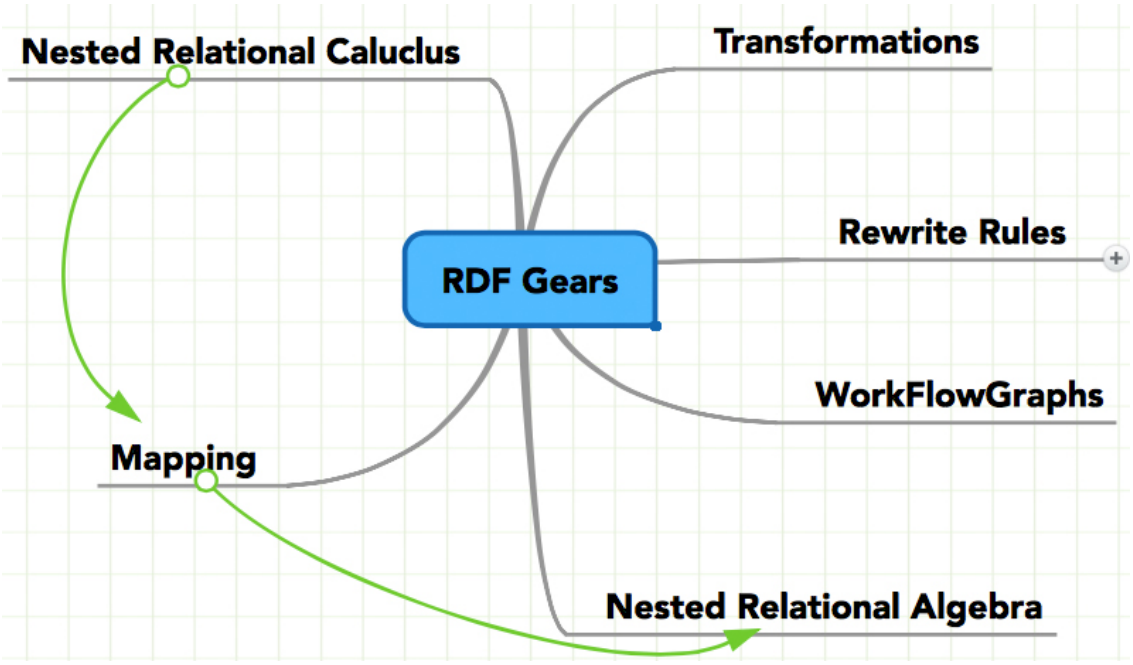
On the Equivalence
of
Nested Relational Languages

GARGI PRASAD



On the Equivalence of Nested Relational Languages

Gargi Prasad

Department of Software and Computer Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

This dissertation is submitted for the degree of

Master of Science in Computer Science

Delft University of Technology 2014

To my grandfather,

Dr. Rameshwar Prasad

3rd January 1919 - 4th February 2014

Author: Gargi Prasad
Title: ***On the Equivalence of Nested Relational Languages***
Program: Master of Science in Computer Science.

Thesis Committee:

Chair: prof. dr. ir. G.J.P.M. Houben, Faculty EEMCS, TUDelft
University supervisor: dr. ir. A.J.H. Hidders, Faculty EEMCS, TUDelft
Committee Member: prof. dr. Erik Meijer, TUDelft & Microsoft (Redmond, USA)

©2014, Gargi Prasad.

Acknowledgements

The time spent on this thesis has been the one of the most insightful and technically challenging times in my career so far. It definitely had its high and low points, but became an important part of my life. It not only enhanced my knowledge on the subject matter, but also gave me an experience of research and taught me resilience. It gives me the satisfaction of delving deep into concepts of theoretical computer science, especially databases.
This work would not have been possible without the support and guidance of Prof. J. Hidders. I will be ever grateful for his mentorship and patience. The insightful discussions that I had with him served as constant motivation and reaffirmation that I could continue working on this highly technical subject.
I would also like to acknowledge here the unconditional support of my family, who have encouraged me to work hard every single day. I would also like to thank Anurag, for his constant companionship and incredible support during the course of this thesis.
This thesis marks the conclusion of my Master Program at TUDelft, an important chapter in my life, during which I lost and found significantly but the lessons I learnt during this journey will stay with me forever and inspire me to challenge myself each day with the same spirit.

Abstract

Query optimization has played a vital role in database research since the 1970's and up till now. In the relational context, a prerequisite to query optimization is the study of equivalences over relational algebra expressions. Sophisticated techniques for query transformations for traditional query languages include rewriting based optimizations. Rewriting optimization transforms a query into equivalent one and can have a significant impact on execution time. Though RDF Gears is an inspiration to this work, we aim to develop rewriting based optimization strategies that can be applied to similar query languages like PigLatin and Hive.

We study the various rewriting based optimizations that currently exist and identify many of the existing ideas that can be extrapolated and applied to RDF Gears. We define the syntax and semantics of Nested Relational Algebra(NRA) and Nested Relational Calculus(NRC) and use a mapping between them to transform expressions of one to the other. We come up with a full set of NRA & NRC, an exhaustive list of rewrite rules, which cover all possible rewrites needed for our language. We mathematically find a minimum core set, that we call the concise or the non-derivable set of NRA & NRC and prove their derivability either mathematically or using a theorem prover.

We also present a mathematical proof for equivalence relationship between the rule sets of NRA & NRC. The proof obtained for NRA & NRC that is comprised of basic values, pairs and tuples, for one free variable.

Contents

List of Figures

Nomenclature

Notation

Γ	Binding
$\mathscr{B}$	Set of all basic values
$\mathscr{C}$	Set of Collections
$\mathscr{P}$	Set of Pairs
$\mathscr{V}$	Set of all values
f	NRA function
g	NRA function
h	NRA function
$\lambda_{\uplus}$	Additive Bag Union Function
$\uplus$	Additive Bag Union
$:=$	Assignment
$\{\{e_1 \mid x \, in \, e_2\}\}$	Bag Comprehension
λ_c	Constant Function
set	Set Function
$\{\{\}\}$	Empty Bag
$\lambda_{\{\{\}\}}$	Empty Bag Function
$\lambda_=$	Equality Function

flat	Flatten Function
id	Identity Function
map(f)	Map Function
$\langle a,b\rangle$	Pair
$^{\lambda}\langle a,b\rangle$	Pair Function
π_i	Projection Function
$^{\lambda}\{\{\cdot\}\}$	Singleton Function
in	Belongs to
$^{\lambda}\times$	Cross Product
$^{\lambda}\bot$	Undefined Function
$\bot$	Undefined
$^{\lambda}\langle\rangle$	Empty Tuple Function
$\langle\rangle$	Empty Tuple
c	NRC Constant
e_1	NRC Expression Variable
e_2	NRC Expression Variable
e_3	NRC Expression Variable
e_4	NRC Expression Variable
s	NRC Expression Variable
$t1$	NRC Expression Variable
$t2$	NRC Expression Variable
t	NRC Expression Variable
u	NRC Expression Variable

$x := e\,;\, y$	Let Statement
x	NRC Variable
y	NRC Variable
z	NRC Variable

Acronyms

CPL	Collection Programming Language
DBMS	Database Management System
GUI	Graphical User Interface
NNRC	Named Nested Relational Calculus
NRA	Nested Relational Algebra
NRC	Nested Relational Calculus
ODMG	Object Data Management Group
OQL	Object Oriented Query Language
RDF	Resource Description Framework
RGL	RDF Gears Language
RSA	Relative Set Abstraction
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
URI	Uniform Resource Identifiers

Chapter 1

Introduction

RDF Gears, as described by Eric Feliksik in [1], is a data integration framework developed at TUDelft, which uses the RDF Gears Language(RGL) that allows convenient construction of complex expressions. RGL is a nested workflow language, which allows to combine processes in a workflow, and these workflows can be nested. RDF Gears is based on the nested relational model, and uses it in generating RDF-generating workflows. It also combines these user-defined workflows with SPARQL queries that can work on huge volumes of data. RDF Gears forms the inspiration for our research. In order to optimize RDF Gears query evaluation, we study the rewriting using Nested Relational Algebra and Nested Relational Calculus introduced by Limsoon Wong [2]. The result of this research is theoretical and applicable to, not only RDF Gears, but also other nested relational model based query languages like PigLatin [3] and HiveQL [4].

This chapter is organized as follows. In section 1.1, we explain the motivation behind solving this problem and how this work can be applied to RDF Gears, moreover, what benefits will be achieved by using this work. In section 1.2 and 1.3, we define the problem statement and research questions raised during this thesis, respectively.

1.1 Motivation

RDF Gears, the inspiration for this work, uses a graphical notation to represent its workflow graphs. Defining optimization techniques in this graphical notation is quite cumbersome. Hence, we use a textual notation, such as NRA, which is quite close to the graphical notation. NRC rewriting is even more simple to understand so we also looked at NRC. And we compare the two to each other, as NRA is closer to graphical notation and NRC is closer

to conventional textual calculi and query languages such as SQL. Thus, we can utilize the existing strategies for optimization as well. So it was interesting to delve deep into both and compare them in terms of power of the rewrite rules, which is also our main focus.

1.2 Problem Statement

Query optimization has played a vital role in database research since the 1970's and up till now. A variety of query optimization techniques have been proposed for relational database systems like Heuristic Optimization, Statistics and Cost-based Query Optimization, Indices for fast data access and Semantic Query Optimization.

Indexing and cost-based rewriting have almost become the de-facto standards of query optimization in almost all relational database system since they help to reduce query evaluation time by orders of magnitude thus making database systems more efficient and usable for practical purposes.

In the relational context, a prerequisite to query optimization is the study of equivalences over relational algebra expressions. It is most natural to assume that we need to develop efficient evaluation approaches for SPARQL for efficient realization of Semantic Web. If we can optimize queries it would make it easier to deal efficiently with RDF repositories containing billions of RDF triples.

One of the starting points would be to focus on pushing down projection operators in the operator tree. A good optimization scheme should be able to evaluate the consequence of applying the selection/projection operators at different positions in the execution tree. Early projection may serve as a filter that can significantly decrease the size of intermediate results, thus speeding up subsequent computations.

Sophisticated techniques for query transformations for traditional query languages include rewriting based optimizations. Once the query is parsed, rewrite rules can be applied to the query followed by query plan generation and optimization. Rewriting optimization transforms a query into equivalent one and can have a significant impact on execution time. Query rewriting can be done on the basis of information obtained from the query itself, views to which the query is addressed, integrity constraints and the schema of data queried. In literature, we find many ideas for rewriting based optimizations. One is to find a set of

rewrite rules that can be applied to a parsed query, with each rewrite trying to optimize the query further.

By pushing down selection/projection till we cannot apply it anymore we could end up with a normal form of the expression. Using the rewrite rules based on NRC and NRA, we could achieve a standardized normal form as well, by applying a set of rewrite rules at different levels until we cannot apply any of them anymore. Using this we could achieve a optimized form or normal form of the original query or programming language construct. But we take a step back, and focus on a largely theoretical problem here. We took an initial step towards solving the optimization issue by creating knowledge that can be used to build better optimizers and optimization schemes in the future.

In this process, we try to better understand, which optimization rules are covered in the literature and might be useful, and even find extra rules that are not present in the literature. We investigate that what rewrite rules we can work with, if we want to define a normal form, rather than defining a normal form itself.

In the work done by Limsoon Wong [2] and Leonidas Fegaras [5], we see normalization techniques and sets of rewrite rules, but it is not clear that these are all the rules that we need. In our work, we aim for an initial step by making an inventory of the rewrite rules that have been covered in such literature, and investigate if more rules can be added to these sets so that we could achieve better optimization in the future. We succeed in this endeavour and find such rules that are in fact missing in their work, one such example is the following.

$$^{\lambda}\times \circ\, ^{\lambda}\langle f,g\rangle \equiv \mathbf{map}(^{\lambda}\langle \pi_2,\pi_1\rangle)\circ(^{\lambda}\times \circ\, ^{\lambda}\langle g,f\rangle) \tag{1.1}$$

The above rule claims that if we take a cartesian product of a pair $^{\lambda}\langle f,g\rangle$, it will be the same as taking the cartesian product of pair $^{\lambda}\langle g,f\rangle$ and then applying $\mathbf{map}(^{\lambda}\langle \pi_2,\pi_1\rangle)$ to it. The rule 1.1 clearly holds, and it does not flow from their rules, we prove this later on in our work. We look at what other rules are there that may potentially be used for optimization.

Here we do not aim for completeness, because proving completeness or finding the complete set of rules would mean finding "all rules that hold", which may well be infinite. Since, completeness is uninteresting at this point, we aim to find a kind of generating set that allows us to derive all the rules that hold, but that set in itself is quite compact. It is small in the sense that it has no strict or proper subset that also entails all the other rules that hold. We call this set a minimal set or concise set.

In order to create a powerful generating set, we cross match between NRA and NRC rules sets and see if one can do what the other can do, as well. So that we know that when we do a rewrite in NRA, whether we can do the same rewrite in NRC or not. For example, if we see this projection rule in NRA

$$\pi_1 \circ^{\lambda} \langle f, g \rangle \equiv f \tag{1.2}$$

we see that we can translate this to a corresponding projection rule in NRC as well.

$$(\langle e_1, e_2 \rangle).1 \equiv e_1 \tag{1.3}$$

Even though the syntax is different, one is algebraic and one is based on calculus, one has functions that are composed together and the other uses the same input which happens to be a free variable in NRC, but they are essentially doing the same rewrite, they express the same function.

We also look for equivalence in these NRC and NRA rule sets in a similar manner. When we have a rewrite rule in NRA we check whether we can express this rewrite rule in NRC or not. Say the rewrite rule, we are examining in NRA is,

$$\pi_1 \circ^{\lambda} c \equiv^{\lambda} \bot \tag{1.4}$$

It can be translated to NRC to get the following:

$$x := c \,;x.1 \equiv \bot \tag{1.5}$$

We find there is no direct correspondence to describe the rewrite 1.5 in NRC. However, we have three other rules that collectively describe the rewrite in NRC, which are as follows:

$$x := e_1 \,; x \equiv e_1 \tag{1.6}$$

$$x := e_1 \,; e_2.1 \equiv (x := e_1 \,; e_2).1 \tag{1.7}$$

$$c.1 \equiv \bot \tag{1.8}$$

Similarly we see that in NRC, the expression

$$x := e_1 \,; \bot \equiv \bot \tag{1.9}$$

is equivalent to the following NRA expressions:

$$^{\lambda}\langle\rangle \circ f \equiv^{\lambda} \langle\rangle \tag{1.10}$$

$$\mathbf{flat} \circ \mathbf{map}(^{\lambda}\langle\rangle) \circ^{\lambda} \{\{\cdot\}\} \equiv^{\lambda} \bot \tag{1.11}$$

$$\mathbf{map}(f) \circ^{\lambda} \{\{\cdot\}\} \equiv^{\lambda} \{\{\cdot\}\} \circ f \tag{1.12}$$

So, we say that the two rule sets are equivalent if they can express the same rewrites. This is the main focus of the research, we see whether these two sets are equivalent in the manner explained above, and we prove this in the thesis.

1.3 Research Questions

As the scope of the project we focus on the problem of identifying and proving a certain type of equivalence relationship between the NRA and NRC rule sets. As an outcome of the literature study we present some of the research questions that were encountered during this time. Some of these questions will define many design choices made for this project. These questions will be dealt with throughout the thesis. Some of them are:

- Is there a minimal core set or concise set in NRA such that it subsumes the exhaustive set of all rewrite rules covered in this work?
- Is there a minimal core set or concise set in NRC such that it subsumes the exhaustive set of all rewrite rules covered in this work?
- Is there an equivalence relationship between the rule sets of NRA and NRC?

  Note: By concise we mean that the smallest non-redundant set in which none of the rewrite rules are derivable by the others.

1.4 Organizational Description of Thesis

This thesis is organized in 8 chapters keeping in mind the logical consistency and the flow of the material. Chapter 1 have defined the research goals, the problem statement, scope and motivation behind this thesis. In Chapter 2, we discuss the background information that we have used as a starting point of our work. It also discusses the related work done in this area and how it connects with our work. Chapter 3 outlines the core of our work as it defines the syntax and semantics of Nested Relational Calculus and Nested Relational Algebra that we have used throughout the thesis. It describes in detail each of the symbols used in the syntax of these languages. It also describes the mapping from NRA to NRC and vice-versa.

Chapter 4 relates to NRA only and contains the full set of NRA axioms or rewrite rules that we have considered for our work and the concise set of these axioms which can contain

or prove all the other axioms of NRA. It also describes the work done in order to achieve this concise set. Chapter 5 also relates to NRA and presents the mathematical proofs of the axioms that we say can be proven using our NRA concise set. Chapter 6 presents the full and concise sets of NRC, and also describes the process followed to arrive at the concise set. In addition it describes about Prover 9 [6], the tool we have used to analyze these axioms and the concise and full sets. It also states the auxiliary rules that were needed to describe to Prover 9 the language and operations as such, without which the use of Prover 9 would be impossible.

In Chapter 7, draw a relationship between NRA and NRC using the axioms for Basic Values, Pairs and Tuples, for top level of the expression. The aim is to be able to establish the type of relationship that exists between these languages. In Chapter 8, we summarize our results and conclude this thesis. At this point we even answer the research questions that were asked in Section 1.4. We also present future insights on the application of this work to RDF Gears.

Chapter 2

Background

2.1 Query Optimization

The success of a Database Management System(DBMS) lies largely in the quality, functionality and sophistication of its query optimizer. It is the query optimizer that determines the performance of the system. As the size of underlying data, is growing exponentially many DBMSs proved to be inefficient in handling powerful operations. As defined by Jarke & Koch [7], query optimization tries to solve this problem by integrating a large number of techniques and strategies, ranging from local transformations of queries of access paths and the storage of data on the file system level. The gain in efficiency however, must be traded off against the cost of query optimization itself. Optimization is both a challenge and opportunity Chris J. Date [8] for relational system. It is a challenge because optimization is required if the system is to achieve optimum performance and it is an opportunity because it is precisely one of the strengths of such systems that relational expressions are at a high semantic level that optimization is feasible in the first place.

In the following sections, we get a bird's eye view of query optimization and cover some of the important existing research on the subject. This is done as a preparation for designing query optimizers for RDF gears, all the while keeping in mind that our focus is mainly on algebraic optimization, rewriting of expressions and obtaining more efficient query evaluation plans.

As Yannis E. Ioannidis [9] describes the main function of the query optimizer is to examine "ALL" algebraic expressions and choose the one that is estimated to be the cheapest. The optimization is not repeated until database updates render the access plans invalid or

highly suboptimal. This technique is typically called the cost-based query optimization, which is expensive as opposed to heuristic optimization that involves transformation of the query tree using a set of rules which are expected to improve execution performance. Some systems use only heuristics to reduce the number of choices that must be made in a cost model, however, some systems combine both the techniques.

