowtie \ldots \bowtie R''_p, C, \Theta, Y)$$
   This expression is a **(PCS)JL** normal form.

■

6.3 Adding union: U(PCS)JL

Finally in this section we define a language that combines the virtues of all five preceding relational languages. Using the cascade rule $(\delta\delta)$, the join rule $(\delta\bowtie\delta)$ and the union rule $(\delta\cup)$ a normal form can be constructed for **U(PCS)JL**.

Definition 71 *The language* **U(PCS)JL** *consists of expressions constructed from the functions:*
$\bowtie, \cup, \delta$

Proposition 72 *Any expression in the language* **UPCSJL** *can be transformed into an equivalent expression in the language* **U(PCS)JL**.

Proof: Simple traversal of the **UPCSJL** expressions using:
1. $\pi(R, X) = \delta(R, \mathbf{true}, \emptyset, X)$
2. $\sigma(R, X) = \delta(R, A, \emptyset, attr(R))$
3. $\kappa(R, \Phi) = \delta(R, \mathbf{true}, \Phi, attr(R) \cup hvar(\Phi))$

■

In order to extend the normal form construction with the union operator we need the following rules:

Proposition 73 *(union rule)*
1. $\pi(\kappa(\sigma(R \cup S, A), \Phi), X) \rightarrow \pi(\kappa(\sigma(R, A), \Phi), X) \cup \pi(\kappa(\sigma(S, A), \Phi), X)$ .... *[$\pi\kappa\sigma\cup$]*
2. $\delta(R \cup S, A, \Phi, X) \rightarrow \delta(R, A, \Phi, X) \cup \delta(S, A, \Phi, X)$ .... *[$\delta\cup$]*

Proof:
1. Immediate from the rules ($\sigma\cup$), ($\kappa\cup$) and ($\pi\cup$).
2. Immediate from (1).

■

Definition 74 *(* **U(PCS)JL** *normal form)*
If $R_1, \ldots, R_n$ *are* **(PCS)JL** *normal forms then the following is a* **U(PCS)JL** *normal form:*

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

Proposition 75 *Every wellformed relational expression in* **U(PCS)JL** *can be transformed into an equivalent wellformed* **U(PCS)JL** *normal form.*

Proof: Induction using ($\delta\cup$), ($\delta\delta$), ($\delta\bowtie\delta$) and $R \rightarrow \delta(R, \mathbf{true}, \emptyset, attr(R))$.

■

In analogy to the languages **(PS)JL** and **(PCS)JL** also for **UPCSJL** an efficient induction scheme can be devised. The construction generalizes the construction of Proposition 70.

Definition 76 *(* **U(PCS)JL** *tree)*
1. If R *is a base relation then* R *is a* **U(PCS)JL** *tree.*
2. If R_{ij} *are* **U(PCS)JL** *trees then the following is a* **U(PCS)JL** *tree:*

$$\delta(R_{11} \bowtie \ldots \bowtie R_{1n}, A_1, \Phi_1, X) \cup \ldots \cup \delta(R_{m1} \bowtie \ldots \bowtie R_{mp}, A_m, \Phi_m, X)$$

Proposition 77 *Every wellformed* **U(PCS)JL** *tree can be transformed into an equivalent wellformed* **U(PCS)JL** *normal form.*

Proof: Basis: $R \rightarrow \delta(R, \mathbf{true}, \emptyset, attr(R))$.
Induction: Suppose R is of the following form:

$$\delta(R_{11} \bowtie \ldots \bowtie R_{1n}, A_1, \Phi_1, X) \cup \ldots \cup \delta(R_{m1} \bowtie \ldots \bowtie R_{mn}, A_m, \Phi_m, X)$$

By induction we know that the R_{ij} are in **U(PCS)JL** normal form. For clarity we assume that there are $m \times n$ normal forms R_{ij} in the above expression. If this is not the case then some base relations **yes** can be added without loss of generality. Now we are faced with the problem of bringing this particular expression into **U(PCS)JL** normal form.

