1998

Optimisation of query processing in heterogeneous distributed multidatabase systems

Seyed Mohsen Sedighi
University of Wollongong

Recommended Citation
NOTE

This online version of the thesis may have different page formatting and pagination from the paper copy held in the University of Wollongong Library.

UNIVERSITY OF WOLLONGONG

COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.
Optimisation of Query Processing in Heterogeneous Distributed Multidatabase Systems

A thesis submitted in fulfillment of the requirements for the award of the degree

Doctor of Philosophy

from

UNIVERSITY OF WOLLONGONG

by

Seyed Mohsen Sedighi

School of Information Technology & Computer Science
June 1998
© Copyright 1998

by

Seyed Mohsen Sedighi

All Rights Reserved
Dedicated to

my wife
Declaration

This is to certify that the work reported in this thesis was done by the author, unless specified otherwise, and that no part of it has been submitted in a thesis to any other university or similar institution.

Seyed Mohsen Sedighi
June 10, 1998
Abstract

One of important characteristics of heterogeneous distributed multidatabase systems is autonomy of underlying component database systems. Heterogeneity is also the result of autonomy. Owing to autonomy, optimisation of query processing in the systems demands the detection of a number of optimisation factors and the application of different techniques from the traditional ones.

This thesis considers optimisation of query processing in heterogeneous distributed multidatabase systems. In the systems, a query is decomposed into a collection of subqueries. After translation to the query language of component databases, subqueries are submitted to the component databases. Partial results, obtained from the processing of subqueries, are converted to a canonical data model format and then they are integrated.

Efficiency of query processing in the systems depends on the detection of the impacts of partial results on each other. The impacts are applied to the efficient submission of subqueries to the component databases and to the efficient integration of the partial results.

Initially, we introduce and adopt a generalised database model for query processing to be independent of any particular database model. In the database, queries are represented by algebraic expressions. Based on the model, we develop a technique to label expression trees corresponding to the database expressions for discovering optimisation of query processing.

We address labelling as a general technique in detecting optimisation of query processing that is not only applicable to heterogeneous distributed multidatabase systems but also to centralised and to homogeneous distributed database systems.

We address query optimisation in heterogeneous distributed multidatabase systems at two stages. One stage is engaged with the submission order of subqueries to the component databases by applying the impacts of partial results on each other, query processing costs in the component databases, data transmission costs, data conversion
and data integration costs. At this stage, we present an algorithm to identify the submission order of subqueries for minimisation of the response time.

Another stage is concerned with the efficient integration of partial results. We take advantage of delays in obtaining partial results, while some of them are available at the submission site, and the others are not arrived yet. We address gradual integration of available partial results in the delays. Furthermore, by using impacts of available partial results on each other, their size can be reduced or they can be prepared to speed up future computations.

We propose algorithms for efficient integration of partial results. We consider two different environments for optimisation of query postprocessing. In a static environment, response times of subqueries are predictable. Based on that assumption, we present an algorithm to take the most advantage of the delays to reduce the size of available partial results for minimisation of the response time.

In a dynamic environment, response times of subqueries are not reliable. We develop another algorithm in the dynamic environment to reduce the size of available partial results as much as possible in order to minimise the response time.

As a particular case of the generalised database model, we employ relational algebra operations and expressions.
Acknowledgements

First of all, my sincere thanks go to my supervisor, Dr J. R. Getta. Without his invaluable assistance and support, this thesis would not have been possible. Not only has he been a very good advisor, but also a very good friend.

I sincerely thank my wife and my children. I hope in the future it will be possible to compensate a little for those things that I should have done for them but could not.

Special thanks to my brother Mostafa who has always persuaded and supported me to follow my studies.

Last and not least, I wish to thank the Ministry of Culture and Higher Education of the Islamic Republic of Iran that provided me with the scholarship which enabled me to continue my studies toward a PhD program.
# Contents

Abstract v

Acknowledgements vii

1 Introduction 1

1.1 An Overview of Distributed Query Processing 5

1.1.1 Homogeneous Distributed Query Processing 5

1.1.2 Heterogeneous Distributed Query Processing 8

1.2 Sources and Classes of Optimisation 12

1.3 A Heterogeneous Distributed Query Processing Model 13

1.4 Thesis Outline 13

2 Labelling Technique 14

2.1 A Generalised Database Model 14

2.1.1 Views of Data 14

2.1.2 Operation Classes 15

2.2 Expression Trees 18

2.2.1 Classes of Composite Operations 19

2.3 Labelling Rules 20

2.4 Example: Relational Algebra 25

2.5 Summary 27

3 Optimisation Based on Reductions 30

3.1 Reductions 30

3.1.1 Beneficial Reductions 31

3.1.2 Detection of Reductions 33

3.2 Application of Reductions 35

3.2.1 Classical Optimisation 35
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Corresponding terms of the generalised database model to the traditional and modern database models and the systems like html and file.</td>
<td>15</td>
</tr>
<tr>
<td>2.2</td>
<td>The class of an operation for an argument such as $r$ is identified by the operation and the label of the input edge to the operation. Output edge of the operation is labelled by the class of that operation for argument $r$.</td>
<td>25</td>
</tr>
<tr>
<td>3.1</td>
<td>Reduction expressions for join operation $(k = x \cap y)$.</td>
<td>41</td>
</tr>
<tr>
<td>3.2</td>
<td>Reduction expressions for difference operation $(k = x \cap y)$.</td>
<td>42</td>
</tr>
<tr>
<td>3.3</td>
<td>Reduction expressions for union operation $(k = x \cap y)$.</td>
<td>44</td>
</tr>
<tr>
<td>3.4</td>
<td>Reduction expressions for intersection operation $(k = x \cap y)$.</td>
<td>45</td>
</tr>
<tr>
<td>3.5</td>
<td>Reduction operations for join $(k = x \cap y)$.</td>
<td>48</td>
</tr>
<tr>
<td>3.6</td>
<td>Reduction operations for intersection $(k = x \cap y)$.</td>
<td>48</td>
</tr>
<tr>
<td>3.7</td>
<td>Reduction operations for union $(k = x \cap y)$.</td>
<td>48</td>
</tr>
<tr>
<td>3.8</td>
<td>Reduction operations for difference operation when a reduced argument is on the right hand side of difference $(k = x \cap y)$.</td>
<td>48</td>
</tr>
<tr>
<td>3.9</td>
<td>Reduction operations for difference operation when a reduced argument is on the left hand side of difference $(k = x \cap y)$.</td>
<td>49</td>
</tr>
<tr>
<td>4.1</td>
<td>Three tuples of relation $r$.</td>
<td>55</td>
</tr>
<tr>
<td>4.2</td>
<td>Values for parameters $E_i$ and $D_i$.</td>
<td>67</td>
</tr>
<tr>
<td>4.3</td>
<td>Values for $w_j^i$ and $\rho_j^i$.</td>
<td>67</td>
</tr>
<tr>
<td>5.1</td>
<td>Arrival times of arguments of expression $e$.</td>
<td>79</td>
</tr>
<tr>
<td>6.1</td>
<td>Values for different parameters and available arguments at the arrival times.</td>
<td>86</td>
</tr>
<tr>
<td>6.2</td>
<td>Values for parameters $E_i$ and $D_i$.</td>
<td>98</td>
</tr>
<tr>
<td>6.3</td>
<td>Values for $o_j^i$ and $\rho_j^i$.</td>
<td>98</td>
</tr>
</tbody>
</table>
Values for different parameters and available arguments at the arrival times.
List of Figures

1.1 Distributed query processing ........................................... 2
1.2 A simplified scenario of heterogeneous query processing .......... 4