When querying a database many execution plans exist, in order to select the one with the best estimated performance each and every alternative needs to be considered. A query needs to be re-written by applying transformations to get more efficient and equivalent queries. Jarke & Koch [7], define many transformation rules of a given expression into a equivalent one in order to standardize, simplify and ameliorate the evaluation performance. They specify normalization procedures for standardization, transformation rules for general, quantified expressions and empty relations, query detachment and early evaluation and detection and removal of cycles in graphs for efficient evaluation. They also suggest addition of integrity constraints to change the structure of the query for making it more tractable.

Re-writing does not take into account the actual cost and transformation but depends on the static characteristics of queries. The original query is also kept as one of the alternatives and is only discarded if the transformation is known to be always beneficial. This is followed by examination of all possible execution plans for each query produced and selection of the most efficient one. The cost model specifies arithmetic formulas for estimation of cost of each execution plan. Every step in an execution plan corresponds to a formula which determines its cost. These are based on the size of relations, the distribution of values in these relations or even the size of buffer pool.

The total cost to be minimized is the sum of communication cost (cost of data transmission and processing delays), secondary storage access cost (loading pages into main memory), storage cost (memory buffers) and computation cost (CPU usage). Communication costs are however not considered because our main focus is on evaluation strategy. Some common ideas to reduce such costs are:

- To avoid duplication of effort
- The use of standardized parts
- To look ahead in order to avoid unnecessary operations

- Choosing cheap ways to execute elementary operations and sequencing them in an optimal fashion.

Query trees are used to represent an algebraic query, with the leaves as database relations and the non-leaf nodes as algebraic operators making the query transformations essentially graph transformations. For complicated queries there might be a very large number of evaluation plans when selecting the best evaluation plan the size of the space that the search strategy has to explore DBMSs apply many restriction, some typical restrictions are:

- Selections and projections almost never generate intermediate results. Such a restriction eliminates sub-optimal query trees
- Relations are combined through joints and not through Cartesian products unless needed. This restriction almost always eliminates sub-optimal joint trees.
- The inner operand of each joint is a database relation and not intermediate results. This restriction is more heuristic than the first two and might eliminate the optimal plan as well.

Typical query optimizers use the above restrictions to reduce the size of the space they explore. In order to find the cheapest evaluation plan from the set of alternative plans, many different type of search strategies exist. One of the most common approach is the dynamic programming approach which is used by many commercial query optimizers and is considered the standard in query optimization search strategies. The algorithm is a dynamic pruning exhaustive search algorithm which constructs alternative joint trees based on the previously discussed restrictions. It iterates on the number of relations joined so far, pruning trees that are known to be sub-optimal. Merge scan is one of the joint methods that could be used.

The algorithm is guaranteed to find the optimal plan and reduces the search space by dynamically pruning the sub-optimal parts even in case of partial plans that are generated. However, this is still exponential. Exact evaluation of query optimization procedures is in general computationally impractical and is hampered further by the lack of precise statistical information about the database that is why query evaluation algorithm have to rely heavily on heuristics. In practice the dynamic programming approach works well for queries involving less than 10 joints but is unable to cope with really large queries. To address this randomized algorithms that flip coins to make decisions could be used.

Simulated Annealing, Iterative Improvement, Two-Phase Optimization are some of the generic algorithms that would be adapted and applied to a variety of optimization problems. They operate by searching a graph in which the nodes represent the alternative execution plans and have a cost associated with it. The algorithm tries to find a node with a global minimum cost. Despite the probabilistic nature of randomized algorithms, their efficiency makes them appropriate for larger queries. Iterative Improvement finds a reasonable plan very quickly while simulated annealing is able to find a better plan than the former but takes a longer duration. Two-Phase Optimization is a combination of both and is able to find plans as good as simulated annealing in much shorter time.

Genetic algorithms form another class of generic randomized optimization algorithms that can be applied to query optimization. Over the period of time many other search strategies and algorithms have been proposed especially for query optimizations which are not in scope of this chapter.

2.2 RDF Gears

RDF Gears, as described in Eric Feliksik in [1], is a data integration framework for the Semantic Web. The RDF Gears framework consists of three main components the RDF Gears Language, the RDF Gears Graphical User Interface (GUI) and the RDF Gears Engine.

2.2.1 RDF Gears Language

RGL provides an expressive domain specific language for the development of work flows integrating RDF data with other sources. By combining the semantic web technology with nested relational algebra, it allows developers to work on new domain specific algorithms and provides a right level of abstraction to make the expression of such algorithms convenient. It has clearly defined and understandable functionality and formally defined syntax and semantics.

RGL is workflow language that combines processors in a work flow. Processors are blocks that represent certain functioned having named arguments. The processors has a input and output port which can be connected to another processor's input/output to create an acyclic directed graph which is called the workflow. The workflows can be used as functions in another function in a larger workflow in order to create nested workflows.

RGL uses the basic RDF elements like Literal Uniform Resource Identifiers(URI's) and Graph, it also allows complex nested collection like records and bags based on the Named Nested Relational Calculus(NNRC). Functions operating on these data structures can be customized to make the core RGL more expressive. Re-use of workflows is enabled by nesting workflows iterate over part of a larger work flow and to promote modularity. RGL also features static type system that allows evaluating the correctness of expression before execution.

In order to stay close to RDF paradigm and to make RGL more expressive it allows fetching data from SPARQL endpoint using the RGL-SPARQL operator. RGL is suitable for applying optimization techniques because of its functional and statically typed nature. Currently, lazy evaluation and field map indexing have been implemented. Nested Relational Algebra's rewriting rules can be translated to RGL but will need to be adapted to a graph structure instead of a tree structure as in NRA and named tuples instead of two-tuples only.

Over all we see that RGL is formally defined and easy to understand for non-programmers who deal with data integration in Semantic Web. It does so by abstracting the implementation details of data transformation, storage and optimization, allowing them to focus on conceptual data transformation and not on implementation details. Its high levels of expressivity and extendibility make it a handy tool for developers and researchers working in the field of applied RF data integration.

2.2.2 RDF Gears Engine

The RDF Gears Engine serves as an interpreter for RGL. It executes the workflows defined in XML and uses Jena and ARQ library for querying and manipulation. The engine can be run from command line and has many flags to direct the engine executable as to where the work flow is found, should it type checked or executed, its serialization mechanism etc.

The architecture of the engine is object oriented and hence it is flexible and extendible. Some optimizations are implemented in order to reduce the run time and memory consumption of work flow execution. The pipelining approach is introduced to reduce the space occupied by bags in memory. The lazy evaluation scheme will delay the evaluation of values as long as possible.

Field map indexing technique allows record to store their values in an internal array and map field names to array indices in separate class such that records can share field map index instead of storing a map.

2.3 PigLatin

Pig is used for ad-hoc data analysis and the associated query language is called as PigLatin. Pig is meant for offline, ad-hoc scan-centric work loads. This query language is a good balance between declarative style of Structured Query Language(SQL) and low-level procedural style of map-reduce. The query language program is a sequence of steps where each step specifies only a single high level data transformation. PigLatin supports features mentioned below:

- fully nested data model
- user defined functions
- ability to operate without any schema information

PigLatin has a flexible, fully nested data model and allows complex non-atomic data types such as set, map and tuple.

2.4 HiveQL

HiveQL is the query language of system Hive. It is SQL like query language which supports select, project, join, aggregate, union all and sub-queries in the from clause. This language supports data definition statements to create tables with specific serialization formats and partitioning and bucketing columns. It also supports multi-table insert, where users can perform multiple queries on the same input data using a one HiveQL statement. This language also supports user defined column transformations and aggregation functions.

2.5 Nested Relational Calculus

As discussed by Roth & Korth & Silberschatz [10], a database scheme is defined as a set of rules of a certain order in which a relation corresponds to a value of a particular domain. In case of Nested Relational Calculus the value can be a nested relation in itself. NRC uses

the complex object data model and is a conservative extension of the relational algebra and can itself be viewed as a data processing core of Object Query Language (OQL) introduced in ODMG [11]. The relational calculus is considered to be the basis for SQL language as discussed in Elmasri & Navathe [12].We define the set that we used and its associated operations and semantics in the following chapter.

2.6 Nested Relational Algebra

Roth & Korth & Silberschatz [10] define Nested Relational Algebra as the query language that queries the Nested Relational Model. Given a Tuple calculus expression $\{t|v(t)\}$, there exists an algebra expression d_{t_i} for the limited domain of each tuple variable t_i mentioned in v or any nested relation of v. Query optimization also occurs at relational algebra level as discussed in Silberschatz & Korth [13]. The internals of many database implementations for query processing and optimization are in relational algebra.

2.7 Related Work

There is a good amount of literature based on database calculi and algebras. Some of it is to extend the relational model, others to study complexity and expressive power of languages, some for nested structures and so on. Our work specifically deals with identifying rewrite rules for query optimization using NRA and NRC and drawing an equivalence between these two sets of rewrite rules. RDF Gears serves as an inspiration to formulate these, but they can be very well be applied to other query languages like Piglatin and Hive.

Like in Leonidas Fegaras [5] we also support a small number of operators. Our NRC expressions can be transformed into equivalent expressions using NRA rewrite rules, and vice versa. These expressions can be applied to rewriting based query optimizers useful for only query languages dealing with collection types.

Leonidas Fegaras [14] presents monoid calculus and claims that it provides an effective framework for processing object oriented query languages like OQL. Their monoid calculus is based on monoid comprehensions and deals with multiple bulk (collection) types, arbitrary nesting of type constructors, methods and embedded query expressions permitted whenever a collection may appear. Monoid comprehension is similar to set former notation but applicable to types rather than sets , where a monoid refers to a general template for a

data type. Fegaras describes monoid homomorphisms as a map or query that maps some monoid to a particular monoid. Based on this they define monoid comprehension calculus, that captures operations for collection types. Fegaras formalizes this framework and puts the monoid calculus into a canonical form by an efficient rewrite algorithm called normalization algorithm. These form the basis and a starting point of our research.

Leonidas Fegaras [5] presents a more detailed work and extends the concepts of monoid comprehension calculus making it well typed and more expressive. We take inspiration from the mentioned typing rules while defining the semantics of our NRA and NRC. They define the procedure of translating OQL to the monoid calculus and further extend their normalization algorithm. They also prove that their normalization algorithm works correctly and preserves meaning by proving each individual rewrite transformation correct. The calculus is easy to manipulate using this normalization algorithm as it unnests many forms of nested comprehensions. These comprehensions are then translated to lower level algebra that reflects many rational DBMS's. They also present an implementation of a ODMG(Object Data Management Group)-based DBMS called λ-DB and use it to evaluate the performance of their query-unnesting algorithm.

In this work related to database queries Jan Van den Bussche [19] views the database query languages as just domain specific programming languages and argues that the tools of type systems and reflection can be applied easily to database context. As query languages are less than turing complete they may exhibit static type systems that are both sound and complete as they can prove both the absence and presence of crashes. In this work, Bussche gives an overview of the recent work on types and reflection in programming languages. Reflection is the ability, within the run of a program to inspect a program, generate and execute other programs. Type reflection can be used as an extra feature of query languages. Bussche uses static type systems to ensure that well typed programs do not crash. Earlier work of this was not as flexible because it could prove the absence of crashes but not their presence. This was solved by using polymorphic operators that can be applied to wide range of types. This provides logical data independence. Bussche goes on to define a static type system for NRC which inspires the static type system for our NRC and we have used a very similar notation to describe our typing rules.

Expressions in Named Nested Relational Calculus are not defined on every inputs. NNRC is the canonical query language for nested relational or complex object data model.

In this work, Jan Van den Bussche [17] describes typability of NNRC expressions. He provides a description of the set of all possible typings of an NNRC expression by means of a conjunctive logical formula. He goes on to show that the satisfyability of such conjunction formula is complicated. The author presents an elementary polymorphic type inference for NNRC which was quite useful as a study for this work, since type inference helps in determining for what type of schema is a given query applicable.

Jan Van den Bussche [18] studies the well-definedness problem for a programming language using semantic type checking. Given an expression and an input type deciding whether the semantics of the expression defined for all the inputs adhering to the input type. Well definedness is undecidable for most programming languages and Bussche studies the well definedness in NRC as a sound and complete type system could be found for a smaller specific purpose language. In the work Buscche details the syntax and evaluation relation for NRC expressions. he argues that the well definedness of NRC is undecidable but the well definedness of a smaller PENRC is decidable. When PENRC has a destructor for the singleton set constructor unlike standard NRC, the well definedness becomes undecidable again. He also evaluates the impact of type test expressions that allow the inspction of the type of a value at runtime. he shows that adding a type test construct to PENRC also makes the well definedness undecidable.

Limsoon Wong [15] presents a query language based on comprehension that is equivalent to his version of NRA and NRC as described in [16], and call it Relative Set Abstraction (RSA). He defines the language an semantics for sets, functions, lists and bags, all of which can be nested. The main work interesting to us is Wong's rewriting system on RSA consisting of rewrite rules(or axioms of the equational theory of RSA) that are strongly normalizing. He proves that the conservative property of his language holds uniformly, irrespective of using list/bags or sets.

Limsoon Wong [2], lays the foundation for querying nested collections. He defines query languages, the typing rules in NRC, and discusses the expressive power of NRC and other nested relational languages. He also defines a collection programming language(CPL) and a plethora of structural optimization techniques which have inspired our rewriting techniques as well. In addition, during his extensive work he has also an open query system, Kleisli, that uses CPL and allows introduction of new primitives, optimization rules, cost functions, data scanners and data writers.

It is the extensive study of all these works that has given us inspiration for this thesis and also covered and aided our basic understanding of nested relational model and rewriting based optimizations.

Chapter 3

Syntax and Semantics

This chapter describes syntax and semantics of NRA and NRC. In addition, we briefly describe the operators, functions or variables used in the languages, though most of them are intuitive.

3.1 The Nested Relational Algebra: NRA

The NRA described here is defined over bags (and not sets or lists). The syntax for the algebra over bags we intend to use:

$$\begin{aligned} F ::= {} & \mathbf{id} \mid B \mid F \circ F \mid {}^{\lambda}C \mid {}^{\lambda}\langle F, F \rangle \mid \pi_1 \mid \pi_2 \mid {}^{\lambda}\langle\rangle \mid \\ & {}^{\lambda}\{\{\}\} \mid {}^{\lambda}\{\{\cdot\}\} \mid {}^{\lambda}\uplus \mid \mathbf{map}(f) \mid \mathbf{flat} \mid {}^{\lambda}\times \mid \\ & \mathbf{set} \mid {}^{\lambda}= \mid {}^{\lambda}\bot. \end{aligned}$$

The semantics of NRA are based on the following concepts:

- F is a non-terminal NRA expression
- *F_NRA* will henceforth denote the set of all expressions defined by the above syntax.
- We annotate some constructs with λ to indicate they denote functions rather than values.
- **id** acts like a validation check by returning whatever it receives.
- Function Composition $f \circ g$ means the function g will be computed first and then the function f will be applied to the result.

- Constant constructor ${}^{\lambda}c$ returns a constant.
- Pair Constructor ${}^{\lambda}\langle f,g\rangle$ forms a pair with any two elements.
- Projections π_1 and π_2 are the two projections on pairs. π_1 returns the first element of the pair and π_2 returns the second element of the pair.
- Tuple constructor ${}^{\lambda}\langle\rangle$ generates a tuple and can even generate an empty tuple.
- Bag Constructor ${}^{\lambda}\{\{\}\}$ forms a bag/multiset and can even create an empty bag.
- Singleton function ${}^{\lambda}\{\{\cdot\}\}$ returns the singleton bag.
- Additive Bag Union function $f\ {}^{\lambda}\uplus\ g$ means the result will be the merged set/bag of the compositions resulted by the functions f and g. This expression is meaningful only for compositions because if one of them is not a composition then the result will be ${}^{\lambda}\perp$.
- **map**(f) puts whatever element it receives as puts it in a bag and returns it.
- **flat** returns the contents of the bag, hence when applied to anything but a bag, it would return ${}^{\lambda}\perp$.
- Cross Product $f\ {}^{\lambda}\times\ g$ means the result will be the cross product of the compositions resulted by the functions f and g. Like the ${}^{\lambda}\uplus$, ${}^{\lambda}\times$ is also meaningful only for compositions because if one of them is not a composition then the result will be ${}^{\lambda}\perp$.
- The function **set** eliminates duplicates.
- Equality function ${}^{\lambda}=$ checks whether the two elements in a pair of basic values are the same or not. In case, a non-pair is presented to this function the resultant will be an ${}^{\lambda}\perp$, and the result would be the same in case of a pair containing one or more non-basic values.

The following concepts are used to informally describe a few constructs used in the semantics of NRA and NRC.

- We define the proposition $(x,y)\in[\![e]\!]$ which denotes that the function associated with e maps the value x to y.

- We assume the set $\mathscr{V}$ of all values, which also includes $\perp$ denoting undefined, $\mathscr{B} \subset \mathscr{V}$ of all basic values, $\mathscr{C} \subset \mathscr{V}$ of all collections and $\mathscr{P} \subset \mathscr{V}$ of all pairs. $\mathscr{V}$ can only contain $\perp, \mathscr{B}, \mathscr{P}$ or $\mathscr{C}$. In other words, $\mathscr{V} = \perp \cup \mathscr{B} \cup \mathscr{P} \cup \mathscr{C}$.
  $\mathscr{P} \equiv \{(x,y) | x \in \mathscr{V} \wedge y \in \mathscr{V}\}$
  $\mathscr{C} \equiv \{\{\{v_1, ..., v_m\}\} | \forall_{i=1}^{m} v_i \in \mathscr{V}\}$

- The semantic rules define a total function over $\mathscr{V}$ for each expression in the syntax, i.e. a binary relation that maps each input value, that is a nested value in $\mathscr{V}$, to a unique output value in $\mathscr{V}$. If the input of the function is not suited for the operation, then we return the undefined value.