The problem is split up in bringing each $\delta(R_{i1} \bowtie \ldots \bowtie R_{in}, A_i, \Phi_i, X)$ into **U(PCS)JL** normal form. After that is completed, the union of these **U(PCS)JL** normal forms is again a **U(PCS)JL** normal form. Normalization of each expression $\delta(R_{i1} \bowtie \ldots \bowtie R_{in}, A_i, \Phi_i, X)$ is achieved as follows:

1. ($\bowtie\cup$), (ASS $\circ\cup$), (ASS$\circ\bowtie$):
   Distribute $\cup$ over $\bowtie$. This yields the following expression

   $$\delta(R_{11} \bowtie \ldots \bowtie R_{1n} \cup \ldots \cup R_{p1} \bowtie \ldots \bowtie R_{pn}, A, \Phi, X)$$

   where R_{ij} are **(PCS)JL** normal forms. Note that $attr(R_{11} \bowtie \ldots \bowtie R_{1n}) = \ldots = attr(R_{p1} \bowtie \ldots \bowtie R_{pn})$ as required.

2. ($\delta\cup$):
   Next we bring the unions outwards using the rule ($\delta\cup$) various times. This yields the following expression:
$$\delta(R_{11} \bowtie \ldots \bowtie R_{1n}, A, \Phi, X) \cup \ldots \cup \delta(R_{p1} \bowtie \ldots \bowtie R_{pn}, A, \Phi, X)$$

3. ($\delta\bowtie\delta$):
   Next we merge each $R_{k1} \bowtie \ldots \bowtie R_{kn}$ into one **(PCS)JL** normal form (using the rule ($\delta\bowtie\delta$) n-1 times for each $k = 1, \ldots, p$). This yields an expression
$$\delta(R'_1, A, \Phi, X) \cup \ldots \cup \delta(R'_p, A, \Phi, X)$$
   where $R'_1, \ldots, R'_p$ are the newly constructed **(PCS)JL** normal forms.
   Note that $attr(R'_1) = \ldots = attr(R'_p)$.

4. ($\delta\delta$):
   Applying p times the rule ($\delta\delta$) yields the following expression
$$R''_1 \cup \ldots \cup R''_p$$
   where $R''_1, \ldots, R''_p$ are the newly constructed **(PCS)JL** normal forms. This expression is a **U(PCS)JL** normal form.

■

7 Query optimization of UPCSJL normal forms

In the previous sections we gave constructions for obtaining **UPCSJL** and **U(PCS)JL** normal forms. Transforming a relational expression into its normal form was achieved by application of unconditional rewrite rules. Once a relational expression is in normal form however, *conditional* rewrite rules are applied to further optimize the expression. In this section we describe how this optimization can be achieved. Since **U(PCS)JL** is nothing but a syntactical variety of **UPCSJL** we will only consider the latter in the forthcoming discussion.

Optimization of **PSJL** expressions is well understood in literature on query transformations. 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. A procedure for direct optimization of relational expressions, including the union operator, consists of the following steps (freely adapted from [ULL89]):

Algorithm 78 *(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 optimization techniques for **PSJL** expressions, such as Algorithm 78, on the normal form. The next proposition allows us to do so by rewriting a **PCSJL** normal form to an expression that contains a **PSJL** normal form:

Proposition 79 .... *[πκσ]*
$\pi(\kappa(\sigma(R, A), \Phi), X) \rightarrow \pi(\kappa(\pi(\sigma(R, A), tvar(\Phi) \cup (attr(R) \cap X)), \Phi[X]), X)$

Proof: Immediate from $(\pi\kappa*)$ and $(\pi\kappa)$.
■

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.

Algorithm 80 *(normalization before optimization)*

1. *Bring the expression into* **UPCSJL** *normal form:* $S_1 \cup \ldots \cup S_m$
2. *Apply the following steps on* 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* $\pi(\sigma(R_1 \bowtie \ldots \bowtie R_n, A), Y)$ *with Algorithm 78.*

Note that Algorithm 80 includes the calculation operator. The algorithm improves on the previous one since optimization is applied on a normal form. However there still is some inefficiency in the application of Algorithm 78 inside Algorithm 80 because projection is first pushed down over selection and subsequently distributed over the join operator. The next rule combines these two steps into a single rule:

Proposition 81 .... *[πσ⋈]*
$\pi(\sigma(R \bowtie S, A), X) \rightarrow \pi(\sigma(R \bowtie \pi(S, attr(S) \cap (attr(R) \cup var(A) \cup X)), A), X)$

Proof: $\pi(\sigma(R \bowtie S, A), X)$
$\rightarrow \pi(\sigma(\pi(R \bowtie S, attr(R \bowtie S) \cap (var(A) \cup X)), A), X)$ .... $(\pi\sigma)$
$\rightarrow \pi(\sigma(R \bowtie \pi(S, attr(S) \cap (attr(R) \cup var(A) \cup X)), A), X)$ .... $(\pi\bowtie)$
■

In the above rule 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.