2.1 An expression tree of $\Psi_1(\Psi_2(\Psi_4(r_1, r_2), \Psi_5(r_3)), \Psi_3(\Psi_6(r_4), \Psi_7(r_5, r_6), \Psi_8(r_7)))$ in which paths for leaf nodes $r_1$ and $r_4$ are depicted in thicker lines. ........................................... 19
2.2 An expression tree of $\Psi_1(\Psi_2(\Psi_4(r_1, r_2), \Psi_5(r_3)), \Psi_3(\Psi_6(r_4), \Psi_7(r_5, r_6), \Psi_8(r_7, r_8)))$ in which $\alpha$ and $\beta$ are composite operations. .......... 20
2.3 Difference is a non-commutative operation. In (a) the operation node is labelled by “l−” because the investigated input is on the left hand side of the operation. In (b) the operation node is labelled by “r−” because the considered input is on the right hand side of the operation. ....... 26
2.4 An expression tree of relational expression $((r_1 \cup r_2) - (\sigma_\phi (r_3))) \cup ((r_4 - r_5) \bowtie r_6)$. ................................................................. 27
2.5 An expression tree in which edges in paths for $r_1$ and $r_5$ are labelled. ................................................................. 28
3.1 An expression tree of $\Psi_1(\Psi_2(\Psi_4(r_1, r_2), \Psi_5(r_3)), \Psi_3(\Psi_6(r_4), \Psi_7(r_5, r_6), \Psi_8(r_7)))$. A common operation for any two or more leaf nodes is a root of the smallest subtree that contains those nodes. .................. 32
3.2 An expression tree of $e$, where $\Psi$ is the common operation for two arguments $r_i$ and $r_j$. ............................................... 32
3.3 An expression tree of $e$, where one of the leaf nodes is replaced by reduction expression $\Psi_{ji}(r_i, r_j)$. ............................................... 33
3.4 An expression tree of $(r_1 \cap r_2) \bowtie (r_3 - r_4)$ ........................................ 34
3.5 An expression tree in which $L_r$ and $L_s$ are the classes of the operations whose outputs are the inputs to common operation $\Psi$ for two arguments $r$ and $s$. ........................................... 34
3.6 An expression tree of a relational expression. ........................................... 36
3.7 An expression tree of a relational expression in which $r_j$ is replaced by reduction expression $\Psi_{ji}(r_i, r_j)$. ................................................. 36
3.8 Sequential subquery processing. ................................................. 39
3.9 Parallel subquery processing. ................................................... 39
3.10 A labelled expression tree of relational expression $(r_1 \cap r_2) \mathcal{N} ((r_3 - r_4) \cup r_5)$ for paths of $r_1$ and $r_2$. ................................................. 42
3.11 A labelled expression tree of relational expression $((r_1 \cap r_2) - r_3) - ((r_4 \mathcal{N} r_5) \cap r_6)$ for paths of $r_1$ and $r_2$. ................................................. 43
3.12 A labelled expression tree of relational expression $((r_1 \cap r_2) - r_3) - ((r_4 \mathcal{N} r_5) \cap r_6)$ for the paths of $r_3$ and $r_6$. ................................................. 43
3.13 A labelled expression tree of relational expression $r_1 \cup (r_2 - (r_3 - r_4))$ for paths of $r_1$ and $r_4$. ................................................... 45
3.14 A labelled expression tree of relational expression $(r_1 \mathcal{N} r_2) \cap ((r_3 \mathcal{N} r_4) - r_5)$ for paths of $r_1$ and $r_3$. ................................................... 46
4.1 The amount of work in each next step is reduced. ......................... 60
4.2 Sequence of execution for $r_1, \ldots, r_4$. .................................. 69
5.1 A skeleton of an expression tree with the common subtrees. ............ 77
8.1 An expression tree in which the common operation for two arguments $r$ and $s$ is join and the labels of the input edges to the operation are $A_x(r)$ and $N_y(s)$. ................................................. 110
8.2 An expression tree in which the common operation for two arguments $r$ and $s$ is join and the labels of the input edges to the operation are $N_x(r) +$ and $E_y(s)$. ................................................. 114
8.3 An expression tree in which the common operation for two arguments $r$ and $s$ is difference and the labels of the input edges to the operation are $A_x(r)$ and $N_y(s)$. ................................................. 118
8.4 An expression tree in which the common operation for two arguments $r$ and $s$ is intersection and the labels of the input edges to the operation are $A_x(r)$ and $N_y(s)$. ................................................. 137
8.5 An expression tree in which the common operation for two arguments $r$ and $s$ is intersection and the labels of the input edges to the operation are $N_x(r)$ and $E_y(s)$. ................................................. 142
Query optimisation is very important in efficient execution of queries in database systems where complex queries are posed and interactive responses are demanded by human beings. Another dimension of the efficiency can be the substantial savings in the cost of queries.

Query processing consists of translation of a declarative query into low-level data manipulation operations and execution of the operations. Query optimisation is the process by which an optimal execution strategy among a set of alternatives for a given query is searched for and found. If queries are expressed as sequences of low-level data manipulation operations, any optimisation is performed manually by the user [Date90]. In such a system, the kinds of operations and the order of execution of those operations are usually decided by the user.

Typically, query optimisation has three aspects [GaHK92]:

1. An execution space.
2. A cost model.
3. A search strategy.

The execution space defines the set of all alternative plans for the processing of a query. The plans are equivalent in a sense that their computations have the same results but they differ in the execution order of operations and the way the operations are implemented. The cost model estimates the cost of an execution plan. The search space is used by a search algorithm to obtain the minimum cost plan.

In general, the search space for a given query is very large. Thus, finding an optimal execution strategy is computationally intractable. Furthermore, many query optimisation algorithms are based on heuristics to prune the exhaustive search which do not necessarily lead to an optimal solution [YooL89].
In centralised database systems, for declarative queries such as SQL ones, query processing is performed in two steps. They are query translation and query optimisation. For example, in relational database systems, query translation takes a SQL query and translates it into an expression of relational algebra. Since there are more than one algebraic translation of the query, some of them are better than the others from the point of view of the performance. The best translation of the query is determined by using algebraic transformation rules and a cost function which estimates the execution cost of the query according to the algebraic specification.

In distributed database systems, there are two more steps between query translation and query optimisation, called data localisation and global query optimisation [OzsV91]. Data localisation, as the name indicates, localises data of query with regard to data distribution information. In this step, the fragments (i.e. distributed data) which are involved in the query are decided. The query is then transformed into a query that operates on fragments. Thus, if items of global data are fragmented, they are replaced with the fragments. Distributed data can be restructured by using the inverse of the fragmentation rules (i.e. union for horizontal fragments and join for vertical fragments). In the global query optimisation, an optimal execution plan with regards to order of operations, data movement between sites, and the choice of distributed and local algorithms is generated(Figure 1.1).

An execution strategy for a distributed query is based on a cost function which is central to the global query optimisation. The cost function is often defined in terms
of time units which refer to the resources such as disk space, disk I/O, buffer space, CPU cost and communication cost. Generally, the cost function is a combination of I/O, CPU and communication costs. To make the cost function simpler, only the communication cost in many systems is considered as the most significant factor. This is valid when local processing cost is negligible in comparison with the communication cost.

In heterogeneous distributed multidatabase systems (HDMDBSs), there are two more important features. They are autonomy and heterogeneity of component databases participating in the systems that largely affect the query processing.

Owing to autonomy and heterogeneity, data transmittability between component database systems may not be supported. Before data transmission, the data may need to be transformed from one structure to another structure. Due to autonomy, component database systems may accept only queries. Furthermore, because of autonomy, no query processing algorithms can be imposed on the component database systems. The statistical data of component databases, required to optimise queries, may not be revealed because of the autonomy. Although there are similarities between homogeneous distributed database systems and HDMDBSs, in comparison, query processing in HDMDBSs is much more complicated.

A simplified scenario of query processing in HDMDBSs is as follows. A query in the systems is initially decomposed into a collection of subqueries and is also translated to a data integration expression. Each subquery needs to be sent to one component database. After translation to the query language of the component databases, subqueries are submitted to the component databases for independent processing. Partial results of subquery processing are transmitted to the submission site and then they are converted to a canonical data model format. After conversion, the partial results are integrated by the order specified in the data integration expression (Figure 1.2).

An approach to query processing in HDMDBSs is to submit all subqueries to the component databases without considering what impacts partial results of the subqueries have on each other. This approach is efficient as long as any partial result, obtained from query processing in the relating component database, cannot effectively reduce the size of partial results of query processing in other component databases. Furthermore, in this approach, subqueries are processed in parallel and the partial results are transmitted to the submission site in parallel.