The semantics of the algebra is defined by the following rules.

$$\frac{x \in \mathscr{V}}{(x,x) \in [\![\mathbf{id}]\!]}$$

$$\frac{(x,y) \in [\![f]\!] \qquad (y,x) \in [\![g]\!]}{(x,z) \in [\![f \circ g]\!]}$$

$$\frac{x \in \mathscr{V}}{(x,c) \in [\![^{\lambda} c]\!]}$$

$$\frac{(x,y) \in [\![f]\!] \qquad (x,z) \in [\![g]\!]}{(x, \langle y,z \rangle) \in [\![^{\lambda} \langle f,g \rangle]\!]}$$

$$\frac{x,y \in \mathscr{V}}{(\langle x,y \rangle, x) \in [\![\pi_1]\!]} \qquad \frac{x \notin \mathscr{P}}{(x, \perp) \in [\![\pi_1]\!]}$$

$$\frac{x = \{\{x_1, \ldots, x_n\}\} \in \mathscr{V} \qquad y = \{\{y_1, \ldots, y_m\}\} \in \mathscr{V} \qquad \vee i = \{\{1, \ldots n\}\} \qquad (x,y) \in [\![f]\!]}{(x,y) \in [\![map(f)]\!]}$$

$$\frac{x,y \in \mathscr{V}}{(\langle x,y \rangle, y) \in [\![\pi_2]\!]} \qquad \frac{x \notin \mathscr{P}}{(x, \perp) \in [\![\pi_2]\!]}$$

$$\frac{x \in \mathscr{V}}{(x, \langle\rangle) \in [\![^{\lambda} \langle\rangle]\!]}$$

$$\frac{x \in \mathscr{V}}{(x, \{\{\}\}) \in [\![^{\lambda} \{\{\}\}]\!]}$$

$$\frac{x \in \mathscr{V}}{(x, \{\{x\}\}) \in [\![^{\lambda} \{\{\cdot\}\}]\!]}$$

$$\frac{x,y,z \in \mathscr{V} \qquad z = x \uplus y}{(\langle x,y\rangle, z) \in [\![^{\lambda} \uplus]\!]} \qquad \frac{x \notin \mathscr{P}}{(x,z) \in [\![^{\lambda} \uplus]\!]} \qquad \frac{x \notin \mathscr{C} \vee y \notin \mathscr{C}}{(\langle x,y\rangle, \bot) \in [\![^{\lambda} \uplus]\!]}$$

$$\frac{x = \{\{x_1,\ldots,x_n\}\} \in \mathscr{V} \qquad y = \uplus_{i=1}^{n} x_i}{(x,y) \in [\![\mathbf{flat}]\!]} \qquad \frac{x = \{\{v_1,\ldots,v_m\}\} \exists i : v_i \notin \mathscr{C}}{(x,\bot) \in [\![\mathbf{flat}]\!]} \qquad \frac{x \notin \mathscr{C}}{(x,\bot) \in [\![\mathbf{flat}]\!]}$$

$$\frac{x = \{\{x_1,\ldots,x_n\}\} \in \mathscr{V} \qquad y = \{\{y_1,\ldots,y_m\}\} \in \mathscr{V} \qquad z = \uplus_{i=1}^{n}(\uplus_{j=1}^{m} \{\{\langle x_i, y_j\rangle\}\})}{(\langle x,y\rangle, z) \in [\![^{\lambda} \times]\!]}$$

$$\frac{x \notin \mathscr{C} \vee y \notin \mathscr{C}}{(\langle x,y\rangle, \bot) \in [\![^{\lambda} \times]\!]} \qquad \frac{x \notin \mathscr{P}}{(x,\bot) \in [\![^{\lambda} \times]\!]}$$

$$\frac{x = \{\{x_1,\ldots,x_n\}\} \in \mathscr{V} \qquad y = \cup_{i=1}^{n} \{\{x_i\}\}}{(x,y) \in [\![\mathbf{set}]\!]} \qquad \frac{x \notin \mathscr{C}}{(x,\bot) \in [\![\mathbf{set}]\!]}$$

$$\frac{x \in \mathscr{B}}{(\langle x,x\rangle, \{\{\langle\rangle\}\}) \in [\![^{\lambda} =]\!]} \qquad \frac{x,y \in \mathscr{B} \qquad x \neq y}{(\langle x,y\rangle, \{\{\}\}) \in [\![^{\lambda} =]\!]}$$

$$\frac{x \notin \mathscr{P}}{(x,\bot) \in [\![^{\lambda} =]\!]} \qquad \frac{y \notin \mathscr{B} \vee z \notin \mathscr{B}}{(\langle y,z\rangle, \bot) \in [\![^{\lambda} =]\!]}$$

3.2 The Nested Relational Calculus: NRC

The NRC described here is defined over bags (and not sets or lists). The syntax for the calculus over bags we intend to use:

$$\begin{aligned} E ::= {} & X \mid C \mid \langle E,E\rangle \mid E.1 \mid E.2 \mid \langle\rangle \mid \\ & \{\{\}\} \mid \{\{E\}\} \mid E \uplus E \mid \{\{E \mid X \in E, \ldots, X \in E\}\} \mid \\ & B(E) \mid \mathbf{set}(E) \mid E \approx E \mid X := E \;;\; E \mid \bot. \end{aligned}$$

The semantics of NRC are based on the following concepts:

- E is a non-terminal NRC expression
- *E_NRC* will henceforth denote the set of all expressions defined by the above syntax.
- X denotes variables
- c represents basic value constants
- Pair constructor is shown by $\langle E, E \rangle$ and it combines two elements to form a pair.
- $E.1$ and $E.2$ are projections on a pair. In case of non-pairs they would result in $\perp$.
- $\langle\rangle$ the empty tuple or unit value
- $\{\{\}\}$ denotes bags/multisets
- $\{\{E\}\}$ is the Singleton bag with a single NRC expression within
- Additive Bag Union $E \uplus F$ means the result will be the merged set/bag of the compositions E and F. E and F are both NRC expressions. $E \uplus F$ is meaningful only for compositions because if one of them is not a composition then the result will be $\perp$.
- We will use in the right-hand side of comprehensions the short-hand $x := e$ for $x \in \{\{e\}\}$. We will there also write $(e_1 \approx e_2)$ for $z \in (e_1 \approx e_2)$ with z some fresh variable.
- b the basic user-defined functions
- The function **set** eliminates duplicates.
- The expression $e_1 \approx e_2$ compares basic values and returns $\{\{\langle\rangle\}\}$ if they are equal and $\{\{\}\}$ otherwise.
- The expression $x := e_1$ **;** e_2 denotes the assignment of the value of e_1 to variable x and the subsequent evaluation of e_2 under this binding. Note that this construct is redundant, but rewriting it might lead to duplicates of e_1 which we would try to avoid in an efficient query evaluation plan.

The semantics of the calculus is defined by the following rules.

$$\frac{}{\Gamma \vdash x \Rightarrow \Gamma(x)} \qquad \frac{}{\Gamma \vdash c \Rightarrow c} \qquad \frac{\Gamma \vdash e_1 \Rightarrow v_2 \qquad \Gamma \vdash e_2 \Rightarrow v_2}{\Gamma \vdash \langle e_1, e_2 \rangle \Rightarrow \langle v_1, v_2 \rangle}$$

$$\frac{\Gamma \vdash e \Rightarrow \langle v_1, v_2 \rangle}{\Gamma \vdash e.1 \Rightarrow v_1 \qquad \Gamma \vdash e.2 \Rightarrow v_2} \qquad \frac{\Gamma \vdash e \Rightarrow v \qquad v \notin \mathscr{P}}{\Gamma \vdash e.1 \Rightarrow \bot \qquad \Gamma \vdash e.2 \Rightarrow \bot} \qquad \frac{}{\Gamma \vdash \langle\rangle \Rightarrow \langle\rangle}$$

$$\frac{}{\Gamma \vdash \{\{\}\} \Rightarrow \{\{\}\}} \qquad \frac{\Gamma \vdash e \Rightarrow v}{\Gamma \vdash \{\{e\}\} \Rightarrow \{\{v\}\}}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow \{\{v_1, \ldots, v_n\}\} \qquad \Gamma \vdash e_2 \Rightarrow \{\{w_1, \ldots, w_m\}\}}{\Gamma \vdash e_1 \uplus e_2 \Rightarrow \{\{v_1, \ldots, v_n, w_1, \ldots, w_m\}\}}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow x \qquad \Gamma \vdash e_2 \Rightarrow y \qquad x \notin \mathscr{C} \vee y \notin \mathscr{C}}{\Gamma \vdash e_1 \uplus e_2 \Rightarrow \bot}$$

$$\frac{\Gamma \vdash e_2 \Rightarrow \{\{v_1, \ldots, v_m\}\} \qquad \forall_{i=1}^{m} (\Gamma_{[x \mapsto v_i]} \vdash e_1 \Rightarrow w_i)}{\Gamma \vdash \{\{e_1 \mid x \in e_2\}\} \Rightarrow \{\{w_1, \ldots, w_m\}\}} \qquad \frac{\Gamma \vdash e_2 \Rightarrow v \qquad v \notin \mathscr{C}}{\Gamma \vdash \{\{e_1 \mid x \in e_2\}\} \Rightarrow \bot}$$

$$\frac{\Gamma \vdash e_2 \Rightarrow \{\{v_1, \ldots, v_m\}\} \qquad \forall_{i=1}^{m} (\Gamma_{[x \mapsto v_i]} \vdash \{\{e_1 \mid \Delta\}\} \Rightarrow w_i)}{\Gamma \vdash \{\{e_1 \mid x \in e_2, \Delta\}\} \Rightarrow \uplus_{i=1}^{m} w_i} \qquad \frac{\Gamma \vdash e_2 \Rightarrow v \qquad v \notin \mathscr{C}}{\Gamma \vdash \{\{e_1 \mid x \in e_2, \Delta\}\} \Rightarrow \bot}$$

$$\frac{\Gamma \vdash e \Rightarrow v \qquad (v, w) \in [\![b]\!]}{\Gamma \vdash b(e) \Rightarrow w}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow \{\{v_1, \ldots, v_n\}\}}{\Gamma \vdash \mathbf{set}(e) \Rightarrow \cup_{i=1}^{n} (\{\{v_i\}\})} \qquad \frac{\Gamma \vdash e_1 \Rightarrow v \qquad v \notin \mathscr{C}}{\Gamma \vdash \mathbf{set}(e) \Rightarrow \bot}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow v \qquad \Gamma \vdash e_2 \Rightarrow v \qquad v \in \mathscr{B}}{\Gamma \vdash e_1 \approx e_2 \Rightarrow \{\{\langle\rangle\}\}}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow v_1 \qquad \Gamma \vdash e_2 \Rightarrow v_2 \qquad v_1, v_2 \in \mathscr{B} \qquad v_1 \neq v_2}{\Gamma \vdash e_1 \approx e_2 \Rightarrow \{\{\}\}}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow v_1 \qquad \Gamma \vdash e_2 \Rightarrow v_2 \qquad v_1 \notin \mathscr{B} \vee v_2 \notin \mathscr{B}}{\Gamma \vdash e_1 \approx e_2 \Rightarrow \{\{\}\}} \qquad \frac{\Gamma \vdash e_1 \Rightarrow v_1 \qquad \Gamma_{[x \mapsto v_1]} \vdash e_2 \Rightarrow v_2}{\Gamma \vdash x := e_1 \,;\, e_2 \Rightarrow v_2}$$

3.3 Mapping

Mapping is defined for both NRA and NRC to transform expressions of one language to another, i.e., NRA to NRC and vice versa. Both languages describe functions over nested relations (NRC does so if we assume only one free variable, otherwise it is a function that maps several nested relations, one for each variable, to a nested relation). All expressions in NRA and NRC are represented by *F_NRA* and *E_NRC*, respectively. A mapping between two languages, is a set of basic steps/rewrites using which we can convert any string of one language to an equivalent string in the second language. Here, equivalent means that they have the same meaning. For example, π_i in NRA is equivalent to x.1 in NRC, and they both mean projection. We will now define two functions, one that maps NRA expression to an equivalent NRC expression (with one free variable x), and one that maps each NRC expression (with one free variable) to an equivalent NRA expression.

3.3.1 Mapping NRA to NRC

We show that each NRA expression can be represented by an NRC expression where x is the free input variable as given in Jan Hidders [21].

$$M(\mathbf{id}) = x \tag{3.1}$$

$$M(f) = f(x) \tag{3.2}$$

$$M(g \circ f) = x := M(f)\ \mathbf{;}\ M(g) \tag{3.3}$$

$$M(^{\lambda}c) = c \tag{3.4}$$

$$M(^{\lambda}\langle f,g\rangle) = \langle M(f), M(g)\rangle \tag{3.5}$$

$$M(\pi_i) = x.i \tag{3.6}$$

$$M(^{\lambda}\langle\rangle) = \langle\rangle \tag{3.7}$$

$$M(^{\lambda}\{\{\}\}) = \{\{\}\} \tag{3.8}$$

$$M(^{\lambda}\{\{\cdot\}\}) = \{\{x\}\} \tag{3.9}$$

$$M(^{\lambda}\uplus) = x.1 \uplus x.2 \tag{3.10}$$

$$M(\mathbf{map}(f)) = \{\{M(f) \mid x \in x\}\}^1 \tag{3.11}$$

$$M(\mathbf{flat}) = \{\{x \mid x \in x, x \in x\}\}^2 \tag{3.12}$$

$$M(^{\lambda}\times) = \{\{(y,z) \mid y \in x.1, z \in x.2\}\}^3 \tag{3.13}$$

$$M(\mathbf{set}) = \mathbf{set}(x) \tag{3.14}$$

$$M(^{\lambda}=) = (x.1 \approx x.2) \tag{3.15}$$

$$M(^{\lambda}\bot) = \bot \tag{3.16}$$

3.3.2 Transformation of NRA to NRC

In Section 3.3.1, we saw how an NRA expression can be mapped to an NRC expression. In fact, using these mapping rules in combination we can transform an entire NRA equation to a NRC equation. We show an example of this type of transformation below. These transformations are crucial for establishing a relationship between two sets of equations that we see in Chapter 7.

Given an NRA equation :
$^{\lambda}\uplus \circ {}^{\lambda}\langle {}^{\lambda}\langle f,g\rangle, h\rangle \equiv {}^{\lambda}\bot$
We apply the mapping M on both the sides
$M(^{\lambda}\uplus \circ {}^{\lambda}\langle {}^{\lambda}\langle f,g\rangle, h\rangle) \equiv M(^{\lambda}\bot)$
On L.H.S. considering equation 3.3 and 3.5 we obtain
$x := {}^{\lambda}\langle M(^{\lambda}\langle f,g\rangle), h\rangle \; ; \; M(^{\lambda}\uplus)$
now we apply mapping 3.5 and 3.10
$x := {}^{\lambda}\langle {}^{\lambda}\langle M(f), M(g)\rangle, M(h)\rangle \; ; \; x.1 \uplus x.2$
L.H.S of equation using the principle [that each $M(f)$ with f being an expression variable for NRA is replaced with specific expression variable e_i of NRC] becomes
$x := {}^{\lambda}\langle {}^{\lambda}\langle e_1, e_2\rangle, e_3\rangle \; ; \; x.1 \uplus x.2$

[1] It can also be defined as, M(**map**(f)) = $\{\{M(f)_{[x/y]} \mid y \in x\}\}$, as every x in equation 3.11, does not need to be distinct; where, $M(f)_{[x/y]}$ denotes that we compute $M(f)$ such that x is determined by the value of y.

[2] It can also be defined as, M(**flat**) =$\{\{z \mid y \in x, z \in y\}\}$, as every x in equation 3.12, does not need to be distinct.

[3] $x, y \,\&\, z$ are all required to be distinct here, i.e. $y \neq z \,\&\, y \neq x \,\&\, z \neq x$.

R.H.S. can be written using equation 3.16
$\perp$
the complete NRA to NRC mapped equations is
$x := {}^{\lambda}\langle {}^{\lambda}\langle e_1, e_2\rangle, e_3\rangle \; ; \; x.1 \uplus x.2 \equiv \perp$

3.3.3 Mapping NRC to NRA

We show that each NRC expression with a single free variable x can be represented by an NRA expression

$$M'(x) = \mathbf{id} \tag{3.17}$$

$$M'(c) = {}^{\lambda}c \tag{3.18}$$

$$M'(\langle e_1, e_2\rangle) = {}^{\lambda}\langle M'(e_1), M'(e_2)\rangle \tag{3.19}$$

$$M'(e.i) = \pi_i \circ M'(e) \tag{3.20}$$

$$M'(\langle\rangle) = {}^{\lambda}\langle\rangle \tag{3.21}$$

$$M'(\{\{\}\}) = {}^{\lambda}\{\{\}\} \tag{3.22}$$

$$M'(\{\{e\}\}) = {}^{\lambda}\{\{\cdot\}\} \circ M'(e) \tag{3.23}$$

$$M'(e_1 \uplus e_2) = {}^{\lambda}\uplus \circ {}^{\lambda}\langle M'(e_1), M'(e_2)\rangle \tag{3.24}$$

$$M'(\{\{e \mid y \in e'\}\}) = \mathbf{map}(M'(e_{[x/x.1, y/x.2]})) \circ {}^{\lambda}\times \circ {}^{\lambda}\langle {}^{\lambda}\{\{\cdot\}\}, M'(e')\rangle \tag{3.25}$$

$$M'(\{\{e \mid y \in e', \Delta\}\}) = \mathbf{flat} \circ M'(\{\{\{\{e \mid \Delta\}\} \mid y \in e'\}\}) \tag{3.26}$$

$$M'(f(e)) = f \circ M'(e) \tag{3.27}$$

$$M'(\mathbf{set}(e)) = \mathbf{set} \circ M'(e) \tag{3.28}$$

$$M'(e_1 \approx e_2) = {}^{\lambda}= \circ {}^{\lambda}\langle M'(e_1), M'(e_2)\rangle \tag{3.29}$$

$$M'(y := e\ ;\ e') = M'(e'_{[x/x.1,y/x.2]}) \circ^{\lambda} \langle \mathbf{id}, M'(e) \rangle^4 \tag{3.30}$$

$$M'(x := e\ ;\ e') = M'(e') \circ M'(e) \tag{3.31}$$

$$M(\perp) = {}^{\lambda}\perp \tag{3.32}$$

3.3.4 Transformation of NRC to NRA

In Section 3.3.3, we saw how an NRC expression can be mapped to an NRA expression. In fact, using these mapping rules in combination we can transform an entire NRC equation to a NRA equation. We show an example of this type of transformation below. These transformations are crucial for establishing a relationship between two sets of equations that we see in Chapter 7.

Given the equation:
$x := c\ ;\ \langle\rangle \equiv \langle\rangle$

On applying mapping M' on both the sides of the equation we get :
$M'(x := c\ ;\ \langle\rangle) \equiv M'(\langle\rangle)$
Now using equation 3.31 on the L.H.S. and equation 3.21 on R.H.S. we get :
$M'(c) \circ M'(\langle\rangle) \equiv {}^{\lambda}\langle\rangle$
We use equation 3.18 and 3.21 on the L.H.S. we get :
${}^{\lambda}c \circ {}^{\lambda}\langle\rangle \equiv {}^{\lambda}\langle\rangle$
The above equation is the transformed NRA equation.