Also calculation can be pushed down over projection, selection and join with a single rule:

Proposition 82 .... *[κπσ⋈]*
$\kappa(\pi(\sigma(R \bowtie S, A), X), \Phi) \rightarrow \pi(\sigma(R^X \bowtie \kappa(S^X, \Phi), A^X), hvar(\Phi) \cup X)$
if $tvar(\Phi) \subset attr(S)$

Proof: $\kappa(\pi(\sigma(R \bowtie S, A), X), \Phi)$
$\rightarrow \pi(\kappa(\sigma(R^X \bowtie S^X, A^X), \Phi), hvar(\Phi) \cup X)$ .... $(\kappa\pi)$
$\rightarrow \pi(\sigma(\kappa(R^X \bowtie S^X, \Phi), A^X), hvar(\Phi) \cup X)$ .... $(\kappa\sigma)$
$\rightarrow \pi(\sigma(R^X \bowtie \kappa(S^X, \Phi), A^X), hvar(\Phi) \cup X)$ .... $(\kappa\bowtie)$
■

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

Proposition 84 *(generalization of πσ⋈)* .... *[πσ⋈⋈]*
$\pi(\sigma(R_1 \bowtie \ldots \bowtie R_i \bowtie \ldots \bowtie R_n, A), X)$
$\rightarrow \pi(\sigma(R_1 \bowtie \ldots \bowtie$
$\pi(R_i, attr(R_i) \cap (attr(R_1 \bowtie \ldots \bowtie R_{i-1} \bowtie R_{i+1} \bowtie \ldots \bowtie R_n) \cup var(A) \cup X))$
$\bowtie \ldots \bowtie R_n, A), X)$

Proposition 85 *(generalization of $\kappa\pi\sigma\bowtie$)* .... *[$\kappa\pi\sigma\bowtie\bowtie$]*
$\kappa(\pi(\sigma(R_1 \bowtie \ldots \bowtie R_i \bowtie \ldots \bowtie R_n, A), X), \Phi_1 \cup \ldots \cup \Phi_j \cup \ldots \cup \Phi_q)$
$\rightarrow \kappa(\pi(\sigma(R_1^X \bowtie \ldots \bowtie$
$\quad \kappa(R_i^X, \Phi_j)$
$\quad \bowtie \ldots \bowtie R_n^X, A^X), hvar(\Phi_j) \cup X), \Phi_1 \cup \ldots \cup \Phi_{j-1} \cup \Phi_{j+1} \cup \ldots \cup \Phi_q)$
$\quad$ *if* $tvar(\Phi_j) \subset attr(R_i)$

Now we combine the above generalized rules into an optimization algorithm that uses normalization for **UPCSJL**:

Algorithm 86 *(normalization before optimization)*

1. *Bring the expression into* **UPCSJL** *normal form:* $S_1 \cup \ldots \cup S_m$
2. *Apply the following steps on* 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 The language CONSL

In the language **CONSL** calculation, selection and projection are combined into a single operator χ. For this reason χ resembles the δ operator defined in Section 5.2 but contrary to the δ operator, χ merges the computation of the calculate operator and the condition of the selection operator into a single constraint set A. Expressive power is not lost however, since calculations can be represented as constraints in A. In fact, with the χ operator it is possible to create an infinite relation from a finite one, which could not be achieved by the δ operator due to its strict wellformedness conditions.

Definition 87 $\chi : \mathcal{R} \times \mathcal{C} \times \mathcal{V} \rightarrow \mathcal{R}$
$\chi(R, A, X) := \tau(R \bowtie \mathcal{D}(u_1, \ldots, u_n), A, X)$
$\quad$ *where* $\{u_1, \ldots, u_n\} = var(A) - attr(R)$

Definition 88 *A variable x is bound by a relational expression R if* $x \in attr(R)$.