A deficiency of the approach, if the impacts of partial results on each other are not taken into account, may be transmission of huge amount of unnecessary data from
component databases to the submission site. As a result, in addition to transmission time, conversion and integration times of partial results are increased.

Therefore, efficiency of query processing in HDMDBSs is consolidated by the recognition of the impacts of partial results on each other. In order to reduce the size of query processing results in the component databases, each subquery uses partial results obtained from processing of some of other subqueries. This requires detecting the impacts that subquery results have on each other before determining in what order subqueries should be submitted to the component databases.

In the exploration of the impacts of partial results on each other, it is required to find partial results of what subqueries should be obtained first and then what subqueries should use the results for processing in the component databases to minimise the response time. The search space is gigantic, because it includes not only the sequential execution of subqueries in the component databases but the parallel execution as well.

Moreover, a component database may accept only queries. Partial results of other subqueries may be needed in order to reduce the size of query processing result in the component database. Since only queries are allowed to be sent to the component database, a method is needed to reduce the size of query processing result in the
component by using the partial results obtained from processing of other subqueries.

Query postprocessing in HDMDBSs involves integration of partial results obtained from the execution of subqueries in the component databases. After the partial results are transmitted to the submission site, the semantic heterogeneities are dealt with and converted to a canonical data model, they are integrated. In fact, in the conversion, the semantic heterogeneities are resolved.

The impacts of partial results on each other are also important to the efficiency of query postprocessing. Furthermore, in HDMDBSs, partial results of query processing in the component databases may not be available at the same time. Some of the partial results may arrive in the submission site earlier than the others. It is important to benefit from the delays when some of the partial results arrive later to perform operations on available partial results for minimising the response time. In the delays, available partial results may gradually be integrated or they may be prepared by using the impacts of partial results on each other to speed up future computations.

In the query postprocessing, partial results of query processing in the component databases may arrive in the submission site at unpredicted times. In such an environment, optimisation of query postprocessing should be dynamic if the response time is to be effectively minimised.

In the next section, we first review query processing in homogeneous distributed database systems and in HDMDBSs called homogeneous and heterogeneous distributed query processing, respectively. Then, the contribution of the thesis is expounded.

1.1 An Overview of Distributed Query Processing

1.1.1 Homogeneous Distributed Query Processing

Queries in homogeneous distributed database systems are processed as follows [YuCh84]:

1. Initial local processing which involves all local processing such as selections and projections at each database site.

2. Reduction operations which include using a sequence of semijoins to reduce the size of relations.

3. Final processing (or integration), when all relations are sent to the final site for integration.
In the late seventies several prototype systems such as system R* [LMHD85], SDD-1 [BGWR81], and distributed INGRES [StoN77] were developed. System R* and Distributed INGRES take both local processing and transmission costs into account, but in SDD-1, the important factor in query optimisation is the minimisation of the communication costs [CerP85].

In homogeneous distributed query processing, it is assumed that the database sites are fully connected to each other. However, a small number of papers have considered query processing and optimisation in star and chain networks. For star networks, an optimal algorithm for general queries has been proposed [KeTY82]. Also, an optimal algorithm that minimises the total data transmission costs for simple queries in star network topologies has been presented [SMKY83]. Query processing for simple queries in chain networks, where data communication costs largely dominate the local processing costs, has been investigated [GurS84]. Simple queries are the queries that, after initial local processing, the relations contain one attribute which is the joining attribute.

Semijoin operations are well-known techniques in query processing of homogeneous distributed database systems that intend to mainly reduce the cost of data transmitted between sites. It was used for the first time by Wong [Wong77] in a greedy algorithm without using the term for query processing in SDD-1 [BGWR81]. The theoretical aspects of semijoins were studied in [BerC81].

Many algorithms have been proposed to determine a semijoin program for an optimal query processing [ApHY83, BlaL82, YuCh82, YuOL84, CheL84, PraV88, CBTY89]. Furthermore, the optimal algorithm proposed in [HevY79] for simple queries has been generalised in [ApHY83] based on a heuristic and an exhaustive search.

Sometimes cardinality of complement of a joining attribute may be smaller than the cardinality of the attribute. In this case, to improve semijoins, it is suggested to send the complement during semijoins execution [YuCh82].

Most semijoin algorithms execute semijoins sequentially in a sense that the reduction effects of one semijoin are used to reduce the cost of other semijoins. To execute the semijoins in parallel, one-shot semijoin execution strategy has been proposed [WaLC91]. The strategy has been used to optimise the response time of distributed query processing [WaCS92].

A query is called a tree query if either its query graph is a tree structure or the query is equivalent to a query that its query graph is still a tree structure, otherwise the query is called cyclic query [BerG81]. Tree queries can be fully reduced by using semijoins
1.1. An Overview of Distributed Query Processing

[Ber81]. An optimal semijoin program for tree queries where the relations can have more than one common attributes has been derived [PraV88]. For special cases of tree queries such as chain and star queries, algorithms to derive optimal sequences of semijoins have been proposed [ChBH84, CheL85].

The cost of a semijoin is the amount of necessary data transmission for its computation and its benefit is the amount of data that is eliminated. Beneficial semijoins are those whose benefits are greater than the costs. For example, in SDD-1 [BGWR81], the application of as many beneficial semijoins as possible is tried. There are cases where joins outperform semijoins. Accordingly, both joins and semijoins as reducers in distributed query processing have been studied [CheY90, CheY93].

It has been shown that the problem of an optimal semijoin program is NP-hard [Hev89]. Thus, efficient semijoin algorithms are necessarily heuristic in nature.

The semantic of set operations can be used to process queries that manipulate sets [ElsK90]. In this work, an attempt is made to reduce the size of data transmission as much as possible by transforming a distributed set query into a distributed non-set query, using functional dependencies.

Semijoin based algorithms are intended mainly to reduce the communication costs. Techniques to reduce local processing costs in distributed query processing are based on horizontal and vertical partitioning schemes [CerP85] in order to have parallel processing at different sites for improving the response time.

In distributed databases, for efficiency and reliability, the data may be duplicated at different database sites. However, optimal processing of queries that reference data with multiple copies is a NP-hard problem [YLCC82].

Distributed-INGRES uses the fragment and replicate strategy [EpSW79]. In the strategy, one of the relations referenced by a query is fragmented, and other relations are replicated at the sites to allow parallel processing. There are a number of algorithms that use fragmentation for processing queries in parallel to minimise the response time [YGCT84, YCTB85, Sege86, YGZT87, CBTY89, YGBC89]. However, for a query to be processed in parallel at different sites, it is required that substantial data transmission and preparation be performed [LiuY93].

Homogeneous distributed query processing mainly concentrates on select-project-join (SPJ) queries with conjunctive predicates. The SPJ queries are a subset of the query languages of today's databases. While this is an important class of queries in which there are many good algorithms for them, there are other important queries such as unions, aggregations and queries with disjunctions that need optimisations as well.
Although there are partial solutions for these classes of queries in centralised query optimisation, the solutions cannot always be extended to distributed query processing [OzsV94] because the data are distributed.

### 1.1.2 Heterogeneous Distributed Query Processing

Autonomy [Corn88, GarK94] is an important feature of HDMDBSs. There are different kinds of autonomy in HDMDBSs, called design, communication, execution and association autonomy [SheL90]. The preservation of full autonomy of component database systems is not only costly but it has great impacts on the query processing. Heterogeneity is another characteristic of HDMDBSs that is mainly the outcome of design autonomy among component database systems.

A little research for optimization of query processing in HDMDBSs has been reported. Most of the research on the problem has concentrated on the semantics of translation and integration. Little research has focused on the problem in general, taking into accounts all features of HDMDBSs. Furthermore, many homogeneous distributed query processing algorithms have been directly applied to HDMDBSs [ApHY83, BerC81, BGWR81]. In general, the algorithms may not be directly applicable to HDMDBSs [LuSh92, LuOG93, MenY95].

There are several basic problems in heterogeneous distributed query processing that make the query processing very complicated. Some of the problems are:

- autonomy and heterogeneity of component databases,
- the lack of effective and realistic cost models needed for query optimisation [Brei90],
- different capabilities of component databases in query processing and
- the transmittability of data between the autonomous component databases.