3.4 Rewrite Rules

The RDF Gears engine executes work flow graphs. These work flow graphs are essentially a list of small tasks/queries/operations that are nested and connected to form directed acyclic graphs. The problem of query optimization can be formulated by taking an initial workflow graph, analyzing it and transforming it to another work flow graph that we claim would be more efficient in executing on the RDF Gears engine and yielding the same result.

The efficiency computation is outside the scope of this work, but here we determine here the kind of transformations that are possible to this original work flow graph. For this we

[4] $M'(y := e\ ;\ e') = M'(e'_{[x/x.1,y/x.2]}) \circ^{\lambda} \langle \mathbf{id}, M'(e) \rangle$ holds only for $y \neq x$.

study the existing literature and find rules that can be applied to this transformation graph to result in one or more work flow graphs, yielding same result as original work flow graph.

Finding the rewrite rules forms the core part of this work and as a result of my thesis we present two sets of rewrite rules one in NRA and other in NRC. We also try to draw a relationship between our set of rules for the two respective notations.

A rewrite can be within the same language, using which we can convert any string of one language to an equivalent string in the same language. We use a set of rewrite rules to describe the possible rewrites within that language. Here e_1 and e_2 are equivalent because they mean the same thing i.e., denote the same function. The solid arrow here, represents that a single rewrite would suffice to reach from e_1 to e_2. In case, of one or more rewrites we will represent that with a dotted arrow.

$$e_1 \xrightarrow{rewrite} e_2$$

Most existing optimization schemes are based on NRC, so we take that as a starting point, especially one by Leonidas Fegaras [5] and transform them to NRA. RGL is very similar to NRA, and doing this brings us close to finding and using existing optimization strategies, as existing literature is overloaded with these. If we are able to utilize this knowledge we can make RDF Gears very efficient. This is the concept behind the rewrite rules and we follow through the procedure and form utilization of these rules in the following chapters.

Chapter 4

Rules for NRA

4.1 Methodology

As described in section 3.4, we start with the optimization schemes discussed in Leonidas Fegaras [5]. We write in NRA notation, our own rewrite rules that cover everything described by Fegaras's rules. We cover all operations and as exhaustive as possible.

We check the validity of the rules based on our semantics, after elimination of rules that are invalid in our definition of NRA, and adding new rules where ever needed we come up with full set to NRA as shown in section 4.2.1 and then we find the redundant rules, as shown in the following sections.

4.2 List of NRA Rules

In this section we show the full set and concise set of NRA. The full set includes both the derivable (subsumed by other rules in the set) and non-derivable (not subsumed by other rules in the set) rules and the concise set includes only the non-derivable rules.

4.2.1 Full Set of NRA Rewrite Rules

$$^{\lambda}c \circ f \equiv {}^{\lambda}c \tag{4.1}$$

$$\mathbf{id} \circ f \equiv f \tag{4.2}$$

$$f \circ \mathbf{id} \equiv f \tag{4.3}$$

$$(h \circ g) \circ f \equiv h \circ (g \circ f) \tag{4.4}$$

$$\mathbf{map}(f) \circ^{\lambda} \{\{\}\} \equiv^{\lambda} \{\{\}\} \tag{4.5}$$

$$^{\lambda}\{\{\}\} \circ f \equiv^{\lambda} \{\{\}\} \tag{4.6}$$

$$^{\lambda}\langle\rangle \circ f \equiv^{\lambda} \langle\rangle \tag{4.7}$$

$$\pi_1 \circ^{\lambda} \langle f, g\rangle \equiv f \tag{4.8}$$

$$\pi_2 \circ^{\lambda} \langle f, g\rangle \equiv g \tag{4.9}$$

$$^{\lambda}\langle g, h\rangle \circ f \equiv^{\lambda} \langle g \circ f, h \circ f\rangle \tag{4.10}$$

$$\mathbf{map}(g) \circ \mathbf{map}(f) \equiv \mathbf{map}(g \circ f) \tag{4.11}$$

$$^{\lambda}\uplus \circ^{\lambda} \langle f, g\rangle \equiv^{\lambda} \uplus \circ^{\lambda} \langle g, f\rangle \tag{4.12}$$

$$^{\lambda}\uplus \circ^{\lambda} \langle^{\lambda}\{\{\}\}, ^{\lambda}\{\{\cdot\}\} \circ f\rangle \equiv^{\lambda} \{\{\cdot\}\} \circ f \tag{4.13}$$

$$\mathbf{flat} \circ \mathbf{flat} \equiv \mathbf{flat} \circ \mathbf{map}(\mathbf{flat}) \tag{4.14}$$

$$\mathbf{map}(g) \circ \mathbf{flat} \equiv \mathbf{flat} \circ \mathbf{map}(\mathbf{map}(g)) \tag{4.15}$$

$$\mathbf{flat} \circ \mathbf{map}(^{\lambda}\{\{\cdot\}\}) \equiv \mathbf{map}(\mathbf{id}) \tag{4.16}$$

$$\mathbf{flat} \circ^{\lambda} \langle f, g\rangle \equiv^{\lambda} \bot \tag{4.17}$$

$$^{\lambda}\uplus \circ^{\lambda} \langle^{\lambda}\langle f, g\rangle, h\rangle \equiv^{\lambda} \bot \tag{4.18}$$

$$\mathbf{map}(f) \circ^{\lambda} c \equiv^{\lambda} \bot \tag{4.19}$$

$$^{\lambda}\times \circ^{\lambda} \langle f, ^{\lambda}\times \circ^{\lambda} \langle g, h\rangle\rangle \equiv \mathbf{map}(^{\lambda}\langle \pi_1 \circ \pi_1, ^{\lambda}\langle \pi_2 \circ \pi_1, \pi_2\rangle\rangle) \circ^{\lambda} \langle^{\lambda}\times \circ^{\lambda} \langle f, g\rangle, h\rangle \tag{4.20}$$

$$\mathbf{flat} \circ (^{\lambda}\uplus \circ^{\lambda} \langle f, g\rangle) \equiv^{\lambda} \uplus \circ^{\lambda} \langle \mathbf{flat} \circ f, \mathbf{flat} \circ g\rangle \tag{4.21}$$

$$^{\lambda}\uplus \circ^{\lambda} \langle f, ^{\lambda}\uplus \circ^{\lambda} \langle g, h\rangle\rangle \equiv^{\lambda} \uplus \circ^{\lambda} \langle^{\lambda}\uplus \circ^{\lambda} \langle f, g\rangle, h\rangle \tag{4.22}$$

$$\mathbf{flat} \circ \mathbf{map}(^{\lambda}\langle\rangle) \circ^{\lambda} \{\{\cdot\}\} \equiv^{\lambda} \bot \tag{4.23}$$

$$\mathbf{flat} \circ \mathbf{map}(^{\lambda}\langle f, g\rangle) \circ^{\lambda} \{\{\cdot\}\} \equiv^{\lambda} \bot \tag{4.24}$$

$$\mathbf{map}(h) \circ (^{\lambda}\uplus \circ^{\lambda} \langle f, g\rangle) \equiv^{\lambda} \uplus \circ^{\lambda} \langle \mathbf{map}(h) \circ f, \mathbf{map}(h) \circ g\rangle \tag{4.25}$$

$$^{\lambda}\times \circ^{\lambda} \langle ^{\lambda}\langle f, g\rangle, h\rangle \equiv^{\lambda} \bot \tag{4.26}$$

$$\mathbf{flat} \circ^{\lambda} \langle\rangle \equiv^{\lambda} \bot \tag{4.27}$$

$$^{\lambda}\uplus \circ^{\lambda} \langle ^{\lambda}c, f\rangle \equiv^{\lambda} \bot \tag{4.28}$$

$$\pi_1 \circ^{\lambda} \langle\rangle \equiv^{\lambda} \bot \tag{4.29}$$

$$\pi_2 \circ^{\lambda} \langle\rangle \equiv^{\lambda} \bot \tag{4.30}$$

$$\pi_1 \circ^{\lambda} c \equiv^{\lambda} \bot \tag{4.31}$$

$$\pi_2 \circ^{\lambda} c \equiv^{\lambda} \bot \tag{4.32}$$

$$\pi_1 \circ \mathbf{map}(f) \equiv^{\lambda} \bot \tag{4.33}$$

$$\pi_2 \circ \mathbf{map}(f) \equiv^{\lambda} \bot \tag{4.34}$$

$$\mathbf{map}(f) \circ^{\lambda} \langle\rangle \equiv^{\lambda} \bot \tag{4.35}$$

$$\mathbf{map}(f) \circ^{\lambda} \{\{\cdot\}\} \equiv^{\lambda} \{\{\cdot\}\} \circ f \tag{4.36}$$

$$\mathbf{flat} \circ^{\lambda} c \equiv^{\lambda} \bot \tag{4.37}$$

$$\mathbf{flat} \circ \mathbf{map}(^{\lambda}c) \circ^{\lambda} \{\{\cdot\}\} \equiv^{\lambda} \bot \tag{4.38}$$

$$^{\lambda}\times \circ^{\lambda} \langle ^{\lambda}\{\{\cdot\}\} \circ f, ^{\lambda}\{\{\cdot\}\} \circ g\rangle \equiv^{\lambda} \{\{\cdot\}\} \circ^{\lambda} \langle f, g\rangle \tag{4.39}$$

$$^{\lambda}\times \circ^{\lambda} \langle f, ^{\lambda}\uplus \circ^{\lambda} \langle g, h\rangle\rangle \equiv^{\lambda} \uplus \circ^{\lambda} \langle ^{\lambda}\times \circ^{\lambda} \langle f, g\rangle, ^{\lambda}\times \circ^{\lambda} \langle f, h\rangle\rangle \tag{4.40}$$

$$^{\lambda}\times \circ^{\lambda} \langle \mathbf{map}(g) \circ f, h\rangle \equiv \mathbf{map}(^{\lambda}\langle g \circ \pi_1, \pi_2\rangle) \circ^{\lambda} \times \circ^{\lambda} \langle f, h\rangle \tag{4.41}$$

$$^{\lambda}\times \circ^{\lambda} \langle f, g\rangle \equiv \mathbf{map}(^{\lambda}\langle \pi_2, \pi_1\rangle) \circ (^{\lambda}\times \circ^{\lambda} \langle g, f\rangle) \tag{4.42}$$

$$^{\lambda}\times \circ\, ^{\lambda}\langle \mathbf{flat}\circ f, h\rangle \equiv \mathbf{flat}\circ \mathbf{map}(^{\lambda}\times \circ\, ^{\lambda}\langle \pi_1, ^{\lambda}\{\{\cdot\}\}\circ \pi_2\rangle)\circ\, ^{\lambda}\times \circ\, ^{\lambda}\langle f, h\rangle \tag{4.43}$$

$$\mathbf{flat}\circ \mathbf{map}(^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\{\{\cdot\}\}, g\circ \pi_1\rangle)\circ\, ^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\{\{\cdot\}\}, f\rangle \equiv\, ^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\{\{\cdot\}\}, f\rangle, g\rangle \tag{4.44}$$

$$\frac{\mathbf{map}(\mathbf{id})\circ f \equiv f}{\mathbf{map}(\pi_1)\circ\, ^{\lambda}\times \circ\, ^{\lambda}\langle f, f\rangle \equiv f} \tag{4.45}$$

$$^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\{\{\cdot\}\}\circ f, h\rangle \equiv \mathbf{map}(^{\lambda}\langle f\circ \pi_1, \pi_2\rangle)\circ (^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\{\{\cdot\}\}, h\rangle) \tag{4.46}$$

$$^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\uplus \circ\, ^{\lambda}\langle f, g\rangle, h\rangle \equiv\, ^{\lambda}\uplus \circ\, ^{\lambda}\langle ^{\lambda}\times \circ\, ^{\lambda}\langle f, h\rangle, ^{\lambda}\times \circ\, ^{\lambda}\langle g, h\rangle\rangle \tag{4.47}$$

$$\mathbf{flat}\circ\, ^{\lambda}\{\{\}\} \equiv\, ^{\lambda}\{\{\}\} \tag{4.48}$$

$$\mathbf{map}(f)\circ\, ^{\lambda}\langle f, g\rangle \equiv\, ^{\lambda}\bot \tag{4.49}$$

$$^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\langle\rangle, f\rangle \equiv\, ^{\lambda}\bot \tag{4.50}$$

$$^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}f, g\rangle \equiv\, ^{\lambda}\bot \tag{4.51}$$

4.2.2 Concise Set: Non-Derivable Rewrite Rules of NRA

The rules from 4.1 to 4.45 form the concise set of NRA. The concise set of NRA is the minimal non-redundant rewrite rule set and no rule within this set can be derived from other rules. This derivability was checked using Prover 9 and thus we obtained the concise/non-derivable set of NRA rewrite rules. In Chapter 5 we validate the results of Prover 9 by mathematically proving the derivable rules.

Claim: *The presented concise set of NRA is a minimal set of rewrite rules for NRA, i.e., none of these rules can be derived using any of the other rules in this set.*

Chapter 5

Proof of Derivability for Rewrite Rules of NRA

In this chapter, we only prove the redundancy of the rewrite rules present in the full set of NRA, in terms of the rewrite rules in the concise set of NRA defined in chapter 4. By redundant, we mean that these rewrite rules can be derived, considering all the other rules in the concise set of NRA defined in the chapter 4.

We initially use Prover 9 to do a rough analysis, while arriving at the concise set, but here we verify logically whether Prover 9 shows accurate results. Also, we present a logical, more human readable proof that is easy to understand and explains our approach of eliminating redundancies in an even better and more mathematical way. To refer to the way of proving that Prover 9 uses refer to the Appendix C.

Claim: *Concise Set of NRA* $\equiv$ *Full Set of NRA*

5.1 Derivable Rewrite Rules

This section gives the list of rules which could be derived using some of the rules of the concise set mentioned in the Chapter 4.

$$^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\{\{\cdot\}\} \circ f, h\rangle \equiv \mathbf{map}(^{\lambda}\langle f \circ \pi_1, \pi_2\rangle) \circ (^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\{\{\cdot\}\}, h\rangle) \tag{4.46}$$

$$^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\uplus \circ\, ^{\lambda}\langle f,g\rangle,h\rangle \equiv\, ^{\lambda}\uplus \circ\, ^{\lambda}\langle ^{\lambda}\times \circ\, ^{\lambda}\langle f,h\rangle,\, ^{\lambda}\times \circ\, ^{\lambda}\langle g,h\rangle\rangle \tag{4.47}$$

$$\mathbf{flat} \circ\, ^{\lambda}\{\{\}\} \equiv\, ^{\lambda}\{\{\}\} \tag{4.48}$$

$$\mathbf{map}(f) \circ\, ^{\lambda}\langle f,g\rangle \equiv\, ^{\lambda}\perp \tag{4.49}$$

$$^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\langle\rangle, f\rangle \equiv\, ^{\lambda}\perp \tag{4.50}$$

$$^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda} f,g\rangle \equiv\, ^{\lambda}\perp \tag{4.51}$$

The mathematical proofs for the above mentioned rules are mentioned in the section 5.2. The proofs for rules 4.49 through 4.51 are not presented here because of their length and complexity but they were verified using a theorem prover Prover 9.

5.2 Proofs

5.2.1 Proof I

$$^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\{\{\cdot\}\} \circ f,h\rangle \equiv \mathbf{map}(^{\lambda}\langle f\circ\pi_1,\pi_2\rangle) \circ (^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\{\{\cdot\}\},h\rangle) \tag{4.46}$$

To prove L.H.S = R.H.S.

L.H.S. = $^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\{\{\cdot\}\} \circ f,h\rangle$

= $^{\lambda}\times \circ\, ^{\lambda}\langle \mathbf{map}(f) \circ\, ^{\lambda}\{\{\cdot\}\},h\rangle$ from (4.36).

= $\mathbf{map}(^{\lambda}\langle f\circ\pi_1,\pi_2\rangle) \circ\, ^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\{\{\cdot\}\},h\rangle$ from (4.41).

L.H.S. is equivalent to R.H.S. Hence Proved.

5.2.2 Proof II

$$^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\uplus \circ\, ^{\lambda}\langle f,g\rangle,h\rangle \equiv\, ^{\lambda}\uplus \circ\, ^{\lambda}\langle ^{\lambda}\times \circ\, ^{\lambda}\langle f,h\rangle,\, ^{\lambda}\times \circ\, ^{\lambda}\langle g,h\rangle\rangle \tag{4.47}$$

To prove L.H.S = R.H.S.

L.H.S. = $^{\lambda}\times \circ\, ^{\lambda}\langle ^{\lambda}\uplus \circ\, ^{\lambda}\langle f,g\rangle,h\rangle$

= $\mathbf{map}(^{\lambda}\langle \pi_2,\pi_1\rangle) \circ (^{\lambda}\times \circ\, ^{\lambda}\langle h,\, ^{\lambda}\langle f,g\rangle\rangle)$ from (4.42).

$= \quad \mathbf{map}(^{\lambda}\langle \pi_2, \pi_1 \rangle) \circ (^{\lambda}\uplus \circ {}^{\lambda}\langle {}^{\lambda}\times \circ {}^{\lambda}\langle h, f\rangle, {}^{\lambda}\times \circ {}^{\lambda}\langle h, g\rangle\rangle)$ from (4.40).

$= \quad {}^{\lambda}\uplus \circ {}^{\lambda}\langle \mathbf{map}(^{\lambda}\langle \pi_2, \pi_2\rangle \circ {}^{\lambda}\times \circ {}^{\lambda}\langle h, f\rangle), \mathbf{map}(^{\lambda}\langle \pi_2, \pi_1\rangle \circ {}^{\lambda}\times \circ {}^{\lambda}\langle h, g\rangle)\rangle$ from (4.25).

$= \quad {}^{\lambda}\uplus \circ {}^{\lambda}\langle {}^{\lambda}\times \circ {}^{\lambda}\langle f, h\rangle, {}^{\lambda}\times \circ {}^{\lambda}\langle g, h\rangle\rangle$ from (4.42).

L.H.S. is equivalent to R.H.S. Hence Proved.

5.2.3 Proof III

$$\mathbf{flat} \circ {}^{\lambda}\{\{\}\} \equiv {}^{\lambda}\{\{\}\} \tag{4.48}$$

To prove L.H.S = R.H.S.

L.H.S. = $\mathbf{flat} \circ {}^{\lambda}\{\{\}\}$

$= \quad \mathbf{flat} \circ \mathbf{map}(^{\lambda}\{\{\cdot\}\}) \circ {}^{\lambda}\{\{\}\}$ from (4.5).

$= \quad \mathbf{map}(\mathbf{id}) \circ {}^{\lambda}\{\{\}\}$ from (4.16).

$= \quad {}^{\lambda}\{\{\}\}$ from (4.5).

L.H.S. is equivalent to R.H.S. Hence Proved.