The definition of χ is valid because in Section 5.1 we did not restrict the relational argument R in $\tau(R, A, X)$ to finite relations only. As an immediate consequence in $\chi(R, A, X)$ we allow in the constraint set A variables that are not bound by R (which was not the case for the τ operator). However projection variables that do not occur anywhere in the expression are not permitted:

Proposition 89 *The expression* $\chi(R, A, X)$ *is wellformed iff* $X \subset attr(R) \cup var(A)$.

Definition 90 *The language* **CONSL** *consists of expressions constructed from the functions:*
$\bowtie, \cup, \chi$

Proposition 91 *(join rule)* .... *[χ⋈χ]*
$\chi(R, A, X) \bowtie \chi(S, B, Y) \rightarrow \chi(R^X \bowtie S^Y, A^X \wedge B^Y, X \cup Y)$

Proof:
$\chi(R, A, X) \bowtie \chi(S, B, Y)$
$\rightarrow \tau(R \bowtie \mathcal{D}(u_1, \ldots, u_n), A, X) \bowtie \tau(S \bowtie \mathcal{D}(v_1, \ldots, v_m), B, Y)$
$\rightarrow \tau(R^X \bowtie S^Y \bowtie \mathcal{D}(u_1^X, \ldots, u_n^X, v_1^Y, \ldots, v_m^Y), A^X \wedge B^Y, X \cup Y)$ .... (τ⋈τ)
$\rightarrow \tau(R^X \bowtie S^Y \bowtie \mathcal{D}(z_1, \ldots, z_k), A^X \wedge B^Y, X \cup Y)$ .... (∗)
$\rightarrow \chi(R^X \bowtie S^Y, A^X \wedge B^Y, X \cup Y)$
As justification for step (∗) we have to construct a set $Z = \{z_1, \ldots z_k\}$ such that:

$$Z = var(A^X \wedge B^Y) - attr(R^X \bowtie S^Y)$$

But consider the set of variables W constructed from the variables in A^X not bound by R^X and the variables in B^Y not bound by S^Y:

$$W = \{u_1^X, \ldots, u_n^X, v_1^Y, \ldots, v_m^Y\}$$

We claim that $Z \subset W$ since in the construction no bound variables become unbound. Now the set Z is obtained by deleting from W all variables that become bound. ■

Proposition 92 *(cascade rule)* .... *[χχ]*
$\chi(\chi(R, A, X), B, Y) \rightarrow \chi(R^X, A^X \wedge B, Y)$

Proof: Let $V = \{v_1, \ldots, v_m\} = var(B) - X$.
$\chi(\chi(R, A, X), B, Y)$
$\rightarrow \chi(\tau(R \bowtie \mathcal{D}(u_1, \ldots, u_n), A, X), B, Y)$
$\rightarrow \tau(\tau(R \bowtie \mathcal{D}(u_1, \ldots, u_n), A, X) \bowtie \mathcal{D}(v_1, \ldots, v_m), B, Y)$
$\rightarrow \tau(\tau(R \bowtie \mathcal{D}(u_1, \ldots, u_n), A, X) \bowtie \tau(\mathcal{D}(v_1, \ldots, v_m), \mathbf{true}, V), B, Y)$
$\rightarrow \tau(\tau(R^X \bowtie \mathcal{D}(u_1^X, \ldots, u_n^X) \bowtie \mathcal{D}(v_1, \ldots, v_m), A^X, X \cup V), B, Y)$ .... (τ⋈τ)
$\rightarrow \tau(R^X \bowtie \mathcal{D}(u_1^X, \ldots, u_n^X) \bowtie \mathcal{D}(v_1, \ldots, v_m), A^X \wedge B, Y)$ .... (ττ)
$\rightarrow \tau(R^X \bowtie \mathcal{D}(z_1, \ldots, z_k), A^X \wedge B, Y)$ .... (∗)
$\rightarrow \chi(R^X, A^X \wedge B, Y)$
In the step (τ⋈τ) the variables $V = \{v_1, \ldots, v_m\}$ are not renamed because $V \cap X = \emptyset$ and renaming with respect to X was already applied on R and $u_1^X, \ldots, u_n^X$ and so no clashes can occur.
As justification for step (∗) we have to construct a set $Z = \{z_1, \ldots z_k\}$ such that:

$$Z = var(A^X \wedge B) - attr(R^X)$$

But consider the set of variables W:

$$W = \{u_1^X, \ldots, u_n^X, v_1, \ldots, v_m\}$$

We claim that $Z = W$.
■
Before we can give a union rule for the χ operator we need the next proposition:

Proposition 93 *(union rule)*
1. $\pi(\sigma(R \cup S, A), X) \rightarrow \pi(\sigma(R, A), X) \cup \pi(\sigma(S, A), X)$ .... *[πσ∪]*
2. $\tau(R \cup S, A, X) \rightarrow \tau(R, A, X) \cup \tau(S, A, X)$ .... *[τ∪]*

Proof:
1. Immediate from the rules (σ∪) and (π∪).
2. Immediate from (1).
■

Proposition 94 *(union rule)* .... *[χ∪]*
$\chi(R \cup S, A, X) \rightarrow \chi(R, A, X) \cup \chi(S, A, X)$

Proof: Let $\{u_1, \ldots, u_n\} = var(A) - attr(R) = var(A) - attr(S)$.
$\chi(R \cup S, A, X)$
$\rightarrow \tau((R \cup S) \bowtie \mathcal{D}(u_1, \ldots, u_n), A, X)$
$\rightarrow \tau(R \bowtie \mathcal{D}(u_1, \ldots, u_n) \cup S \bowtie \mathcal{D}(u_1, \ldots, u_n), A, X)$ .... $(\bowtie\cup)$
$\rightarrow \tau(R \bowtie \mathcal{D}(u_1, \ldots, u_n), A, X) \cup \tau(S \bowtie \mathcal{D}(u_1, \ldots, u_n), A, X)$ .... $(\tau\cup)$
$\rightarrow \chi(R, A, X) \cup \chi(S, A, Y)$
■

Definition 95 *(* **CONSL** *normal form)*
If R_{ij} are base relations then the following is a **CONSL** *normal form:*

$$\chi(R_{11} \bowtie \ldots \bowtie R_{1n}, A_1, X) \cup \ldots \cup \chi(R_{m1} \bowtie \ldots \bowtie R_{mp}, A_m, X)$$

Proposition 96 *Every wellformed relational expression in* **CONSL** *can be transformed into an equivalent wellformed* **CONSL** *normal form.*

Proof: Induction using $(\chi\chi)$, $(\chi\bowtie\chi)$, $(\chi\cup)$ and $R \rightarrow \chi(R, \textbf{true}, attr(R))$. ■

Definition 97 *(* **CONSL** *tree)*
1. If R is a base relation then R is a **CONSL** *tree.*
2. If R_{ij} are **CONSL** *trees then the following is a* **CONSL** *tree:*

$$\chi(R_{11} \bowtie \ldots \bowtie R_{1n}, A_1, X) \cup \ldots \cup \chi(R_{m1} \bowtie \ldots \bowtie R_{mp}, A_m, X)$$

Proposition 98 *Every wellformed* **CONSL** *tree can be transformed into an equivalent wellformed* **CONSL** *normal form.*

Proof: Similar to the construction of Proposition 77. ■

9 Conclusions

In this paper we have defined a series of languages {**PSJL**, **PCSJL**, **UPCSJL**}. We have shown that a Normal Form Theorem for each of these languages exists, by giving a construction to transform arbitrary relational expressions into normal form. Since renaming for attributes can be defined directly in terms of calculation and projection, it is included in the normalization construction.

Subsequently we introduced the languages {(**PS**)**JL**, (**PCS**)**JL**, **U**(**PCS**)**JL**} to obtain a normal form more efficiently by combining relational operators if possible. In addition we changed the normal form construction to further reduce the number of applications of the induction hypothesis.

There are several directions for further research in this area. First of all the translation of **RL** expressions into **PCSJL** expressions, using **CONSL** as an intermediate language, is to be worked out: 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.