Thus, in order to simplify query processing problem, usually some assumptions are made. For example, it may be assumed that heterogeneity of the components are resolved and attention is paid to the cost models of the component databases [DuKS92, ZhuL94a, ZhuL94b, HaSN96]. In general, if both heterogeneity and autonomy of the components are ignored, then heterogeneous query processing problems may be shifted to homogeneous ones.

A heterogeneous distributed query optimisation problem was first addressed in Multibase [LanR82, SBDG89]. The query optimisation is performed at local and global
1.1. An Overview of Distributed Query Processing

levels, using centralised and distributed query optimisation techniques, extended with additional techniques for generalised objects and partially overlapped data. After a global query, expressed in DAPLEX [Ship81], is decomposed, the decomposed sub-queries may be reduced by removing the operations that are not supported by the component database systems. However, the main objective is integration rather than query optimisation [RePR89].

Limitations of semijoin reduction for queries that can include outerjoins and aggregations are pointed in [Daya85]. An outerjoin-aggregation sequence is proposed to define generalised entity types. Also, semiouterjoins are used to reduce the size of partial results in the component databases.

Query decomposition algorithms in the cases of non-replicated and partially replicated data are proposed in [RusC87]. In their approach, semijoins cannot be used between two component databases. Furthermore, semijoins are only possible between a global site and a component database site. Since communication with a component database is through its query facility, no modifications to the component are required. A new component database can be added to the heterogeneous database system by defining only the export schema and establishing communication line to it. In the proposed work, autonomy of the component database systems is fully preserved and the heterogeneity is not considered.

To integrate local schemas for providing global views of heterogeneous database systems, outerjoin is used [Chen90]. Since processing a one-sided outerjoin (left or right) is cheaper than an outerjoin, and processing a join is cheaper than one-sided outerjoin, conditions for identifying when an outerjoin can be avoided and one-sided outerjoin can be used or when a join can be used instead of one-sided outerjoin are considered [Chen90]. A solution for producing unnecessary null values by an outerjoin is suggested [SchC92]. In the solution, in order to remove the null values, output of an outerjoin operation is produced as a set of sets.

An entity join operator joins two relations on their compatible or incompatible keys [ChaS91, TsaC90]. An extension of semijoin to reduce data transmission cost for entity join query processing in a heterogeneous database environment where data transmission cost mainly dominates query processing cost at component databases is proposed [TsaC94].

The need for a domain translation table (DTT) that contains a value identifier (surrogate) and semantically equivalent attribute values is proposed for global query optimisation [RamP95]. If the DTT is to be stored in component databases, it obviously
1.1. An Overview of Distributed Query Processing

violates the autonomy.

The semantic query optimisation approach [King81] is adopted for query processing in heterogeneous database systems [Card94, PaCY92]. Furthermore, in decomposing a global query to subqueries, the semantics which are concerned with the integrity constraints of the heterogeneous database system are used to simplify execution of subqueries at the component databases. The heterogeneity problem of the component databases is ignored but the autonomy is fully preserved.

Optimising queries by taking advantage of schema conflicts has been studied in [LeCL95]. In their approach, different schema conflicts are analysed and classified into different types. Then, based on the differences, the costs of executing the same relational operation on the differences are evaluated and a weight is assigned to each of them. Also, the component database sites that return the same result are identified. The search for the least execution cost is based on computation of total weights and using a systematic optimisation method. In general, the optimisation issues in this approach are at the semantic level.

For integration operation, three operations called 2-way outerjoin, key derivation and generalised attribute derivation operations have been defined [LimS93]. The operations are used along with relational algebra operations. Cost-based optimisation is the basis of the work. An attempt has been made to adopt the query optimisation techniques of R [LMHD85] to the query optimisation. To obtain the statistical information about component databases, needed for optimisation, query processing are performed in four phases.

There have been a few approaches at the physical level to optimise queries in heterogeneous database systems [DuKS92, ZhuL94a, ZhuL94b, LuOG93, EvrD95, DuSD95, HaSN96]. As mentioned earlier, one of the main problems in optimising heterogeneous database queries is the lack of cost models for component database systems because the components are autonomous. The first attempt to overcome the problem was the technique introduced in [DuKS92]. In their work, the cost model parameters of a component database required for query optimisation are estimated through database calibration. Another approach is a classification for grouping the component database queries and assigning a cost estimation formula for the queries in each class [ZhuL94a]. Another technique based on building a fuzzy cost model about optimisation parameters is proposed by the same authors [ZhuL94b].

The proposed techniques for deriving query cost functions for component database systems in [DuKS92] and [ZhuL94a] have some limitations which are enumerated in
1.1. An Overview of Distributed Query Processing

[HaSN96]. In that work [HaSN96], a new method based on classical clustering algorithm to classify queries for approximating the query cost functions is described.

An optimisation strategy that reduces heterogeneous database query response time by using tree transformations is studied in [DuSD95]. In fact, a few algorithms are proposed which use basic transformations to balance a left deep join tree to a bushy tree. In their study, the application of two phase optimisation [HonS91] in heterogeneous database query optimisation is emphasised because in order to extend an existing optimiser, no modifications to it are needed. In two phase optimisation, first a query is optimised with respect to the total cost, then the result is improved with respect to the response time. In their approach to heterogeneous database query optimisation, heterogeneity of component databases is not considered.

Query optimisation in the MIND project [DDKO95] is investigated in [EvrD95]. To maximise parallel execution of intersite joins, two phase optimisation technique is used. In the first phase, a linear order for the most profitable joins is produced. Then, in the second phase, an attempt is made to maximise parallel execution of the intersite joins by using the appearance times of partial results, data transmission costs and conversion costs of subqueries. In the case of data replication, an algorithm is proposed for load balancing among the component databases.

Optimisation of query postprocessing in HDMDBSs for the first time was considered in [SedG96a]. In our approach, the delays in obtaining partial results of subquery processing in the component database systems are identified. The delays are also addressed in [AFTU96] by considering only delays which are the result of the communication network problems. In our approach, a few optimisation techniques based on transformations of a data integration expression are proposed to minimise the response time. Also, to discover impacts of subquery results on each other, a technique based on labelling an expression tree is proposed. By taking advantage of the delays, available partial results can be gradually integrated. Moreover, in the delays, the impacts can be used to reduce the size of available partial results or to prepare them for speeding up future computations.

The impacts of subquery results on each other determine the potential for optimisation of query processing. Based on that fact, we have proposed an algorithm to identify in what order subqueries should be submitted to the component databases to minimise the response time [SedG96b].
The Problem

Except for a few approaches to optimisation of query processing in HDMDBSs such as those proposed in [RusC87, SedG96a], many other approaches assume transmissibility of data between component database systems. In general, the assumption in HDMDBSs may not be true [MenY95].

In this thesis, the labelling technique proposed in [SedG96a] is considered in a more general sense. We introduce a generalised database model where queries are represented by algebraic expressions. Based on the model, the database expressions are labelled to detect optimisation of query processing. We show that a technique for detecting the impacts of partial results on each other is not only applicable to HDMDBSs but to centralised and to homogeneous distributed database systems.

We also assume that transmission of data to the component databases is not supported. Thus, consideration is given to how the impacts of partial results on each other can be used without data transmission to reduce the size of query processing results in component databases. By using the impacts of partial results on each other, an algorithm is proposed to discover in what order subqueries must be submitted to the component databases to minimise the response time.

In the thesis, optimisation of query postprocessing is considered in detail. Algorithms are proposed for minimisation of the response time in two different environments, static and dynamic. In the static environment, it is assumed that response times of query processing in the components are predictable, while in the dynamic environment, the response times are unknown.

We employ relational algebra expressions and operations that are examined as a particular case of the generalised database model to detect the impacts of relations on each other.

1.2 Sources and Classes of Optimisation

There are two sources of optimisation [Ullm89], called algebraic manipulation and cost-based estimation strategies. In algebraic manipulation, queries are optimised apart from the actual data and the physical structure of the data. In cost-based estimation strategies, based on the data and the structure, the best strategy is found.