Chapter 6

Rules for NRC

6.1 Methodology

In order to build the full set of NRC rewrite rules, as a starting point, we take each rule of NRA and transform it using the mapping, as presented in Chapter 3. In addition, we write a lot of unique NRC rules to formulate this full set of NRC.

After elimination of the rules that are invalid in our definition of NRC, we come up with the full set of the rewrite rules. Following this we eliminate redundancies, as explained in the Chapter 7, and come up with the concise set of NRC.

6.2 Full Set of NRC Rewrite Rules

In this section, the complete set of NRC equations is given. The variables shown in green are NRC variables. The expression variables are shown in blue and the basic value constant is represented in black. In the following set, we have introduced the concept of tail as an independent expression, including the empty tail. Moreover, the singleton bag expression has been removed and is replaced with ε.

$$x := e_1 ; x \equiv e_1 \tag{6.1}$$

$$\frac{x \neq y}{x := e_1 ; y \equiv y} \tag{6.2}$$

$$x := e_1 ; c \equiv c \tag{6.3}$$

$$x := e_1 ; \bot \equiv \bot \tag{6.4}$$

$$x := e_1 \,;\, \langle e_2, e_3 \rangle \equiv \langle x := e_1 \,;\, e_2, x := e_1 \,;\, e_3 \rangle \tag{6.5}$$

$$x := e_1 \,;\, e_2.1 \equiv (x := e_1 \,;\, e_2).1 \tag{6.6}$$

$$x := e_1 \,;\, e_2.2 \equiv (x := e_1 \,;\, e_2).2 \tag{6.7}$$

$$x := e_1 \,;\, \{\{\}\} \equiv \{\{\}\} \tag{6.8}$$

$$x := e_1 \,;\, \langle\rangle \equiv \langle\rangle \tag{6.9}$$

$$x := e_1 \,;\, e_2 \uplus e_3 \equiv (x := e_1 \,;\, e_2) \uplus (x := e_1 \,;\, e_3) \tag{6.10}$$

$$(\langle e_1, e_2 \rangle).1 \equiv e_1 \tag{6.11}$$

$$(\langle e_1, e_2 \rangle).2 \equiv e_2 \tag{6.12}$$

$$\langle\rangle.1 \equiv \bot \tag{6.13}$$

$$\{\{\}\}.1 \equiv \bot \tag{6.14}$$

$$(\{\{e_1 \mid e_2\}\}).1 \equiv \bot \tag{6.15}$$

$$(e_1 \uplus e_2).1 \equiv \bot \tag{6.16}$$

$$c.1 \equiv \bot \tag{6.17}$$

$$\bot.1 \equiv \bot \tag{6.18}$$

$$\langle\rangle.2 \equiv \bot \tag{6.19}$$

$$\{\{\}\}.2 \equiv \bot \tag{6.20}$$

$$(\{\{e_1 \mid e_2\}\}).2 \equiv \bot \tag{6.21}$$

$$(e_1 \uplus e_2).2 \equiv \bot \tag{6.22}$$

$$c.2 \equiv \bot \tag{6.23}$$

$$\bot.2 \equiv \bot \tag{6.24}$$

$$e_1 \uplus e_2 \equiv e_2 \uplus e_1 \tag{6.25}$$

$$e_1 \uplus (e_2 \uplus e_3) \equiv (e_1 \uplus e_2) \uplus e_3 \tag{6.26}$$

$$\langle e_1, e_2 \rangle \uplus e_3 \equiv \bot \tag{6.27}$$

$$\langle\rangle \uplus e_1 \equiv \bot \tag{6.28}$$

$$c \uplus e_1 \equiv \bot \tag{6.29}$$

$$e_1 \uplus \bot \equiv \bot \tag{6.30}$$

$$\frac{(e_1 \equiv (\{\{x \mid x \; in \; e_1\}\}))}{e_1 \uplus e_1 \equiv e_1} \tag{6.31}$$

$$\{\{e_1 \mid x \; in \; \langle e_2, e_3 \rangle \; , \; e_4\}\} \equiv \bot \tag{6.32}$$

$$\{\{e_1 \mid x \; in \; c \; , \; e_2\}\} \equiv \bot \tag{6.33}$$

$$\{\{e_1 \mid x \; in \; \langle\rangle \; , \; e_2\}\} \equiv \bot \tag{6.34}$$

$$\{\{e_1 \mid x \; in \; \bot \; , \; e_2\}\} \equiv \bot \tag{6.35}$$

$$x := e_1 \; ; (\{\{e_2 \mid e_3\}\}) \equiv (\{\{e_2 \mid (x := e_1 \; ; e_3\}\})) \tag{6.36}$$

$$x := e_1 \; , \; \varepsilon \equiv x := e_1 \tag{6.37}$$

$$\varepsilon \; , \; x := e_1 \equiv x := e_1 \tag{6.38}$$

$$\varepsilon \; , \; \varepsilon \equiv \varepsilon \tag{6.39}$$

$$\frac{x \neq y}{x := e_1 \; ; y := e_2 \equiv y := (x := e_1 \; ; e_2) \; ; x := e_1} \tag{6.40}$$

$$x := e_1 \; ; x := e_2 \equiv x := (x := e_1 \; ; e_2) \tag{6.41}$$

$$(s \; , \; t) \; , \; u \equiv s \; , \; (t \; , \; u) \tag{6.42}$$

$$\{\{e_1 \mid t\ ,\ x := e_2\}\} \equiv \{\{x := e_2\ in\ e_1 \mid t\}\} \tag{6.43}$$

$$x := x\ ;\ e_1 \equiv e_1 \tag{6.44}$$

$$x := \langle\rangle\ ;\ c \equiv c \tag{6.45}$$

$$x := c\ ;\ \langle\rangle \equiv \langle\rangle \tag{6.46}$$

$$\varepsilon\ ,\ x\ in\ e_1 \equiv x\ in\ e_1 \tag{6.47}$$

$$x\ in\ e_1\ ,\ \varepsilon \equiv x\ in\ e_1 \tag{6.48}$$

$$\frac{-det(y\ ,\ e_1)\ \&\ x \neq y}{x := e_1\ ;\ y := e_2 \equiv y := (x := e_1\ ;\ e_2)\ ;\ (x := e_1)} \tag{6.49}$$

$$\frac{-det(x\ ,\ e_1)}{x := e_1\ ;\ x := e_2 \equiv x := (x := e_1\ ;\ e_2)} \tag{6.50}$$

$$\frac{-det(x\ ,\ e_1)\ \&\ x \neq y}{x := e_1\ ;\ y\ in\ e_2 \equiv y\ in\ (x := e_1\ ;\ e_2)\ ;\ (x := E)} \tag{6.51}$$

$$\frac{-det(x\ ,\ e_1)}{x := e_1\ ;\ x\ in\ e_2 \equiv x\ in\ (x := e_1\ ;\ e_2)} \tag{6.52}$$

$$\frac{-det(x\ ,\ e_1)}{y := e_2\ ;\ e_1 \equiv x := e_2\ ;\ y := x\ ;\ e_1} \tag{6.53}$$

$$\frac{-det(x\ ,\ e_1)}{y\ in\ e_2\ ;\ e_1 \equiv x\ in\ e_2\ ;\ y := x\ ;\ e_1} \tag{6.54}$$

$$x := (\{\{(x := y\ ;e_1) \mid y\}\}\ in\ x)\ ;\ x.1 \equiv \bot \tag{6.55}$$

$$x := \langle\rangle\ ;\ x.1 \equiv \bot \tag{6.56}$$

$$x := \langle\langle e_1, e_2\rangle, e_3\rangle\ ;\ x.1 \uplus x.2 \equiv \bot \tag{6.57}$$

$$(\{\{e_1 \mid x\ in\ e_2\ ,\ e_3\}\}) \equiv (\{\{e_1 \mid x := e_2\ ;\ e_3\}\}) \tag{6.58}$$

$$\{\{e_1 \mid e_2\ ,\ x\ in\ e_3\ ,\ e_4\}\} \equiv \{\{e_1 \mid e_2\ ,\ x := e_3\ ;\ e_4\}\} \tag{6.59}$$

$$\{\{e_1 \mid t_1\ ,\ x\ in\ \{\{e_2\}\}\ ,\ t_2\}\} \equiv \{\{e_1 \mid t_1\ ,\ x := e_2\ ;\ t_2\}\} \tag{6.60}$$

$$\{\{e_1 \mid x \; in \; \{\{e_2\}\} \; , \; t_2\}\} \equiv \{\{e_1 \mid x \; := \; e_2 \; ; \; t_2\}\} \tag{6.61}$$

$$x \; := \; \langle e_1, e_2 \rangle \; ; \; (\{\{(x \; := y \; ; \; e_3) \mid y \; in \; x\}\}) \equiv \bot \tag{6.62}$$

$$x \; := \; c \; ; \; (\{\{(x \; := \; y \; ; \; e_1) \mid y \; in \; x\}\}) \equiv \bot \tag{6.63}$$

$$x \; := \; \langle\rangle \; ; \; (\{\{(x \; := \; y \; ; \; e_1) \mid y \; in \; x\}\}) \equiv \bot \tag{6.64}$$

$$x \; := \; \langle e_1, e_2 \rangle \; ; \; (\{\{z \mid y \; in \; x, z \; in \; y\}\}) \equiv \bot \tag{6.65}$$

$$x \; := \; c \; ; \; (\{\{z \mid y \; in \; x, z \; in \; y\}\}) \equiv \bot \tag{6.66}$$

$$x \; := \; \langle\rangle \; ; \; (\{\{z \mid y \; in \; x, z \; in \; y\}\}) \equiv \bot \tag{6.67}$$

$$x \; := \; (\{\{(x \; := \; y \; ; \; \langle e_1, e_2 \rangle) \mid y \; in \; x\}\}) \; ; \; (\{\{z \mid y \; in \; x, z \; in \; y\}\}) \equiv \bot \tag{6.68}$$

$$x \; := \; (\{\{(x \; := \; y \; ; \; e_1) \mid y \; in \; x\}\}) \; ; \; (\{\{z \mid y \; in \; x, z \; in \; y\}\}) \equiv \bot \tag{6.69}$$

$$x \; := \; (\{\{(x \; := \; y \; ; \; \langle\rangle) \mid y \; in \; x\}\}) \; ; \; (\{\{z \mid y \; in \; z, z \; in \; y\}\}) \equiv \bot \tag{6.70}$$

$$x \; := \; \langle\langle e_1, e_2 \rangle, e_3 \rangle \; ; \; (\{\{\langle y, z \rangle \mid y \; in \; x.1\}\}, z \; in \; x.2) \equiv \bot \tag{6.71}$$

$$x \; := \; \langle c, e_1 \rangle \; ; \; (\{\{\langle y, z \rangle \mid y \; in \; x.1\}\}, z \; in \; x.2) \equiv \bot \tag{6.72}$$

$$x \; := \; \langle\langle\rangle, e_1 \rangle \; ; \; \{\{\langle y, z \rangle \mid y \; in \; x.1, z \; in \; x.2\}\} \equiv \bot \tag{6.73}$$

$$\begin{gathered} x \; := \; \langle e_1, e_2 \rangle \; ; \; \{\{(x \; := \; y \; ; \; x.1 \mid y \; in \; x))\}\} \equiv \\ x \; := \; \langle \{\{(x \; := \; y \; ; \; e_1 \mid y \; in \; x\}\}), (x \; := \; y \; ; \; \{\{e_2 \mid y \; in \; x\}\}) \rangle \; ; \; x.1 \end{gathered} \tag{6.74}$$

$$x \; := \; \langle e_1, e_2 \rangle \; ; \; \{\{z \mid y \; in \; x \; , \; z \; in \; y\}\} \equiv \bot \tag{6.75}$$

$$x \; := \; \langle c, e_1 \rangle ; x.1 \uplus x.2 \equiv \bot \tag{6.76}$$

$$x \; := \; \langle\langle\rangle, e_1 \rangle ; x.1 \uplus x.2 \equiv \bot \tag{6.77}$$

$$(x \; := \; \langle e_1, e_2 \rangle \; ; \; x.1) \equiv e_1 \tag{6.78}$$

$$(x := \langle e_1, e_2 \rangle \,;\, x.2) \equiv e_2 \tag{6.79}$$

$$\langle e_1, x := \langle e_2, e_3 \rangle \,;\, x.1 \rangle \equiv x := \langle \langle e_1, e_2 \rangle, e_3 \rangle \,;\, x.1 \tag{6.80}$$

$$\langle x := \langle e_1, e_2 \rangle \,;\, x.1, e_3 \rangle \equiv x := \langle \langle e_1, e_3 \rangle, e_2 \rangle \,;\, x.1 \tag{6.81}$$

$$\begin{array}{c} x := \langle\rangle \,;\, (x := e_1 \,;\, (x := \langle\rangle \,;\, (x := e_2 \,;\, \langle\rangle))) \equiv \\ x := \langle\rangle; x := e_2 \,;\, (x := \langle\rangle \,;\, (x := e_1 \,;\, \langle\rangle))) \end{array} \tag{6.82}$$

$$x := \langle e_1, e_1 \rangle \,;\, \langle\rangle \equiv x := e_1 \,;\, \langle\rangle \tag{6.83}$$

$$x := \langle e_1, e_2 \rangle \,;\, \langle\rangle \equiv x := \langle x := e_2; \langle\rangle, e_1 \rangle; \langle\rangle \tag{6.84}$$

$$x := \langle e_1, e_2 \rangle \,;\, \langle\rangle \equiv x := \langle e_2, x := e_1; \langle\rangle \rangle; \langle\rangle \tag{6.85}$$

$$x := \langle e_1, e_2 \rangle \,;\, \langle\rangle \equiv x := \langle e_2, e_1 \rangle; \langle\rangle \tag{6.86}$$

$$x := \langle e_1, \langle e_2, e_3 \rangle \rangle \,;\, \langle\rangle \equiv x := \langle \langle e_1, e_2 \rangle, e_3 \rangle \,;\, \langle\rangle \tag{6.87}$$

$$x := x.2 \,;\, \langle\rangle \equiv x := x.1 \,;\, \langle\rangle \tag{6.88}$$

$$x := e_1 \,;\, c \equiv x := \langle c, e_1 \rangle \,;\, x.1 \tag{6.89}$$

$$x := \langle\rangle \,;\, \langle\rangle \equiv x := x := \,;\, \langle\rangle \tag{6.90}$$

$$x := x.2 \,;\, \langle\rangle \equiv x := x.1 \,;\, \langle\rangle \tag{6.91}$$

$$x := \langle e_1, \langle e_2, e_1 \rangle \rangle \,;\, \langle\rangle \equiv x := \langle e_2, e_1 \rangle \,;\, \langle\rangle \tag{6.92}$$

$$x := \langle e_1, e_2 \rangle \,;\, x.2 \equiv x := \langle e_2, e_1 \rangle \,;\, x.1 \tag{6.93}$$

$$x := (\{\{(x := y \,;\, e_1) \mid y \; in \; x\}\}) \,;\, x.1 \equiv \bot \tag{6.94}$$

$$x := \langle x := \langle e_1, e_2 \rangle \,;\, x.1, e_3 \rangle; x.1 \equiv x := \langle e_1, \langle e_2, e_3 \rangle \rangle \,;\, x.1 \tag{6.95}$$

$$x := \langle e_1, x := e_1 \,;\, e_2 \rangle \,;\, \langle\rangle \equiv x := x := e_1 \,;\, e_2 \,;\, \langle\rangle \tag{6.96}$$

$$x := \langle e_1, \langle e_1, e_2 \rangle \rangle \; ; \; \langle \rangle \equiv x := \langle e_1, e_2 \rangle \; ; \; \langle \rangle \tag{6.97}$$

$$x := \langle e_1, x := \langle e_1, e_2 \rangle \; ; \; x.1 \rangle \; ; \; x.1 \equiv x := \langle e_1, \langle e_3, e_2 \rangle \rangle \; ; \; x.1 \tag{6.98}$$

$$x := \langle e_1, e_2 \rangle \; ; \; x.1 \equiv x := \langle e_2, e_1 \rangle ; x.2 \tag{6.99}$$

$$x := \langle e_1, x := e_1 \; ; \; e_2 \rangle \; ; \; \langle \rangle \equiv x := x := e_1 \; ; \; e_2 \; ; \; \langle \rangle \tag{6.100}$$

$$x := \langle e_1, \langle e_1, e_2 \rangle \rangle \; ; \; \langle \rangle \equiv x := \langle e_1, e_2 \rangle \; ; \; \langle \rangle \tag{6.101}$$

$$x := \langle e_1, \langle e_2, e_1 \rangle \rangle \; ; \; \langle \rangle \equiv x := \langle e_2, e_1 \rangle \; ; \; \langle \rangle \tag{6.102}$$

$$x := \langle \rangle \; ; \; \langle \rangle \equiv x := x \; ; \; \langle \rangle \tag{6.103}$$

6.3 Auxiliary rules for NRC

We have used mainly Prover 9 (Mace 4 in some circumstances only), to analyse and verify the rules. Mace4 searches for finite models and counterexamples. It helped in fine tuning the rules, and removing inconsistencies. It also assists in checking for redundancy in the rules. In a few cases, where proofs of redundancy were very complicated and considerably large we have relied on Prover 9 for answers.

To explain the rules to Prover 9 and to be able to express the rules in it, we had to come up with Prover 9 notations for NRC. In fact, not only the notation, especially in cases of NRC, since the rules have very different kind of operations like comprehension, substitution etc., we had to come up with auxiliary rules for NRC. These rules explain the true meaning of these operations to NRC. Care was taken to define symbols correctly and assign appropriate priorities to the various operators that we used.

We now list the notion of variables used for NRC, and briefly describe the meaning of the notion or the rationale for some of these symbols used.

6.3.1 Notion of Variables

In Prover 9, the representation of mathematical equations is primarily done by equations between terms which consist only of variables, constants and function symbols. In order to

represent NRC variables in Prover 9 we propose a convention given below.

We use the constant $v0$ and unary function $v()$ to represent variables. This gives us expressions like the following : $v0$, $v(v0)$, $v(v(v0))$, $v(v(v(v0)))$, ...

A translation that tells us how NRC variables are represented is shown below:

$$x \Rightarrow v0$$
$$y \Rightarrow v(v0)$$
$$z \Rightarrow v(v(v0))$$

Thus, the expression x.1 is represented as $\pi_1(v0)$ and y.1 as $\pi_1(v(v0))$. In this way every NRC expression can be represented in a prover 9 expression without any open variables, as we see in the following rules.

$$\frac{v(x) \equiv v(y)}{x \equiv y} \tag{6.104}$$

$$\frac{x \equiv y}{v(x) \equiv v(y)} \tag{6.105}$$

$$v0 \neq v(x) \tag{6.106}$$

6.3.2 Notion of *det*

det is defined as "has a free variable".
$x := e_1$, it is defined as usual, i.e., x is a free variable in e_1.