Index

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.

[DEN88] van Denneheuvel, S. & van Emde Boas, P., *Constraint solving for databases,* Proc. of NAIC **1**, Apr. 1988

[DEN89] van Denneheuvel, S. & Renardel de Lavalette, G. R., *Normalization 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)

The ITLI Prepublication Series

1990

Logic, Semantics and Philosophy of Language	
LP-90-01 Jaap van der Does	A Generalized Quantifier Logic for Naked Infinitives
LP-90-02 Jeroen Groenendijk, Martin Stokhof	Dynamic Montague Grammar
LP-90-03 Renate Bartsch	Concept Formation and Concept Composition
LP-90-04 Aarne Ranta	Intuitionistic Categorial Grammar
LP-90-05 Patrick Blackburn	Nominal Tense Logic
LP-90-06 Gennaro Chierchia	The Variablity of Impersonal Subjects
LP-90-07 Gennaro Chierchia	Anaphora and Dynamic Logic
LP-90-08 Herman Hendriks	Flexible Montague Grammar
LP-90-09 Paul Dekker	The Scope of Negation in Discourse, towards a flexible dynamic Montague grammar
LP-90-10 Theo M.V. Janssen	Models for Discourse Markers
LP-90-11 Johan van Benthem	General Dynamics
LP-90-12 Serge Lapierre	A Functional Partial Semantics for Intensional Logic
LP-90-13 Zisheng Huang	Logics for Belief Dependence
Mathematical Logic and Foundations	
ML-90-01 Harold Schellinx	Isomorphisms and Non-Isomorphisms of Graph Models
ML-90-02 Jaap van Oosten	A Semantical Proof of De Jongh's Theorem
ML-90-03 Yde Venema	Relational Games
ML-90-04 Maarten de Rijke	Unary Interpretability Logic
ML-90-05 Domenico Zambella	Sequences with Simple Initial Segments
ML-90-06 Jaap van Oosten	Extension of Lifschitz' Realizability to Higher Order Arithmetic, and a Solution to a Problem of F. Richman
ML-90-07 Maarten de Rijke	A Note on the Interpretability Logic of Finitely Axiomatized Theories
ML-90-08 Harold Schellinx	Some Syntactical Observations on Linear Logic
ML-90-09 Dick de Jongh, Duccio Pianigiani	Solution of a Problem of David Guaspari
Computation and Complexity Theory	
CT-90-01 John Tromp, Peter van Emde Boas	Associative Storage Modification Machines
CT-90-02 Sieger van Denneheuvel, Gerard R. Renardel de Lavalette	A Normal Form for PCSJ Expressions
CT-90-03 Ricard Gavaldà, Leen Torenvliet, Osamu Watanabe, José L. Balcázar	Generalized Kolmogorov Complexity in Relativized Separations
CT-90-04 Harry Buhrman, Leen Torenvliet	Bounded Reductions
CT-90-05 Sieger van Denneheuvel, Karen Kwast	Efficient Normalization of Database and Constraint Expressions
Other Prepublications	
X-90-01 A.S. Troelstra	Remarks on Intuitionism and the Philosophy of Mathematics, Revised Version
X-90-02 Maarten de Rijke	Some Chapters on Interpretability Logic
X-90-03 L.D. Beklemishev	On the Complexity of Arithmetical Interpretations of Modal Formulae
X-90-04	Annual Report 1989
X-90-05 Valentin Shehtman	Derived Sets in Euclidean Spaces and Modal Logic
X-90-06 Valentin Goranko, Solomon Passy	Using the Universal Modality: Gains and Questions
X-90-07 V.Yu. Shavrukov	The Lindenbaum Fixed Point Algebra is Undecidable
X-90-08 L.D. Beklemishev	Provability Logics for Natural Turing Progressions of Arithmetical Theories
X-90-09 V.Yu. Shavrukov	On Rosser's Provability Predicate
X-90-10 Sieger van Denneheuvel, Peter van Emde Boas	An Overview of the Rule Language RL/1
X-90-11 Alessandra Carbone	Provable Fixed points in $I\Delta_0+\Omega_1$, revised version
X-90-12 Maarten de Rijke	Bi-Unary Interpretability Logic