Generally, optimisation algorithms are classified into heuristic optimisation and systematic optimisation [WhaK90]. Heuristic optimisation uses heuristics for transforming a query to an equivalent one to be processed efficiently (e.g. [WonY76]). In systematic
optimisation, an optimal solution can be found by computing and comparing all different strategies (e.g. [SACL89]). In systematic optimisation, techniques such as random search, single start, multistart, or simulated annealing are employed [GroM95].

1.3 A Heterogeneous Distributed Query Processing Model

A heterogeneous distributed query processing model is used throughout the thesis that the participating component database systems are autonomous and heterogeneous. The component database systems are accessed through their external user interfaces. They accept only queries. In other words, transmission of data to the components is not supported.

The cost of query processing in the components is estimated from one of the techniques proposed for HDMDBSs [DuKS92, ZhuL94a, HaSN96]. Syntax and semantic heterogeneities of the components are resolved. This is the main assumption of the system. Another term for a global query, of course after translation, is data integration expression.

1.4 Thesis Outline

In the next chapter, a generalised database model is introduced and based on the model, a labelling technique is proposed to detect optimisation of query processing. In chapter 3, optimisation of query processing based on reducing the size of partial results of query processing in component databases by applying a labelling technique is presented and various applications of the technique are considered. Chapter 4 is devoted to an optimal query processing in HDMDBSs. In chapter 5, optimisation of query postprocessing is considered. Chapter 6 concentrates on optimisation of query postprocessing in static and dynamic environments. In chapter 7, a summary of the contribution is presented. Chapter 8 includes an appendix.
The objectives of this chapter are to provide the theoretical backgrounds for optimisation of query processing in heterogeneous distributed multidatabase systems based on syntactical transformations of queries.

We adopt a generalised database model for query processing where queries are represented by algebraic expressions. Expression trees are used for two dimensional representation of corresponding expressions. We propose a technique to label expression trees for detecting optimisation of query processing.

As a particular case of this study, a labelling theory is examined to label expression trees corresponding to relational algebra expressions.

### 2.1 A Generalised Database Model

A database model is a general framework for describing how data and their manipulations are represented. The definition of a database model, as commonly accepted [Codd79, Date90, HamM81, HulY84, TsiL82], consists of three distinct components:

- structures or views of data for representing objects and their relationships,
- operations that provide mechanisms for restructuring data in a variety of ways and
- the capability of expressing integrity constraints.

In the following discussion, structures and operations of a generalised database model are introduced. In the database, queries are represented by algebraic expressions.

#### 2.1.1 Views of Data

To be independent of any particular database model for introducing the typical concepts, we have adopted a generalised database model. In the database, data are grouped
### 2.1. A Generalised Database Model

<table>
<thead>
<tr>
<th>database system terminology</th>
<th>relational</th>
<th>object-oriented</th>
<th>hierarchical</th>
<th>network</th>
<th>html</th>
<th>file</th>
</tr>
</thead>
<tbody>
<tr>
<td>data container</td>
<td>relation</td>
<td>class</td>
<td>record type</td>
<td>record type</td>
<td>template</td>
<td>file</td>
</tr>
<tr>
<td>data item</td>
<td>tuple</td>
<td>object</td>
<td>occurrence</td>
<td>occurrence</td>
<td>page</td>
<td>record</td>
</tr>
<tr>
<td>property</td>
<td>attribute</td>
<td>variable</td>
<td>field</td>
<td>field</td>
<td>link</td>
<td>+ text</td>
</tr>
</tbody>
</table>

| record type | record | occurrence | field | link | + text | field |

Table 2.1: Corresponding terms of the generalised database model to the traditional and modern database models and the systems like html and file.

into collections of *data containers*. A data container consists of a set of homogeneous *data items*. Each data item is described by a set of properties.

Corresponding terms of the generalised database model to the traditional and modern database models such as relational, object-oriented, hierarchical, network and the systems like html and file are given in Table 2.1.

In the database, data items of two data containers may have one or more similar properties. By similarity, for example, of two properties, we mean that they intend and express one thing. Similar properties of two or more data containers will be called *common properties* of those data containers.

#### 2.1.2 Operation Classes

In the generalised database model, inputs to operations are data containers. Each operation produces a new data container as the output. An operation may be either *unary*, *binary*, *ternary*, ..., or *n-ary* operation.

As stated earlier, a data container consists of a set of data items. The number of data items of a data container will be called *cardinality* of the data container.

Based on an individual input and what is produced in the output with respect to the input, an operation in the database is classified. For a non-unary operation with respect to one of the inputs, the operation falls into an operation class, while for another input, the operation may fall into either the same class or another class.

Consider operation $\alpha(r_1, \ldots, r_i, \ldots, r_m)$. For an input data container such as $r_i$, we look at the appearance of the data items and the properties of $r_i$ in the output of $\alpha$. Let the output of $\alpha$ be data container $r$. Consider each one of the following assumptions:

- For each data item $k$ in $r_i$ there exist data item $j$ in $r$ such that the values of the
respective common properties of \( k \) and \( j \) are exactly the same. If the common properties of \( r \) and \( r_i \) do not consist of all properties of the data items of \( r_i \), then the number of properties of \( r_i \) in the output of \( \alpha \) is reduced.

- For each data item \( j \) in \( r \) there exists data item \( k \) in \( r_i \) such that the values of the respective common properties of \( k \) and \( j \) are exactly the same. In other words, there exists at least a data item in \( r_i \) such that the values of the corresponding common properties are not in \( r \). Thus, by focusing on the common properties in data container \( r \), the cardinality of \( r_i \) for the common properties in the output of \( \alpha \) is reduced.

- For each data item \( k \) in \( r_i \) there exists data item \( j \) in \( r \) such that the values of the respective common properties of \( k \) and \( j \) are exactly the same. Thus, by focusing on the common properties in data container \( r \), all data items of \( r_i \) for the common properties arrive in the output of \( \alpha \). Moreover, there may exist the values of the respective common properties in \( r \) that are not in \( r_i \). In this case, the cardinality of \( r_i \) for the common properties in the output of \( \alpha \) is increased.

- For each data item \( k \) in \( r_i \) there does not exist data item \( j \) in \( r \) such that the values of the respective common properties of \( k \) and \( j \) are the same. In this case, no data items of \( r_i \) for the common properties arrive in the output of \( \alpha \).

Therefore, in the output of operation \( \alpha \), the number of the properties of the data items of \( r_i \) may be decreased. The operation may either increase or decrease the cardinality of \( r_i \) for the common properties in the output. Furthermore, the data items of \( r_i \) for the common properties may never arrive in the output of \( \alpha \).

In general, any operation in the database falls into one of the following operation classes:

1. Property decreasing operation \((\pi_x)\).
2. Cardinality decreasing operation \((E_x)\).
3. Weak cardinality decreasing operation \((E_x+)_i\).
4. Cardinality increasing operation \((A_x+)\).
5. Negation operation \((N_x+)\).
Definition 2.1 Operation $\alpha(r_1, \ldots, r_i, \ldots, r_m)$, whose output is data container $r$ and $x$ are the common properties of $r$ and $r_i$, is a property decreasing operation for input data container $r_i$ iff:

- $x$ do not consist of all properties of the data items of $r_i$ and
- for each data item $k$ in $r_i$ there exist data item $j$ in $r$ such that the values of the respective common properties of $k$ and $j$ are exactly the same.

Definition 2.2 Operation $\alpha(r_1, \ldots, r_i, \ldots, r_m)$, whose output is data container $r$ and $x$ are the common properties of $r$ and $r_i$, is a cardinality decreasing operation for input data container $r_i$ iff for each data item $j$ in $r$ there exists data item $k$ in $r_i$ such that the values of the respective common properties of $k$ and $j$ are exactly the same.

If $x$ are all properties of the data items of $r_i$, then instead of $E_x$, the notation $E$ will represent the cardinality decreasing operation.

Definition 2.3 Operation $\alpha(r_1, \ldots, r_i, \ldots, r_m)$, whose output is data container $r$ and $x$ are the common properties of $r$ and $r_i$, is a weak cardinality decreasing operation for input data container $r_i$ iff:

- it is a cardinality decreasing operation and
- for each data item $k$ in $r_i$ there exists at least data item $j$ in $r$ such that the values of the respective common properties of $k$ and $j$ are not the same.