6.3.3 Auxiliary Rules

$$\frac{det(x\ ,\ \bot)}{\$F} \tag{6.107}$$

$$\frac{det(x\ ,\ \{\{\}\})}{\$F} \tag{6.108}$$

$$\frac{det(x\ ,\ \langle\rangle)}{\$F} \tag{6.109}$$

$$\frac{det(x\ ,\ c)}{\$F} \tag{6.110}$$

$$det(x\ ,\ x) \tag{6.111}$$

$$\frac{x \neq y\ \&\ det(x\ ,\ y)}{\$F} \tag{6.112}$$

$$\frac{det(x\ ,\ e_1.1)}{det(x\ ,\ e_1)} \tag{6.113}$$

$$\frac{det(x\ ,\ e_1.2)}{det(x\ ,\ e_1)} \tag{6.114}$$

$$\frac{det(x\ ,\ (e_1 \uplus e_2))}{det(x\ ,\ e_1)\ |\ det(x\ ,\ e_2)} \tag{6.115}$$

$$\frac{z \neq x\ \&\ det(z\ ,\ (x\ :=\ e_1\ ;\ e_2))}{det(z\ ,\ e_1)\ |\ det(z\ ,\ e_2)} \tag{6.116}$$

$$\frac{det(x\ ,\ (x\ :=\ e_1\ ;\ e_2))}{det(x\ ,\ e_1)} \tag{6.117}$$

$$\frac{det(x\ ,\ (\{\{e_1\ |\ \varepsilon\}\}))}{det(x\ ,\ e_1)} \tag{6.118}$$

$$\frac{x \neq y\ \&\ det(x\ ,\ (\{\{e_1\ |\ (y\ :=\ e_2\ ;\ e_3)\}\}))}{det(x\ ,\ (e_2))\ |\ det(x\ ,\ (\{\{e_1\ |\ e_3\}\}))} \tag{6.119}$$

$$\frac{det(x\ ,\ (\{\{e_1\ |\ (x\ :=\ e_2\ ;\ e_3)\}\}))}{det(x\ ,\ (e_2))} \tag{6.120}$$

$$\frac{x \neq y\ \&\ det(x\ ,\ (\{\{e_1\ |\ (y\ in\ e_2\ ;\ e_3)\}\}))}{det(x\ ,\ (e_2))\ |\ det(x\ ,\ (\{\{e_1\ |\ e_3\}\}))} \tag{6.121}$$

$$\frac{det(x\ ,\ (\{\{e_1\ |\ (x\ in\ e_2\ ;\ e_3)\}\}))}{det(x\ ,\ (e_2))} \tag{6.122}$$

6.4 Concise Set: Non-Derivable Set of NRC Rewrite Rules

The rules from 6.1 to 6.54 form the concise set of NRC. The concise set of NRC is the minimal non-redundant rewrite rule set and no rule within this set can be derived using other rules. This derivability was checked using Prover 9, with the help of auxiliary rules and thus we obtained the concise/non-derivable set of NRC rewrite rules.

Claim: *The presented concise set of NRC is a minimal set of rewrite rules for NRC, i.e., none of these rules can be derived using any of the other rules in this set.*

Chapter 7

Equivalence of NRA & NRC Ruleset

In this chapter we present the equivalence proof for NRC and NRA rule set, consisting of basic values, pairs and tuples. The NRC rule set R_1 is given in section 7.3.2 and NRA rule set R_2 is given in section 7.3.1.

7.1 Theorem 1 : Relationship of NRC to NRA

Theorem 1 :*NRC rule set R_1 is subsumed by NRA rule set R_2 if any two expressions e_1 and e_2, such that e_1 can be rewritten to e_2 using the rules of R_1 and applying them only at the top level of an expression.*

7.1.1 Proof

Given NRA with M'(e_1) and M'(e_2), as the corresponding mapped versions of e_1 and e_2 from NRC.

Prove that there exists a finite number of steps to rewrite from M'(e_1) to M'(e_2) .

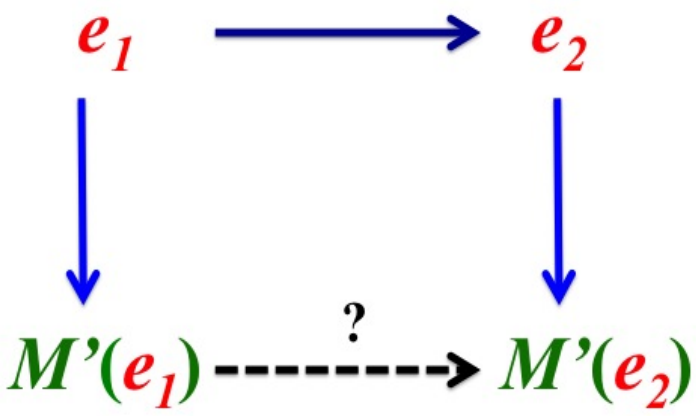


Fig. 7.1 Relationship of NRC and NRA

We prove the theorem using induction on the number of steps it takes to rewrite e_1 to e_2. We will prove that the theorem holds for $n = 1$ in the first step, where n is number of steps it takes to rewrite from e_1 to e_2. And then, that it if holds for n and smaller then it also holds for $n+1$ in the next step of induction.

Step 1 of Induction

- In this step, we show that the rewrite rules that transform e_1 to e_2, also have corresponding rewrite rule(s) in NRA.
- For proving this, we transform the rewrite rules of NRC, one by one, to the rewrite rules of NRA. After each transformation we check whether this transformed rewrite rule already has a corresponding rewrite rule in the other language or not.
- If it does have a corresponding rewrite rule in the other language we can easily use that rewrite M'(e_1) to M'(e_2) also.
- If it does not have a corresponding rewrite rule in the other we check whether this transformed rewrite rule can be expressed or deduced using a set of already existing rewrite rules in the NRA.
- If we can establish this for all our existing rules, we can say that any M'(e_1) can be re written to M'(e_2) .

Step n of Induction

Assume that e_1 --→ $e_n + 1$.
Definition:- e_i --→ e_j *denotes that* e_i *can be rewritten to* e_j
Then it follows that e_1 --→ e_n and e_n --→ e_{n+1}. Both paths have length n or smaller. So, by induction we know that M'(e_1) --→ M'(e_{n+1}) as in ODMG [11].

7.2 Theorem 2 : Relationship of NRA to NRC

Theorem 2 : *NRA rule set R_2 which is subsumed by NRC rule set R_1 if any two expressions e_1 and e_2, such that e_1 can be rewritten to e_2 using the rules in R_2 and applying them only at the top level of an expression.*

7.2.1 Proof

Given, NRA with strings e_1 and e_2, such that e_1 can be rewritten to e_2.
Given NRC, with M(e_1) and M(e_2), as the corresponding mapped versions of e_1 and e_2 from NRA.

Prove that there exists a finite number of steps to rewrite from M(e_1) to M(e_2).

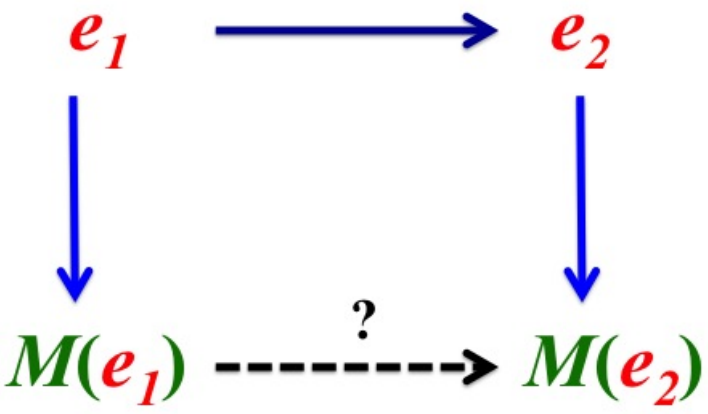


Fig. 7.2 Relationship of NRA and NRC

We prove the theorem using induction on the number of steps it takes to rewrite e_1 to e_2. We will prove that the theorem holds for $n = 1$ in the first step, where n is number of steps

it takes to rewrite from e_1 to e_2. And then, that it if holds for n and smaller then it also holds for $n+1$ in the next step of induction.

Step 1 of Induction

- In this step, we show that the rewrite rules that transform e_1 to e_2, also have corresponding rewrite rule(s) in NRC.
- For proving this, we transform the rewrite rules of NRA, one by one, to the rewrite rules of NRC. After each transformation we check whether this transformed rewrite rule already has a corresponding rewrite rule in the other language or not.
- If it does have a corresponding rewrite rule in the other language we can easily use that rewrite M(e_1) to M(e_2) also.
- If it does not have a corresponding rewrite rule in the other we check whether this transformed rewrite rule can be expressed or deduced using a set of already existing rewrite rules in the NRC.
- If we can establish this for all our existing rules, we can say that any M(e_1) can be re written to M(e_2) .

Step n of Induction

Assume that $e_1 \dashrightarrow e_n + 1$.
Definition:- $e_i \dashrightarrow e_j$ *denotes that* e_i *can be rewritten to* e_j
Then it follows that $e_1 \dashrightarrow e_n$ and $e_n \dashrightarrow e_{n+1}$. Both paths have length n or smaller. So, by induction we know that M(e_1) $\dashrightarrow$ M(e_{n+1}) as in ODMG [11].

7.3 Rewritable Equations in NRA and NRC

The rewriting of rules from NRA to NRC and vice versa, we considered only those constructs of NRA and NRC that manipulate the basic values $\mathscr{B}$ and pairs $\mathscr{P}$ but not the bags. Since we only consider rewrites at the top level we can assume that NRC expressions contain one free variable x.

7.3.1 NRA to NRC Conversion

The NRA rule set R_2 which is considered for rewriting in NRC is shown below:

$$\mathbf{id} \circ f \equiv f \tag{7.1}$$

$$f \circ \mathbf{id} \equiv f \tag{7.2}$$

$$^{\lambda}c \circ f \equiv {}^{\lambda}c \tag{7.3}$$

$$(h \circ g) \circ f \equiv h \circ (g \circ f) \tag{7.4}$$

$$^{\lambda}\langle\rangle \circ f \equiv {}^{\lambda}\langle\rangle \tag{7.5}$$

$$\pi_1 \circ {}^{\lambda}\langle f, g\rangle \equiv f \tag{7.6}$$

$$\pi_2 \circ {}^{\lambda}\langle f, g\rangle \equiv g \tag{7.7}$$

$$^{\lambda}\langle g, h\rangle \circ f \equiv {}^{\lambda}\langle g \circ f, h \circ f\rangle \tag{7.8}$$

$$\pi_1 \circ {}^{\lambda}\langle\rangle \equiv {}^{\lambda}\perp \tag{7.9}$$

$$\pi_2 \circ {}^{\lambda}\langle\rangle \equiv {}^{\lambda}\perp \tag{7.10}$$

$$\pi_1 \circ {}^{\lambda}c \equiv {}^{\lambda}\perp \tag{7.11}$$

$$\pi_2 \circ {}^{\lambda}c \equiv {}^{\lambda}\perp \tag{7.12}$$

The conversion from NRA to NRC by application of the mapping mentioned in chapter 3 is shown below.

For equation 7.1 : $\mathbf{id} \circ f \equiv f$
$M(\mathbf{id} \circ f) \equiv M(f)$ on application of M we can write
$x := M(f)\ ;\ M(\mathbf{id}) \equiv M(f)$ Using mapping 3.3 on L.H.S.
$x := e_1\ ;\ x \equiv e_1$ Using mapping 3.1 on L.H.S
It is seen that the above NRC equation is present in the concise set of NRC as shown by equation 6.1. Thus, it is not required to check if it is provable using NRC concise set.

For equation 7.2 : $f \circ \mathbf{id} \equiv f$

$M(f \circ \mathbf{id}) \equiv M(f)$ on application of M we can write
$x := M(\mathbf{id})\ ; M(f) \equiv M(f)$ Using mapping 3.3 on L.H.S.
$x := x\ ; e_1 \equiv e_1$ Using mapping 3.1 on L.H.S
The above rewritten NRC equation is present in the concise set of NRC as shown by equation 6.44. Thus, it is not required to check if it is provable using NRC concise set.

For equation 7.3 : $^{\lambda}c \circ f \equiv {}^{\lambda}c$
$M({}^{\lambda}c \circ f) \equiv M({}^{\lambda}c)$ on application of M we can write
$x := M(f)\ ; M({}^{\lambda}c) \equiv M({}^{\lambda}c)$ Using mapping 3.3 on L.H.S. and mapping 3.4 on R.H.S.
$x := e_1\ ; c \equiv c$
The above rewritten NRC equation is present in the concise set of NRC as shown by equation 6.3. Thus, it is not required to check if it is provable using NRC concise set.

For equation 7.4 : $(h \circ g) \circ f \equiv h \circ (g \circ f)$
$M((h \circ g) \circ f) \equiv M(h \circ (g \circ f))$ on application of M we can write
$x := M(f);M(h \circ g) \equiv x := M(g \circ f);M(h)$ Using mapping 3.3 both on L.H.S. and R.H.S.
$x := M(f);(x := M(g);M(h)) \equiv x := (x := M(f);M(g));M(h)$
Using mapping 3.3 both on L.H.S. and R.H.S.
$x := e_1;(x := e_2;e_3) \equiv x := (x := e_1;e_2);e_3$
The above rewritten NRC equation is not present in the concise set and it is proved below.

R.H.S. = $x := (x := e_1;e_2);e_3$
$(x := e_1;x := e_2);e_3$
$x := e_1;(x := e_2;e_3)$
R.H.S. $\equiv$ L.H.S, Hence Proved.

For equation 7.5 : $^{\lambda}\langle\rangle \circ f \equiv {}^{\lambda}\langle\rangle$
$M({}^{\lambda}\langle\rangle \circ f) \equiv M({}^{\lambda}\langle\rangle)$ on application of M we can write
$x := M(f);M({}^{\lambda}\langle\rangle) \equiv \langle\rangle$ Using mapping 3.3 on L.H.S. and 3.7 on R.H.S.
$x := e_1;\langle\rangle \equiv \langle\rangle$ Using mapping 3.7 on L.H.S.
The above rewritten NRC equation is present in the concise set, equation 6.9.Thus, it is not required to check if it is provable using NRC concise set.

For equation 7.6 : $\pi_1 \circ {}^{\lambda}\langle f,g\rangle \equiv f$
$M(\pi_1 \circ {}^{\lambda}\langle f,g\rangle) \equiv M(f)$ on application of M we can write

$x := M(^{\lambda}\langle f,g\rangle);M(\pi_1) \equiv M(f)$ Using mapping 3.3 on L.H.S.
$x := \langle M(f),M(g)\rangle;x.1 \equiv M(f)$ Using mapping 3.5 and 3.6 on L.H.S.
$x := \langle e_1,e_2\rangle;x.1 \equiv e_1$
The above rewritten NRC equation is not present in the concise set, thus it is required to be proved. The proof is shown below.

R.H.S. = $x := \langle e_1,e_2\rangle;x.1$
$= (x := \langle e_1,e_2\rangle;x).1$ from 6.6
$= (\langle e_1,e_2\rangle).1$ from 6.1

$= e_1$ from 6.11
R.H.S $\equiv$ L.H.S. Hence Proved.

For equation 7.7 : $\pi_2 \circ {}^{\lambda}\langle f,g\rangle \equiv g$
$M(\pi_2 \circ {}^{\lambda}\langle f,g\rangle) \equiv M(f)$ on application of M we can write
$x := M(^{\lambda}\langle f,g\rangle);M(\pi_2) \equiv M(f)$ Using mapping 3.3 on L.H.S.
$x := \langle M(f),M(g)\rangle ;x.2 \equiv M(f)$ Using mapping 3.5 and 3.6 on L.H.S.
$x := \langle e_1,e_2\rangle ;x.2 \equiv e_2$
The above rewritten NRC equation is not present in the concise set, thus it is required to be proved. The proof is shown below.

R.H.S. = $x := \langle e_1,e_2\rangle;x.2$
$= (x := \langle e_1,e_2\rangle;x).2$ from 6.6
$= (\langle e_1,e_2\rangle).2$ from 6.1
$= e_2$ from 6.12
R.H.S $\equiv$ L.H.S. , Hence proved

For equation 7.8: ${}^{\lambda}\langle g,h\rangle \circ f \equiv {}^{\lambda}\langle g\circ f,h\circ f\rangle$
$M(^{\lambda}\langle g,h\rangle \circ f) \equiv M(^{\lambda}\langle g\circ f,h\circ f\rangle)$ on application of M we can write
$x := M(f);M(^{\lambda}\langle g,h\rangle) \equiv \langle M(g\circ f),h\circ f\rangle$ Using mapping 3.3 on L.H.S. and 3.5 on R.H.S.
$x := M(f);\langle M(g),M(h)\rangle) \equiv \langle x := M(f);M(g),x := M(f);M(h)\rangle$ Using mapping 3.5 on L.H.S. and 3.3 on R.H.S.
$x := e_1;\langle e_2,e_3\rangle) \equiv \langle x := e_1;e_2,x := e_1;e_3\rangle$
The above rewritten NRC equation is present in the concise set, equation 6.5.Thus, it is not required to check if it is provable using NRC concise set.

For equation 7.9: $\pi_1 \circ^{\lambda} \langle\rangle \equiv^{\lambda} \bot$

$M(\pi_1 \circ^{\lambda} \langle\rangle) \equiv M(^{\lambda}\bot)$ on application of M we can write

$x := M(^{\lambda}\langle\rangle);M(\pi_1) \equiv \bot$ Using mapping 3.3 on L.H.S. and 3.16 on R.H.S.

$x := \langle\rangle;x.1 \equiv \bot$ Using mapping 3.7 and 3.6 on L.H.S.

The above rewritten NRC equation is not present in the concise set, thus it is required to be proved. The proof is shown below.

L.H.S. = $x := \langle\rangle;x.1$

$= (x := \langle\rangle;x).1$ from 6.6

$= \langle\rangle.1$ from 6.1

$= \bot$ from 6.13

Hence Proved. Thus this equation is subsumed in NRC rewrite rules.

For equation 7.10: $\pi_2 \circ^{\lambda} \langle\rangle \equiv^{\lambda} \bot$

$M(\pi_2 \circ^{\lambda} \langle\rangle) \equiv M(^{\lambda}\bot)$ on application of M we can write

$x := M(^{\lambda}\langle\rangle);M(\pi_2) \equiv \bot$ Using mapping 3.3 on L.H.S. and 3.16 on R.H.S.

$x := \langle\rangle;x.2 \equiv \bot$ Using mapping 3.7 and 3.6 on L.H.S.

The above rewritten NRC equation is not present in the concise set, thus it is required to be proved. The proof is shown below.