If $x$ are all properties of the data items of $r_i$, then instead of $E_{x^+}$, the notation $E^+$ will represent the weak cardinality decreasing operation.

Definition 2.4 Operation $\alpha(r_1, \ldots, r_i, \ldots, r_m)$, whose output is data container $r$ and $x$ are the common properties of $r$ and $r_i$, is a cardinality increasing operation for input data container $r_i$ iff:

- for each data item $k$ in $r_i$ there exists data item $j$ in $r$ such that the values of the respective common properties of $k$ and $j$ are exactly the same and
- for each data item $k$ in $r_i$ there exists at least data item $j$ in $r$ such that the values of the respective common properties of $k$ and $j$ are not the same.

If $x$ are all properties of the data items of $r_i$, then instead of $A_{x^+}$, the notation $A^+$ will represent the cardinality increasing operation.
Definition 2.5 Operation $\alpha(r_1, \ldots, r_i, \ldots, r_m)$, whose output is data container $r$ and $x$ are the common properties of $r$ and $r_i$, is a negation operation for input data container $r_i$ iff for each data item $k$ in $r_i$ there does not exist data item $j$ in $r$ such that the values of the respective common properties of $k$ and $j$ are the same.

If $x$ are all properties of the data items of $r_i$, then instead of $N_x+$, the notation $N+$ will represent the negation operation.

Example 2.1 In relational algebra, join is a cardinality decreasing operation for an input relation iff it reduces cardinality of the input relation in the output. Selection is also a cardinality decreasing operation for the input relation. Union is a cardinality increasing operation for an input relation iff the output includes tuples of the input relation and tuples of the other input relation. Projection is a property decreasing operation for the input relation. Difference is cardinality decreasing operation for an input relation iff it reduces cardinality of the relation in the output. Also, difference is negation operation for a particular input relation iff the output consists of some tuples of the other input relation except tuples of the particular input relation. A weak cardinality decreasing operation is the combination of both a cardinality decreasing and a cardinality increasing operations.

In general, in operation $\alpha(r_1, \ldots, r_i, \ldots, r_m)$, whose output is data container $r$ and $x$ are the common properties of $r$ and $r_i$, we will assume that $x$ are all properties of the data items of $r_i$. Otherwise, it will be stated explicitly. Thus, instead of using operation classes $E_x, E_x+, A_x+$ and $N_x+$, in order, the notations $E, E+, A+$ and $N+$ for the operation classes will be used.

Note that the symbol "+" of operation classes $E+, A+$ and $N+$ denotes that the operations also produce data items of other input data container(s) other than the data items of the particular input data container in the outputs.

In the property decreasing operation $(\pi_x), \pi$ may be any operation class such as $E, E+, A+$, or $N+$. If the input to a property decreasing operation is all data items of a data container, as a convention, $\pi_x$ is changed to $A_x$. Note that $A$ by itself is not an operation class.

2.2 Expression Trees

In a data integration expression, impacts of arguments on each other are very important in the optimisation of query processing. Labelling expression trees of data integration
expressions is a technique to detect impacts of the arguments on each other.

The data integration expression evaluates to data containers which are represented by arguments of the expression. Data sources of the data containers are generally distributed in different geographical locations. Although the data sources originate from heterogeneous component databases, there are no syntax or semantic heterogeneities between their corresponding data containers. Furthermore, the heterogeneities between data of data containers have already been resolved.

An expression tree [Maie83] is a tree structure that corresponds to a given expression. Leaf nodes of the tree represent the operand symbols and its internal nodes are labelled by operator symbols of the expression.

A path for a leaf node in an expression tree consists of all edges and nodes from the leaf node to the root node.

**Example 2.2** Figure 2.1 shows an expression tree of $\Psi_1(\Psi_2(\Psi_4(r_1, r_2), \Psi_5(r_3)), \Psi_3(\Psi_6(r_4), \Psi_7(r_5, r_6), \Psi_8(r_7)))$, where the root node is $\Psi_1$ and paths for leaf nodes $r_1$ and $r_4$ are depicted in thicker lines.

### 2.2.1 Classes of Composite Operations

As mentioned earlier, any operation in the database falls into an operation class for an input to the operation. The objective is to detect the class of each operation in a given expression for any input to the operation.

In an expression that consists of just one atomic operation, the class of the operation for each argument is available. For an expression that consists of a set of operations,
detecting the class of each operation for any particular input demands the application of a set of rules.

Consider an expression tree of $\Psi_1(\Psi_2(\Psi_4(r_1, r_2), \Psi_5(r_3)), \Psi_3(\Psi_6(r_4), \Psi_7(r_5, r_6), \Psi_8(r_7, r_8)))$ depicted in Figure 2.2. Operation nodes labelled by $\alpha$ and $\beta$ are composite operations. The class of operation $\alpha$ for each one of input arguments $r_1$, $r_2$, and $r_3$ is the class of operation $\Psi_2$ for the same input argument. In order to detect the class of a composite operation such as $\alpha$ for each input argument, the edges in the path of a leaf node corresponding to the input argument in the expression tree are labelled.

In an expression tree, the edges connected to an operation node are of two kinds. One is an input edge and the other one is an output edge.

In the path of an expression tree, output edges are labelled. The label of an output edge consists of an operation class such as $E$, $E+$, $A+$, $N+$, or $\pi_x$ and the label of the leaf node in the path. For example, if the operation class for an operation in the path of leaf node $r$ is $E+$, the output edge of the operation is labelled by $E(r)+$. In other words, the class of an operation for an input to the operation is identified by the label of the output edge.

2.3 Labelling Rules

In an expression tree, operation nodes connected to leaf nodes are atomic operations. Thus, the output edges of such operations can be labelled by the operation classes. In the tree, the operation nodes that are not connected to the leaf nodes are composite operations because their inputs are outputs of other operations. To discover the class of a composite operation with respect to a leaf node, the edges in the path of the leaf node are labelled. Labelling edges of the path starts from the edge connected to the
leaf node and it is continued to the output edge of the operation. Having the label of an operation node and the label of an input edge to the operation, the class of the operation for the leaf node can be obtained by using the labelling rules.

The labelling edges of a path in an expression tree obeys the following rules:

**Rule 2.1** In every path of an expression tree, any edge connected to leaf node \( r \) is labelled by \( A(r) \).

In Figure 2.1 all edges connected to leaf nodes \( r_1, \ldots, r_7 \), in order, can be labelled by \( A(r_1), \ldots, A(r_7) \).

**Rule 2.2** If, in a path, an input edge of a node is labelled by \( A(r) \) for data container \( r \) as the leaf node, the output edge is labelled by the operation class of the node.

For example, if an input edge of a cardinality increasing operation for data container \( r \) is labelled by \( A(r) \), the output edge is labelled by \( E(r) \).

**Rule 2.3** If, in a path, an input edge of a cardinality increasing operation is labelled by \( E(r) \) for data container \( r \), the output edge of the operation is labelled by \( E(r)^+ \).

**Justification:** Assume an input edge of a cardinality increasing operation is labelled by \( E(r) \). The input consists of some of the data items of data container \( r \). Since the operation is cardinality increasing one, the output consists of all data items of the input and data items of other input data container(s). Thus, the output consists of some data items of \( r \) and data items of other data container(s). Hence, the output edge is labelled by \( E(r)^+ \).

**Rule 2.4** Assume that, in a path, an input edge of a node is labelled by \( A(r)^+ \) for data container \( r \). If the node is:

1. a cardinality decreasing operation, then the output edge is labelled by \( E(r)^+ \).
2. a negation operation, then the output edge is labelled by \( N(r)^+ \).
3. a cardinality increasing operation, then the output edge is labelled by \( A(r)^+ \).

**Justification:** Assume that an edge in a path is labelled by \( A(r)^+ \). This means that all data items of data container \( r \) and data items of other data container(s) are input to the node in the path.
1. If the node is a cardinality decreasing operation, the operation removes some data items of input $r$ and some data items of the other data container(s) in the output. Thus, the output consists of some data items of data container $r$ and some data items of the other data container(s). Hence, the output edge is labelled by $E(r)+$.

2. If the node is a negation operation for the input, none of the data items of the input which are data items of $r$ and data items of other data container(s) arrive in the output of the operation. However, there are data items of other data container(s) other than data items of the input, some of which arrive in the output of the operation. Thus, the output edge is labelled by $N(r)+$.

3. If the node is a cardinality increasing operation, in addition to all data items of $r$ and the data items of other data container(s), the data items of other data container(s) other than data items of the input arrive in the output of the operation. Thus, the output edge of the node is still labelled by $A(r)+$.

**Rule 2.5** If, in a path, a node is a cardinality decreasing operation and its input edge is labelled by $E(r)$ for data container $r$, the output edge of the operation is also labelled by $E(r)$.

**Justification:** Assume that an input edge for a cardinality decreasing operation is labelled by $E(r)$ for data container $r$. The operation removes some data items of the input in the output. The output still consists of some data items of $r$. Thus, the output edge is also labelled by $E(r)$.

**Rule 2.6** If, in a path, a node is a negation operation and its input edge is labelled by $E(r)$ for data container $r$, the output edge of the operation is labelled by $E(r)+$.

**Justification:** Assume that the label of an input edge of a negation operation is $E(r)$ for data container $r$. Since the operation is negation, none of the data items of input which are some data items of $r$ arrive in the output of the operation. However, there are data items of other inputs to the operation other than those data items of $r$, some of which arrive in the output. Among these data items, there are some data items of $r$ other than those data items of the input edge with label $E(r)$. Thus, the output edge is labelled by $E(r)+$, which means that the output consists of some data items of $r$ (other than those data items in the input) and some data items of other data container(s).
Rule 2.7 If, in a path, a property decreasing operation projects $x$ properties, all data items of the input with the $x$ properties arrive in the output. In other words, if the input edge is labelled either by $A(r)$, $A(r)+$, $E(r)$, $E(r)+$, or $N(r)+$, the output edge is labelled either by $A_x(r)$, $A_x(r)+$, $E_x(r)$, $E_x(r)+$, or $N_x(r)+$, respectively.

Justification:

- If the label of the input edge is $A(r)$, the operation projects $x$ properties of all data items of $r$. Thus, the output edge is labelled by $A_x(r)$.
- If the label of the input edge is $E(r)$, the operation projects $x$ properties of some data items of $r$. Thus, the output edge is labelled by $E_x(r)$.
- If the label of the input edge is $A(r)+$, the operation projects $x$ properties of all data items of $r$ and data items of other data container(s). Thus, the output edge is labelled by $A_x(r)+$.
- If the label of the input edge is $E(r)+$, the operation projects $x$ properties of some data items of $r$ and data items of other data container(s). Thus, the output edge is labelled by $E_x(r)+$.
- If the label of the input edge is $N(r)+$, the operation projects $x$ properties of data items of other data container(s) other than data items of $r$. Thus, the output edge is labelled by $N_x(r)+$.

Rule 2.8 If, in a path, the label of an input edge of a node is $E(r)+$ for data container $r$ and the operation is any operation except a property decreasing one, the output edge is also labelled by $E(r)+$.

Justification: Assume an input edge of a node is labelled by $E(r)+$ for data container $r$.

- If the node is a cardinality decreasing operation, the output consists of some data items of the input. Since the input consists of some data items of $r$ and data items of other data container(s), some of those data items of the input arrive in the output. Thus, the output edge is also labelled by $E(r)+$.
- If the node is a negation operation, the output consists of none of the data items of the input. However, there are other inputs to the operation, some of which data items arrive in the output. These data items consists of some data items of $r$ and some data items of other data container(s) other than those data items of the input with label $E(r)+$. Thus, the output edge is still labelled by $E(r)+$. 
• If the node is a cardinality increasing operation, the output consists of all data items of the input and data items of other data container(s). Thus, the output is also labelled by $E(r)+$.

**Rule 2.9** If, in a path, an input edge of a cardinality decreasing operation is labelled by $N(r)+$ for data container $r$, the output edge of the operation is also labelled by $N(r)+$.

**Justification:** The input consists of data items of other data container(s) other than data items of $r$. Since the operation removes some data items of the input, in the output there will still be no data items of $r$ except some data items of other data container(s) other than data items of $r$. Thus, the output edge is also labelled by $N(r)+$.

**Rule 2.10** If, in a path, an input edge of either a negation or a cardinality increasing operation is labelled by $N(r)+$ for data container $r$, the output edge is labelled by $E(r)+$.

**Justification:** Assume that an input edge of an operation node in a path is labelled by $N(r)+$ for data container $r$.

• If the node is negation operation, the output of the operation consists of none of the data items of the input. The input consists of some data items of other data container(s) other than data items of $r$. None of these data items arrive in the output. However, the output consists of some data items. Some of them are data items of $r$ and the others are data items of other data container(s). Thus, the output edge is labelled by $E(r)-$.

• If the node is cardinality increasing operation, the output of the operation consists of data items of the input and data items of other data container(s) other than data items of the input. The input consists of some data items of other data container(s) other than data items of $r$. Since the node is cardinality increasing operation, some data items of data container(s) other than the data items of the input arrive in the output. Among these data items, there is at least a data item of $r$. Thus, the output edge is labelled by $E(r)+$.

Summary of the labelling edges of an expression tree for the database operations is given in Table 2.2. In labelling an edge in a path, the operation and label of its input edge must be on hand. In Table 2.2, the first column and the first row, in order, contains all kinds of labels for an argument such as $r$ and all database operations. The class of an operation for argument $r$ is identified by the operation and the label of
2.4 Example: Relational Algebra

<table>
<thead>
<tr>
<th>operation label</th>
<th>cardinality decreasing</th>
<th>cardinality increasing</th>
<th>property decreasing</th>
<th>negation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A(r)$</td>
<td>$E(r)$</td>
<td>$A(r)^+$</td>
<td>$A_2(r)^+$</td>
<td>$N(r)^+$</td>
</tr>
<tr>
<td>$A(r)^+$</td>
<td>$E(r)^+$</td>
<td>$A(r)^+$</td>
<td>$A_2(r)^+$</td>
<td>$N(r)^+$</td>
</tr>
<tr>
<td>$N(r)^+$</td>
<td>$N(r)^+$</td>
<td>$E(r)^+$</td>
<td>$E_2(r)^+$</td>
<td>$E(r)^+$</td>
</tr>
<tr>
<td>$E(r)$</td>
<td>$E(r)$</td>
<td>$E(r)^+$</td>
<td>$E_2(r)^+$</td>
<td>$E(r)^+$</td>
</tr>
<tr>
<td>$E(r)^+$</td>
<td>$E(r)^+$</td>
<td>$E(r)^+$</td>
<td>$E_2(r)^+$</td>
<td>$E(r)^+$</td>
</tr>
</tbody>
</table>

Table 2.2: The class of an operation for an argument such as $r$ is identified by the operation and the label of the input edge to the operation. Output edge of the operation is labelled by the class of that operation for argument $r$.

the input edge to the operation. Thus, if label of an input edge to operation $\alpha$ for argument $r$ is $\beta$, the cross of $\alpha$ and $\beta$ in the table will be the class of the operation for argument $r$. The output edge of the operation is labelled by the class of that operation for argument $r$.

2.4 Example: Relational Algebra

This section describes a special case of the generalised database model. The labelling technique is examined for relational algebra operations and expressions to detect the class of operations for the input relations.

Relational algebra consists of a collection of high-level operations that operate on relations. Each operation has either one or two relations as its input and produces a new relation as the output. The database consists of a collection of relations as its data containers. Data items are tuples of relations and properties of the data items are attributes of tuples.

Operations of relational algebra are join ($\times$), intersection ($\cap$), difference ($-$), union ($\cup$), projection ($\pi$) and selection ($\sigma$).

An expression of relational algebra can be represented by an expression tree corresponding to the expression. Difference is a binary operation. Since it is a non-commutative operation, for each one of the input relations, the operation falls into two different classes. If an input to a difference operation is on the left hand side or the input is on the right hand side, the operation is represented, in order, by "$l-$" or by "$r-$" (Figure 2.3(a) and (b)).