L.H.S. = $x := \langle\rangle;x.2$

$= (x := \langle\rangle;x).2$ from 6.6

$= \langle\rangle.2$ from 6.1

$= \bot$ from 6.19

Hence Proved. Thus this equation is subsumed in NRC rewrite rules.

For equation 7.11 $\pi_1 \circ^{\lambda} c \equiv^{\lambda} \bot$

$M(\pi_1 \circ^{\lambda} c) \equiv M(^{\lambda}\bot)$ on application of M we can write

$x := M(^{\lambda}c)\,;M(\pi_1) \equiv \bot$ Using mapping 3.3 on L.H.S. and mapping 3.16 on R.H.S.

$x := c\,;x.1 \equiv \bot$ Using mapping 3.4 and 3.6 on L.H.S

This equation is not present in NRC concise set so we try if it can be subsumed in the NRC concise set using its existing rewrite rules. The proof is shown below.

L.H.S = $x := c\,;x.1$

$= (x := c\,;x).1$ from 6.6

$= (c.1)$ from 6.1
$= \perp$ from 6.17
$=$ R.H.S
Hence Proved. We see that 4.31 is subsumed in the NRC concise set.

For equation 7.12 $\pi_2 \circ {}^{\lambda} c \equiv {}^{\lambda} \perp$
$M(\pi_2 \circ {}^{\lambda} c) \equiv M({}^{\lambda} \perp)$ on application of M we can write
$x := M({}^{\lambda} c)\,;M(\pi_2) \equiv \perp$ Using mapping 3.3 on L.H.S. and mapping 3.16 on R.H.S.
$x := c\,;x.2 \equiv \perp$ Using mapping 3.4 and 3.6 on L.H.S
This equation is not present in NRC concise set so we try if it can be subsumed in the NRC concise set using its existing rewrite rules. The proof is shown below.
L.H.S = $x := c\,;x.2$
$= (x := c\,;x).2$ from 6.7
$= (c.2)$ from 6.1
$= \perp$ from 6.23
$=$ R.H.S
Hence Proved. We see that 4.32 is subsumed in the NRC concise set.

7.3.2 NRC to NRA Conversion

The NRC rule set R_1 which is considered for rewriting in NRA is shown below:

$$x := e_1\,;x \equiv e_1 \tag{7.13}$$

$$x := e_1\,;c \equiv c \tag{7.14}$$

$$x := e_1\,;\perp \equiv \perp \tag{7.15}$$

$$x := e_1\,;\langle e_2,e_3\rangle \equiv \langle x := e_1\,;e_2,x := e_1\,;e_3\rangle \tag{7.16}$$

$$x := e_1\,;(e_2).1 \equiv (x := e_1\,;e_2).1 \tag{7.17}$$

$$x := e_1\,;(e_2).2 \equiv (x := e_1\,;e_2).2 \tag{7.18}$$

$$x := e_1\,;\langle\rangle \equiv \langle\rangle \tag{7.19}$$

$$(\langle e_1, e_2 \rangle).1 \equiv e_1 \tag{7.20}$$

$$(\langle e_1, e_2 \rangle).2 \equiv e_2 \tag{7.21}$$

$$(\langle\rangle).1 \equiv \bot \tag{7.22}$$

$$(c).1 \equiv \bot \tag{7.23}$$

$$(\bot).1 \equiv \bot \tag{7.24}$$

$$(\langle\rangle).2 \equiv \bot \tag{7.25}$$

$$(c).2 \equiv \bot \tag{7.26}$$

$$(\bot).2 \equiv \bot \tag{7.27}$$

$$x := x \,;\, e_1 \equiv e_1 \tag{7.28}$$

$$x := \langle\rangle \,;\, c \equiv c \tag{7.29}$$

$$x := c \,;\, \langle\rangle \equiv \langle\rangle \tag{7.30}$$

For equation 7.13 : $x := e_1 \,;\, x \equiv e_1$
$M'(x := e_1 \,;\, x) \equiv M'(e_1)$ on application of M' we can write
$M'(x) \circ M'(e_1) \equiv M'(e_1)$ using 3.31 on L.H.S.
$\mathbf{id} \circ f \equiv f$ using 3.18 on L.H.S.
The above rewritten NRA rule is present in the concise set of NRA in equation 4.2.

For equation 7.14 : $x := e_1 \,;\, c \equiv c$
$M'(x := e_1 \,;\, c\,) \equiv M'(c)$ on application of M' we can write
$M'(c) \circ M'(e_1) \equiv {}^{\lambda}c$ using equation 3.31 on L.H.S. and 3.18 on R.H.S
${}^{\lambda}c \circ f \equiv {}^{\lambda}c$ using equation 3.18 on L.H.S.
The above rewritten NRA rule is present in the concise set of NRA in equation 4.1.

For equation 7.15 : $x := e_1 \,;\, \bot \equiv \bot$
$M'(x := e_1 \,;\, \bot) \equiv M'(\bot)$ on application of M' we can write
$M'(\bot) \circ M'(e_1) \equiv {}^{\lambda}\bot$ using equation 3.31 on L.H.S. and 3.32 on R.H.S

$^{\lambda}\perp \circ f \equiv {}^{\lambda}\perp$ using equation 3.32 on L.H.S.

The above rewritten NRA rule is not present in the concise set of NRA. Thus it is proved below.

R.H.S = $^{\lambda}\perp$

$= \mathbf{flat} \circ \mathbf{map}(^{\lambda}\langle\rangle) \circ {}^{\lambda}\{\{\cdot\}\}$ from 4.23

$= \mathbf{flat} \circ {}^{\lambda}\{\{\cdot\}\} \circ {}^{\lambda}\langle\rangle$ from 4.36

$= \mathbf{flat} \circ {}^{\lambda}\{\{\cdot\}\} \circ {}^{\lambda}\langle\rangle \circ f$ from 4.7

$= \mathbf{flat} \circ \mathbf{map}(^{\lambda}\langle\rangle) \circ {}^{\lambda}\{\{\cdot\}\} \circ f$ from 4.36

$= {}^{\lambda}\perp \circ f$ from 4.23

R.H.S. $\equiv$ L.H.S., Hence Proved.

The above NRA rule is proved using the ruleset present in the NRA concise set.

For equation 7.16 : $x := e_1 \,;\, \langle e_2, e_3\rangle \equiv \langle x := e_1 \,;\, e_2, x := e_1 \,;\, e_3\rangle$

$M'(x := E \,;\, \langle F, G\rangle) \equiv M'(\langle x := e_1 \,;\, e_2, x := e_1 \,;\, e_3\rangle)$ on application of M' we can write

$M'(\langle e_2, e_3\rangle) \circ M'(e_1) \equiv {}^{\lambda}\langle M'(x := e_1 \,;\, e_2), M'(x := e_1 \,;\, e_3)\rangle$ using equation 3.31 on L.H.S. and 3.19 on R.H.S.

$^{\lambda}\langle M'(e_2), M'(e_3)\rangle \circ M'(e_1) \equiv {}^{\lambda}\langle M'(e_2) \circ M'(e_1), M'(e_3) \circ M'(e_1)\rangle$ using equation 3.19 on L.H.S. and 3.31 on R.H.S.

$^{\lambda}\langle f, g\rangle \circ h \equiv {}^{\lambda}\langle f \circ h, g \circ h\rangle$

This rewritten NRA rule is present in the NRA concise set equation 4.10.

For equation 7.17 : $x := e_1 \,;\, (e_2).1 \equiv (x := e_1 \,;\, e_2).1$

$M'(x := E \,;\, (F).1) \equiv M'((x := E \,;\, F).1)$ on application of M' we can write

$M'((e_2).1 \circ e_1) \equiv \pi_1 \circ M'(x := e_1 \,;\, e_2)$ using equation 3.31 on L.H.S. and 3.20 on R.H.S.

$(\pi_1 \circ M'(e_2)) \circ M'(e_1) \equiv \pi_1 \circ (M'(e_2) \circ M'(e_1))$ using equation 3.20 on L.H.S. and 3.31 on R.H.S

$(\pi_1 \circ f) \circ g \equiv \pi_1 \circ (f \circ g)$

The above rewritten NRA rule is not present in the concise set of NRA. Thus it will be proved using the concise set of NRA.

R.H.S. = $\pi_1 \circ (f \circ g)$

$= (\pi_1 \circ f) \circ g$ from 4.4

R.H.S. $\equiv$ L.H.S. , Hence Proved.

For equation 7.18 : $x := e_1 \,;\, (e_2).2 \equiv (x := e_1 \,;\, e_2).2$

$M'(x := e_1 \,;\, (e_2).2) \equiv M'((x := e_1 \,;\, e_2).2)$ on application of M' we can write

$M'((e_2).2 \circ e_1) \equiv \pi_2 \circ M'(x := e_1 \,;\, e_2)$ using equation 3.31 on L.H.S. and 3.20 on R.H.S.

$(\pi_2 \circ M'(e_2)) \circ M'(e_1) \equiv \pi_2 \circ (M'(e_2) \circ M'(e_1))$ using equation 3.20 on L.H.S. and 3.31 on R.H.S
$(\pi_2 \circ f) \circ g \equiv \pi_2 \circ (f \circ g)$
The above rewritten NRA rule is not present in the concise set of NRA. Thus it will be proved using the concise set of NRA.
R.H.S. = $\pi_2 \circ (f \circ g)$
$= (\pi_2 \circ f) \circ g$ from 4.4
R.H.S. $\equiv$ L.H.S. , Hence Proved.

For equation 7.19 : $x := e_1 \; ; \langle\rangle \equiv \langle\rangle$
$M'(x := e_1 \; ; \langle\rangle) \equiv M'(\langle\rangle)$ on application of M' we can write
$M'(\langle\rangle) \circ M'(e_1) \equiv {}^{\lambda}\langle\rangle$ using equation 3.31 on L.H.S. and 3.21 on R.H.S.
${}^{\lambda}\langle\rangle \circ M'(e_1) \equiv {}^{\lambda}\langle\rangle$ using equation 3.21 on L.H.S.
${}^{\lambda}\langle\rangle \circ f \equiv {}^{\lambda}\langle\rangle$
The above rewritten NRA rule is present in the NRA concise set equation 4.7.

For equation 7.20 : $(\langle e_1, e_2\rangle).1 \equiv e_1$
$M'((\langle e_1, e_2\rangle).1) \equiv M'(e_1)$ on application of M' we can write
$\pi_1 \circ M'({}^{\lambda}\langle e_1, e_2\rangle) \equiv M'(e_1)$ using equation 3.20 on L.H.S.
$\pi_1 \circ {}^{\lambda}\langle M'(e_1), M'(e_2)\rangle \equiv M'(e_1)$ using equation 3.19 on L.H.S.
$\pi_1 \circ {}^{\lambda}\langle f, g\rangle \equiv f$
The above rewritten NRA rule is present in the NRA concise set equation 4.8.

For equation 7.21 : $(\langle e_1, e_2\rangle).2 \equiv e_2$
$M'((\langle e_1, e_2\rangle).2) \equiv M'(e_2)$ on application of M' we can write
$\pi_2 \circ M'({}^{\lambda}\langle e_1, e_2\rangle) \equiv M'(e_2)$ using equation 3.20 on L.H.S.
$\pi_2 \circ {}^{\lambda}\langle M'(e_1), M'(e_2)\rangle \equiv M'(e_2)$ using equation 3.19 on L.H.S.
$\pi_2 \circ {}^{\lambda}\langle f, g\rangle \equiv g$
The above rewritten NRA rule is present in the NRA concise set equation 4.9.

For equation 7.22 : $(\langle\rangle).1 \equiv \perp$
$M'((\langle\rangle).1) \equiv M'(\perp)$ on application of M' we can write
$\pi_1 \circ M'(\langle\rangle) \equiv {}^{\lambda}\perp$ using equation 3.20 on L.H.S. and 3.32 on R.H.S.
$\pi_1 \circ {}^{\lambda}\langle\rangle \equiv {}^{\lambda}\perp$ using equation 3.21 on L.H.S.
This above rewritten NRA rule is present in the NRA concise set equation 4.29.

For equation 7.23 : $(c).1 \equiv \perp$
$M'((c).1) \equiv M'(\perp)$ on application of M' we can write
$\pi_1 \circ M'(c) \equiv {}^{\lambda}\perp$ using equation 3.20 on L.H.S. and 3.32 on R.H.S.
$\pi_1 \circ {}^{\lambda}c \equiv {}^{\lambda}\perp$ using equation 3.18 on L.H.S.
This above rewritten NRA rule is present in the NRA concise set equation 4.31.

For equation 7.24 : $(\perp).1 \equiv \perp$
$M'((\perp).1) \equiv M'(\perp)$ on application of M' we can write
$\pi_1 \circ M'(\perp) \equiv {}^{\lambda}\perp$ using equation 3.20 on L.H.S. and 3.32 on R.H.S.
$\pi_1 \circ {}^{\lambda}\perp \equiv {}^{\lambda}\perp$ using equation 3.32 on L.H.S.
This above rewritten NRA rule is not present in the NRA concise set. Thus it will be proved using the NRA concise set.
L.H.S. = $\pi_1 \circ {}^{\lambda}\perp$
$= \pi_1 \circ map(f) \circ {}^{\lambda}c$ from 4.19
$= {}^{\lambda}\perp \circ {}^{\lambda}c$ from 4.33
$= {}^{\lambda}\perp$ from ${}^{\lambda}\perp \circ f \equiv {}^{\lambda}\perp$
L.H.S. $\equiv$ R.H.S. , Hence Proved.

For equation 7.25 : $(\langle\rangle).2 \equiv \perp$
$M'((\langle\rangle).2) \equiv M'(\perp)$ on application of M' we can write
$\pi_2 \circ M'(\langle\rangle) \equiv {}^{\lambda}\perp$ using equation 3.20 on L.H.S. and 3.32 on R.H.S.
$\pi_2 \circ {}^{\lambda}\langle\rangle \equiv {}^{\lambda}\perp$ using equation 3.21 on L.H.S.
This above rewritten NRA rule is present in the NRA concise set equation 4.30.

For equation 7.26 : $(c).2 \equiv \perp$
$M'((c).2) \equiv M'(\perp)$ on application of M' we can write
$\pi_2 \circ M'(c) \equiv {}^{\lambda}\perp$ using equation 3.20 on L.H.S. and 3.32 on R.H.S.
$\pi_2 \circ {}^{\lambda}c \equiv {}^{\lambda}\perp$ using equation 3.18 on L.H.S.
This above rewritten NRA rule is present in the NRA concise set equation 4.32.

For equation 7.27 : $(\perp).2 \equiv \perp$
$M'((\perp).2) \equiv M'(\perp)$ on application of M' we can write
$\pi_2 \circ M'(\perp) \equiv {}^{\lambda}\perp$ using equation 3.20 on L.H.S. and 3.32 on R.H.S.
$\pi_2 \circ {}^{\lambda}\perp \equiv {}^{\lambda}\perp$ using equation 3.32 on L.H.S.
This above rewritten NRA rule is not present in the NRA concise set. Thus it will be proved

using the NRA concise set.

L.H.S. $= \pi_2 \circ {}^{\lambda}\perp$

$= \pi_2 \circ map(f) \circ {}^{\lambda}c$ from 4.19

$= {}^{\lambda}\perp \circ {}^{\lambda}c$ from 4.34

$= {}^{\lambda}\perp$ from ${}^{\lambda}\perp \circ f \equiv {}^{\lambda}\perp$

L.H.S. $\equiv$ R.H.S. , Hence Proved.

For equation 7.28 : $x := x\,;e_1 \equiv e_1$

$M'(x := x\,;e_1) \equiv M'(e_1)$ on application of M' we can write

$M'(x) \circ M'(e_1) \equiv M'(e_1)$ using equation 3.31 on L.H.S.

$\mathbf{id} \circ f \equiv f$ using equation 3.17 on L.H.S.

The above obtained NRA rule is part of NRA concise set shown by equation 4.2.

For equation 7.29 : $x := \langle\rangle\,;c \equiv c$

$M'(x := \langle\rangle\,;c) \equiv M'(c)$ on application of M' we can write

$M'(c) \circ M'(\langle\rangle) \equiv {}^{\lambda}c$ using equation 3.31 on L.H.S. and 3.18 on R.H.S.

${}^{\lambda}c \circ {}^{\lambda}\langle\rangle \equiv {}^{\lambda}c$ using equation 3.18 and 3.21 on L.H.S.

The above NRA rule is not present in the NRA concise set. Thus, it is proved below.

L.H.S. $= {}^{\lambda}c \circ {}^{\lambda}\langle\rangle$

$= {}^{\lambda}c$ from 4.1

L.H.S. $\equiv$ R.H.S. , Hence Proved.

For equation 7.30 : $x := c\,;\langle\rangle \equiv \langle\rangle$

$M'(x := c\,;\langle\rangle) \equiv M'(\langle\rangle)$ on application of M' we can write

$M'(\langle\rangle) \circ M'(c) \equiv {}^{\lambda}\langle\rangle$ using equation 3.31 on L.H.S. and 3.21 on R.H.S.

${}^{\lambda}\langle\rangle \circ {}^{\lambda}c \equiv {}^{\lambda}\langle\rangle$ using equation 3.18 and 3.21 on L.H.S.

The above NRA rule is not present in the NRA concise set. Thus, it is proved below. L.H.S. $= {}^{\lambda}\langle\rangle \circ {}^{\lambda}c$

$= {}^{\lambda}\langle\rangle$ from 4.7

L.H.S. $\equiv$ R.H.S. , Hence Proved.

Chapter 8

Conclusions and Future Work

8.1 Summary

Though RDF Gears is an inspiration to this work, we aim to develop rewriting based optimization strategies that can be applied to similar query languages like PigLatin and Hive. We use the existing techniques and study them to be able to extrapolate some of the existing techniques for our rewriting. We come up with exhaustive lists of rewrite rules in NRA and NRC that we subsequently churn in Prover 9 and mathematically eliminate the redundancies to arrive at a minimum core set. We prove that this minimum core set is enough to represent the full exhaustive set of rewrites. For NRC, we also present auxiliary rules needed to assist Prover 9 to understand the semantics of NRC operators.

In the course of this work, in addition to defining NRA and NRC syntax and semantics, we study the various rewriting based optimizations that currently exist and identify many of the existing ideas that can be extrapolated and applied to RDF Gears. Some basic rules can be applied directly and some ideas can be extrapolated and applied. We predominantly rely on Fegaras's and Wong's work and find semantically valid, equivalent strategies for our version of NRA. We also write a lot of additional rewriting schemes in both NRA and NRC, and establish a way in which rules can be easily mapped from one to another, making it very simple to add more rewrite rules to this set later.

The main result of our work is not only these two NRA and NRC rule sets but also a proof of equivalence relationship between these two rule sets. We present the proofs for NRA to NRC and NRC to NRA for the fragments of language that deal with only basic values, pairs and tuples. Moreover, we only consider rewrites at the top level of an expression, for one

free variable in the language.

8.2 Conclusions

We conclude this thesis by presenting answers to the research questions raised at the start of this work:

- ***Is there a minimum core set in NRA such that it subsumes the exhaustive set of all rewrite rules covered in this work?***
  In Chapter 4, we describe the process of arriving at a minimum core set, that we call the concise or the non-derivable set of NRA. We come up with a full set of NRA first, which is nothing but an exhaustive list of rewrite rules, which covers all possible rewrites needed for our language. We use Prover 9, the theorem prover, to reduce this set to a minimum core set which semantically incorporates the full set. But we do not rely on just Prover 9, we also prove this derivability mathematically in Chapter 5. So, indeed there is a minimum core set for NRA in which all possible rewrites are covered, and we also prove this.

- ***Is there a minimum core set in NRC such that it subsumes the exhaustive set of all rewrite rules covered in this work?***
  In Chapter 6, we describe the process of arriving at a minimum core set of NRC. We come up with a full set or exhaustive list of rewrite rules of NRC first. Even here, we use Prover 9, to reduce this set to a minimum core set which semantically incorporates the full set. But a lot of auxiliary rewrite rules were additionally written to describe the semantics of the operators to Prover 9, in order for Prover 9 to be able to arrive at any meaningful result. For this we do not explicitly present the proof of the derivability, but we have verified each and every one of the redundant rewrite rules. So, indeed there is a minimum core set for NRC in which all possible rewrites are covered, and we also prove this.

- ***Is there an equivalence relationship between the rule sets of NRA and NRC?***
  There is an apparent equivalence relationship between the rule sets of NRA and NRC at the top level. We present a mathematical proof for this in Chapter 7. The proof obtained for NRA and NRC that is comprised of basic values, pairs and tuples, for

one free variable. The proof is obtained using mathematical induction, and the equivalence is proved exhaustively, covering all rewrite rules form NRA to NRC and from NRC to NRA.

8.3 Future Work

The equivalence relationship that we have proved for NRA and NRC is for one free variable and covers basic values, pairs and tuples. However, a proof for equivalence and mapping between NRA and NRC allowing rewriting at all levels of expression is desirable.

Also, the mathematical proof that we have presented can be extended to hold for all data types including collections, which would be a very interesting research work in itself, considering the complexity added due to bags and comprehensions.

Another important work that can be taken up is the application of these optimization strategies that are theoretical to a graph based model that RDF Gears uses. Since RGL is very similar to NRA it will be interesting to map these textual rewrites to graphical ones.

Bibliography

[1] Eric Feliksik, *"RDF Gears: A Data Integration Framework for the Semantic Web"*, 2011.

[2] Limsoon Wong, *"Querying Nested Collections: PHD Thesis*, University of Pennsylvania, 1994.

[3] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, Andrew Tomkins, *"Pig Latin: A Not-So-Foreign Language for Data Processing"*, SIGMOD, June, 2008.

[4] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy, *"Hive - A Warehousing Solution Over a Map-Reduce Framework"*, VDLB, August, 2009.

[5] Leonidas Fegaras, David Maier, *"Optimizing Object Queries using and Effective Calculus"*, ACM Transactions of Database Systems 25(4): 457-516, December, 2000.

[6] W. McCune, *" 'Prover9' and 'Mace4' "*, http://www.cs.unm.edu/∼mccune/Prover9, 2005-2010.

[7] Matthais Jarke, Jurgen Koch *"Query Optimization in Database Systems"*, ACM Comput. Surv. 16(2): 111-152, 1984.

[8] Chris J. Date, *"Introduction to Database System"*, Pearson Education Inc., Eighth Edition, January, 2004.

[9] Yannis E. Ioannidis, *"Query Optimization"*, ACM Comput. Surv. 28(1): 121-123, 1996.

[10] Roth, Mark A. and Korth, Herry F. and Silberschatz, Abraham, *"Extended Algebra and Calculus for Nested Relational Databases"*, ACM Transactions of Database Systems 13(4): 389-417, December, 1988.

[11] R. G. G. Cattell, *"The Object Database Standard: ODMG-93 "*, Morgan Kaufmann Publishers Inc., Release 1.2, 1996.

[12] Ramez Elmasri , Shamkant Navathe, *"Fundamentals of Database Systems"*, Addison-Wesley, Sixth Edition, July, 2013.

[13] Abraham Silberschatz, S. Sudarshan, Henry F. Korth, *"Database System Concepts"*, McGraw-Hill, Sixth Edition, January, 2010.

[14] Leonidas Fegaras, David Maier *"Towards an Effective Calculus for Object Query Languages"*, SIGMOD, 1995.

[15] Limsoon Wong, *"Normal Forms and Conservative Properties for Query Languages over Collection Types"*, ACM-PODS, 1993.

[16] Val Breazu-Tannen, Peter Buneman, Limsoon Wong, *"Naturally Embedded Query Languages"*, ICDT, 1992.

[17] Jan Van den Bussche, Stijn Vansummeren, *"Polymorphic Type Inference for the Named Nested Relational Caluculus"*, ACM Transactions of Computational Logic 9(1)-3 December, 2007 .

[18] Jan Van den Bussche, Dirk van Gucht, Stijn Vansummeren, *"Well-Definedness and Semantic Type Checking for the Nested Relational Calculus"*, Database Theory 371(3): 183-199, March, 2007 .

[19] Jan Van den Bussche, Dirk van Gucht, Stijn Vansummeren, *"A Crash Course on Database Queries"*, PODS, June, 2007.

[20] Hector Garcia-Molina, Jeff Ullman, Jennifer Widom, *"Database Systems: The Complete Book"*, Prentice Hall, Second Edition, June, 2008.

[21] Jan Hidders *"Notes on NRA/NRC"*, May, 2013.

Appendix A

Prover 9 Notation for NRA

Following is the mapping of NRA with its Prover 9 Notation:

NRA	Prover 9	NRA	Prover 9
$^{\lambda}c$	c(X)	$\circ$	*
f	F	$^{\lambda}\{\{\}\}$	emptybag
g	G	$^{\lambda}\langle\rangle$	emptytuple
h	H	$^{\lambda}\langle f,g\rangle$	pair(F,G)
π_1	p1	π_2	p2
id	id	**flat**	flat
map(f)	map(F)	$^{\lambda}\uplus$	union
$^{\lambda}\times$	cross	$^{\lambda}\{\{\cdot\}\}$	sngl
$^{\lambda}\perp$	undef		

Following is the concise set of NRA in Prover 9 Notation:

```
% Saved by Prover9-Mace4 Version 0.5B, March 2008 (Dec 2007 LADR).

set(ignore_option_dependencies). % GUI handles dependencies

% Language Options
```

```
op(501, infix_left, "*").

if(Prover9). % Options for Prover9
  set(prolog_style_variables).
  assign(max_seconds, 60).
end_if.

if(Mace4).   % Options for Mace4
  set(prolog_style_variables).
  assign(max_seconds, 60).
end_if.

formulas(assumptions).

c(X)*F=c(X).
id*F=F.
F*id=F.
(H*G)*F=H*(G*F).

map(F)*emptybag=emptybag.
emptybag*F=emptybag.
emptytuple*F=emptytuple.

p1*pair(F,G)=F.
p2*pair(F,G)=G.
pair(G,H)*F=pair(G*F,H*F).

map(G)*map(F)=map(G*F).
union*pair(F,G)=union*pair(G,F).
union*pair(emptybag,sngl*F)=sngl*F.

flat*flat=flat*map(flat).
map(G)*flat=flat*map(map(G)).
flat*map(sngl)=map(id).
flat*pair(F,G)=undef.
```

```
union*pair(pair(F,G),H)=undef.
map(F)*c(X)=undef.

cross*pair(F,cross*pair(G,H))=map(pair(p1*p1,pair(p2*p1,p2)))
                *cross*pair(cross*pair(F,G),H).

flat*(union*pair(F,G))=union*pair(flat*F,flat*G).

union*pair(F,union*pair(G,H))=union*pair(union*pair(F,G),H).

flat*map(pair(F,G))*sngl=undef.
flat*map(emptytuple)*sngl=undef.

map(H)*(union*pair(F,G))=union*pair(map(H)*F,map(H)*G).

cross*pair(pair(F,G),H)=undef.

flat*emptytuple=undef.

union*pair(c(X),F)=undef.

p1*emptytuple=undef.
p2*emptytuple=undef.
p1*c(X)=undef.

p1*map(F)=undef.
p2*map(F)=undef.
p2*c(X)=undef.
map(F)*sngl=sngl*F.
map(F)*emptytuple = undef.

flat*c(X)=undef.
flat*map(c(X))*sngl=undef.
```

```
cross*pair(sngl*F,sngl*G)=sngl*pair(F,G).
cross*pair(F,union*pair(G,H))=union*
         pair(cross*pair(F,G),cross*pair(F,H)).
cross*pair(map(G)*F,H)=map(pair(G*p1,p2))*cross*pair(F,H).
cross*pair(F,G)=map(pair(p2,p1))*(cross*pair(G,F)).
cross*pair(flat*F,H)=flat*map(cross*pair(p1,sngl*p2))
              *cross*pair(F,H).

flat*map(cross*pair(sngl,G*p1))*cross*pair(sngl,F)=
                 cross*pair(cross*pair(sngl,F),G).

map(id)*F=F->map(p1)*cross*pair(F,F)=F.

end_of_list.

formulas(goals).

end_of_list.
```

Appendix B

Prover 9 Notation for NRC

Following is the mapping of NRC with its Prover 9 Notation:

NRC	Prover 9	NRC	Prover 9
c	c(Z)	e	E
e_1	F	$\{\{\}\}$	emptybag
e_2	G	$\langle\rangle$	emptytuple
e_3	H	$\langle e_1, e_2\rangle$	pair(F,G)
$x.1$	p1(X)	$x.2$	p2(X)
$\{\{e_1 \mid e_2\}\}$	(F || G)	x	v(X)
;	;	$\uplus$	+
$:=$	is	$\{\{\cdot\}\}$	sngl
$\perp$	undef	ε	emptyTl

Following is the concise set of NRC in Prover 9 Notation:

```
% Saved by Prover9-Mace4 Version 0.5B, March 2008 (Dec 2007 LADR).

set(ignore_option_dependencies). % GUI handles dependencies

% Language Options
```

```
op(501,infix_left,"is").
op(502,infix_left,"in").
op(503,infix_left,";").
op(504,infix_left,"||").
op(505,infix_left,"+").

if(Prover9). % Options for Prover9
  assign(max_seconds, 28800).
  set(prolog_style_variables).
end_if.

if(Mace4).   % Options for Mace4
  set(prolog_style_variables).
  assign(max_seconds, 60).
end_if.

formulas(assumptions).

v(X) is E;v(X) = E.
v(X)!=v(Y)  ->  v(X) is E;v(Y) = v(Y).

v(X) is E ; c(Z) = c(Z).
v(X) is E ; undef = undef.
v(X) is E ; pair(F,G) = pair( v(X) is E;F , v(X) is E;G ).
v(X) is E ; p1(F) = p1( v(X) is E;F ).
v(X) is E ; p2(F) = p2( v(X) is E;F ).
v(X) is E ; emptybag = emptybag.
v(X) is E ; emptytuple = emptytuple.
v(X) is E ; F+G = ( v(X) is E;F) + (v(X) is E;G).

p1(pair(F,G))=F.
p2(pair(F,G))=G.

p1(emptytuple)=undef.
p1(emptybag)=undef.
```

```
p1(F || G)=undef.
p1(F+G)=undef.
p1(c(Z))=undef.
p1(undef)=undef.

p2(emptytuple)=undef.
p2(emptybag)=undef.
p2(F || G)=undef.
p2(F+G)=undef.
p2(c(Z))=undef.
p2(undef)=undef.

F+G=G+F.
F+(G+H)=(F+G)+H.
pair(F,G) + E = undef.
emptytuple + E = undef.
c(Z) + E = undef.
F+undef=undef.
(F =  (v(X) || v(X) in F) ) -> F+F = F.

(F || v(X) in pair(G,H) ; E) = undef.
(F || v(X) in c(Z) ; E) = undef.
(F || v(X) in emptytuple ; E) = undef.
(F || v(X) in undef ; E) = undef.

-det(v(Y),E) & v(X)!=v(Y)  ->  v(X) is E ; v(Y) is F =
         v(Y) is (v(X) is E;F);(v(X)  is  E).

-det(v(X),E)  -> v(X)  is  E;v(X)  is  F = v(X) is (v(X) is E;F).

-det(v(Y),E) & v(X)!=v(Y)  -> v(X) is E ; v(Y) in F =
          v(Y) in (v(X)  is  E ; F ); (v(X)  is  E ).

-det(v(X),E)  -> v(X) is E;v(X) in F = v(X) in (v(X) is E;F).
```

```
v(X) is E;(F || G) = ( F || ( v(X) is E;G ) ).

v(X) is E;emptyTl = v(X) is E.
emptyTl; v(X) is E = v(X) is E.
emptyTl;emptyTl = emptyTl.

v(X)!=v(Y) -> v(X) is E;v(Y) is F = v(Y) is (v(X) is E;F);v(X) is E.

v(X) is E ;  v(X) is F  = v(X) is (v(X) is E;F).

(S;T);U=S;(T;U).

(H || T;v(X) is E)=(v(X) is E;H || T).

-det(v(X), F) -> v(Y) is E ; F  =  v(X) is E ; v(Y) is v(X) ; F.
-det(v(X), F) -> v(Y) in E ; F  =  v(X) in E ; v(Y)  is  v(X) ; F.

det (v(X) , p1( F) ) -> det( v(X) , F).

det (v(X) , p2( F) ) -> det( v(X) , F).
det (v(X) , undef) -> $F.
det (v(X) , emptybag) -> $F.
det (v(X) , emptytuple) -> $F.
det (v(X) , c(Z)) -> $F.
det (v(X), v(X)).
v(X)!=v(Y) -> ( det (v(X), v(Y)) -> $F ).

det(v(X), (F +G) ) -> det(v(X),F) | det(v(X),G).

v(Z)!=v(X)  ->   (det( v(Z) , (v(X) is E ; F) )  ->
         (det( v(Z) , E ) | det( v(Z) , F ) )).

det( v(X) , (v(X) is E ; F) ) -> det( v(X) , E ).

det( v(X) , (F || emptyTl) ) -> det( v(X) , F ).
```

```
v(X)!=v(Y) -> ((det( v(X) , (F || (v(Y)  is  G; H)) ))  ->
        (det( v(X) , G ) | det( v(X) , (F || H) )) ).

det( v(X) , (F || (v(X)  is  G; H)) ) -> det( v(X) , G ).

v(X)!=v(Y) ->  ( det( v(X) , (F || (v(Y) in G; H)) ) ->
         det( v(X) , G ) | det( v(X) , (F || H) ) ).
det( v(X) , (F || (v(X) in G; H)) ) -> det( v(X) , G ).

v0  != v(X).
v(X) = v(Y)  <->  X = Y.

v(X)  is  v(X); E = E.

v(X) is emptytuple ; c(Z) = c(Z).

v(X) is c(Z) ; emptytuple = emptytuple.

emptyTl; v(X) in E = v(X) in E.

v(X) in E;emptyTl = v(X) in E.

end_of_list.

formulas(goals).

( E || v(X) in ( F ); G ) = ( E || v(X) is F; G ).

end_of_list.
```

Appendix C

Sample Prover 9 Proof for NRA

Proof of NRA rule :

$$\mathbf{flat} \circ^{\lambda} \{\{\}\} \equiv^{\lambda} \{\{\}\}$$

using the NRA notation for prover 9 given in Appendix A.

```
============================== prooftrans ============================
Prover9 (32) version Dec-2007, Dec 2007.
Process 2800 was started by gprasad on CRDDT076.
Thu Feb 20 13:15:02 2014
The command was "/cygdrive/c/Program Files (x86)/Prover9-Mace4/
bin-win32/prover9".
============================== end of head ===========================
op(501,infix_left,"*").

set(prolog_style_variables).
============================== end of input ==========================

============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.01 (+ 0.01) seconds.
% Length of proof is 8.
% Level of proof is 3.
% Maximum clause weight is 11.
% Given clauses 15.
```

```
2 flat * emptybag = emptybag # label(non_clause) # label(goal).  [goal].
6 A * B * C = A * (B * C).  [assumption].
7 map(A) * emptybag = emptybag.  [assumption].
22 flat * map(sngl) = map(id).  [assumption].
23 map(id) = flat * map(sngl).  [copy(22),flip(a)].
66 flat * emptybag != emptybag.  [deny(2)].
110 flat * emptybag = emptybag.  [para(23(a,1),7(a,1,1)),
rewrite([6(6),7(5)])].
111 $F.  [resolve(110,a,66,a)].

============================== end of proof ==========================
```

Appendix D

Sample Prover 9 Proof for NRC

Proof of NRC rule :

$$x := \langle\rangle \,;\, \pi_1(x) \equiv \bot$$

using the NRC notation for prover 9 given in Appendix B.

```
============================== prooftrans ============================
Prover9 (32) version Dec-2007, Dec 2007.
Process 9804 was started by gprasad on CRDDT076,
Thu Feb 20 13:17:45 2014
The command was "/cygdrive/c/Program Files (x86)/Prover9-Mace4/
bin-win32/prover9".
============================== end of head ===========================
op(501,infix_left,"is").
op(502,infix_left,"in").
op(503,infix_left,";").
op(504,infix_left,"||").
op(505,infix_left,"+").

set(prolog_style_variables).
============================== end of input ==========================

============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.00 (+ 0.01) seconds.
```

```
% Length of proof is 10.
% Level of proof is 3.
% Maximum clause weight is 15.
% Given clauses 5.

26 v(A) is emptytuple ; p1(v(A)) = undef # label(non_clause)
# label(goal).  [goal].
27 v(A) is B ; v(A) = B.  [assumption].
34 v(A) is B ; p1(C) = p1(v(A) is B ; C).  [assumption].
35 p1(v(A) is B ; C) = v(A) is B ; p1(C).  [copy(34),flip(a)].
44 p1(emptytuple) = undef.  [assumption].
45 undef = p1(emptytuple).  [copy(44),flip(a)].
136 v(c1) is emptytuple ; p1(v(c1)) != undef.  [deny(26)].
137 v(c1) is emptytuple ; p1(v(c1)) != p1(emptytuple).
[copy(136),rewrite([45(9)])].
149 p1(A) = v(B) is A ; p1(v(B)).  [para(27(a,1),35(a,1,1))].
150 $F.  [resolve(149,a,137,a(flip))].

============================== end of proof ==========================
```