Observation 2.1 In labelling edges in a path of an expression tree, $\times$, $\cap$, $l-$ and $\sigma$
are cardinality decreasing operations.

Justification for the operations being cardinality decreasing ones are as follows:

(join) A join operation has two input relations. Output of the operation consists of concatenated those tuples of both inputs that can be joined. In joining tuples of the inputs, some tuples of the inputs are not joinable. The output consists of concatenation of some tuples of the inputs. It means that in the output, only some tuples of each input arrive. Thus, join is a cardinality decreasing operation.

(intersection) Output of an intersection operation consists of intersection of both inputs' tuples. Some tuples of the inputs have no intersection. In the output, some tuples of each input arrive. Thus, intersection is a cardinality decreasing operation.

(difference) Output of a difference operation consists of tuples of the left hand side input relation to the operation that have no intersection with tuples of the right hand side input relation. The output consists of some tuples of the left hand side input argument. Thus, l− is a cardinality decreasing operation.

(selection) A selection operation removes some tuples of its input. Output of the operation consists of some tuples of the input. Thus, selection is cardinality decreasing operation.

Observation 2.2 In labelling edges in a path of an expression tree, π is a property decreasing operation.

Output of a projection operation consists of projection of some attributes of all tuples of the input. Thus, projection operation is a property decreasing operation.
2.5 Summary

In the generalised database model, data are grouped into collections of data containers. A data container consists of a set of homogeneous data items. Each data item is described by a set of properties.

**Observation 2.3** In labelling edges in a path of an expression tree, $\neg$ is a negation operation.

Justification for such an observation is that, in the output of a difference operation, no tuples of the right hand side input relation will appear. In the output, some tuples of the left hand side input relation arrive. Thus, $\neg$ operation is a negation operation.

**Observation 2.4** In labelling edges in a path of an expression tree, $\cup$ is a cardinality increasing operation.

Justification for such an observation is that, output of a union operation consists of the union of all tuples of both input relations to the operation. Thus, the operation is a cardinality increasing operation for each one of its inputs.

**Example 2.3** Figure 2.4 shows an expression tree of $((r_1 \cup r_2) - (\sigma_\phi (r_3))) \cup ((r_4 - r_5) \Join r_6)$. Figure 2.5 shows the same expression tree where the difference operation nodes have been changed to $\neg$ and $\neg$ because input edges of each difference operation in the paths for $r_1$ and $r_5$, in order, are in the left and in the right hand sides of the operations. In Figure 2.5, edges in the paths for leaf nodes $r_1$ and $r_5$ are labelled.
In the generalised database, inputs to operations are data containers. Each operation produces a new data container as the output. An operation in the database is classified with respect to one of its input data containers. Operations in the database are divided into five general classes.

An expression in the database can be represented by an expression tree. Expression trees are used for two dimensional representation of expressions.

In an expression tree, a path consists of all edges and nodes from a leaf node to the root. Labelling edges in a path permits the discovery of the class of an operation in the path for the leaf node.

In a path of an expression tree, output edges are labelled. The label of an output edge consists of an operation class such as $E$, $E^+$, $A^+$, $N^+$, or $\pi_x$ and the label of the leaf node in the path. For example, if the operation class for an operation in a path of a leaf node $r$ is $E^+$, the output edge of the operation is labelled by $E(r)^+$. For an expression that consists of just one atomic operation, the class of the operation with respect to each argument is available. For an expression that consists of a set of operations, detecting the class of each operation for any particular input requires application of a set of rules.

In an expression tree, operation nodes connected to the leaf nodes are atomic operations. Thus, output edges of such operations can be labelled by the operation classes. In the tree, the operation nodes that are not connected to the leaf nodes are composite operations because their inputs are outputs of other operations. To discover the class of a composite operation with respect to a leaf node, the edges in the path of the leaf node are labelled. Labelling edges of the path starts from the edge connected to the leaf node and it is continued to the output edge of the operation.

Labelling edges of a path in an expression tree obeys a set of rules. Having the label of an operation node and the label of an input edge to the operation, the class of
the operation for the leaf node can be obtained from a table.
Optimisation Based on Reductions

It may be possible to reduce the size of arguments of a data integration expression before integration. This requires the detection of what impacts the arguments have on each other and how much benefit can be obtained in the reductions.

This chapter provides some applications of the labelling theory in query optimisation in database systems. Labelling technique can be applied to detecting optimisation of query processing not only to heterogeneous distributed multidatabase systems but also to centralised and to homogeneous distributed database systems.

Labelling an expression tree of the data integration expression is a step toward detecting the impacts of arguments on each other. After labelling the expression tree, the impacts can be detected.

As a particular case for the application of labelling technique in optimisation of query processing, relational algebra expressions are examined to detect impacts of relations on each other.

3.1 Reductions

In a data integration expression, impacts of arguments on each other are important for reducing their size before integration.

**Definition 3.1** In expression \(e(r_1, \ldots, r_i, \ldots, r_j, \ldots, r_n)\), argument \(r_j\) reduces argument \(r_i\) if it is possible to find an expression as \(\alpha(r_i, r_j)\) such that:

1. \(eval(e(r_1, \ldots, \alpha(r_i, r_j), \ldots, r_j, \ldots, r_n)) = eval(e(r_1, \ldots, r_i, \ldots, r_j, \ldots, r_n))\).
2. \(eval(\alpha(r_i, r_j)) \subseteq r_i\).

Operation \(\alpha\) on \(r_i\) and \(r_j\) will be called the reduction of \(r_i\) by \(r_j\). Also, the expression \(\alpha(r_i, r_j)\) will be called a reduction expression.
3.1. Reductions

In expression $e$, argument $r_i$ may also reduce argument $r_j$ in a reduction expression such as $\beta(r_i, r_j)$, with the conditions that are mentioned above. Thus, if both arguments can be reduced by each other, there exists mutual or full reduction between the arguments. If only one of the arguments can be reduced by another one, there exists partial reduction. Furthermore, if no reduction expressions can be found, the arguments are independent for any reduction.

Example 3.1 Consider relational algebra expression $e = (r_1 - \pi_x(r_2 \bowtie r_3))$. It is possible to replace argument $r_2$ with reduction expression $r_2 \bowtie \pi_x(r_1)$. After replacing argument $r_2$ with the reduction expression, the evaluation result of $e$ is not changed and the size of argument $r_2$ is reduced.

Two arguments of an expression are adjacent if they are operands of an operation. Otherwise, they are called non-adjacent arguments. When two arguments of an expression are non-adjacent, we detect the reduction. If two arguments are adjacent, there exists full reduction between them because each one of the arguments can be replaced by a reduction expression. A trivial example is relational expression $r_1 \bowtie r_2$. Argument $r_1$ can be replaced by $r_1 \bowtie r_2$ and argument $r_2$ can be replaced by $r_2 \bowtie r_1$.

Definition 3.2 In an expression, a common operation for arguments $r_1, \ldots, r_k$ is a root of the smallest subtree in an expression tree of the expression that contains the arguments.

Example 3.2 An expression tree of $\Psi_1(\Psi_2(\Psi_4(r_1, r_2), \Psi_5(r_3)), \Psi_3(\Psi_6(r_4), \Psi_7(r_5, r_6), \Psi_8(r_7)))$ is shown in Figure 3.1. The common operation for arguments $r_1$ and $r_2$ is $\Psi_4$. For arguments $r_4$ and $r_7$, the common operation is $\Psi_3$. Also, the common operation for $r_1$ and $r_4$ is $\Psi_1$.

3.1.1 Beneficial Reductions

In an expression, the amount of savings for reducing the size of an argument in a reduction operation depends on how much time can be saved in computation of the expression. The time for performing the reduction operation must also be compensated in the savings.

Let $e(r_1, \ldots, r_i, \ldots, r_j, \ldots, r_n)$ be an expression in which $r_i$ and $r_j$ are non-adjacent arguments. Assume there is a partial reduction between arguments $r_i$ and $r_j$. Let the reduction expression that reduces $r_j$ by $r_i$ be $\Psi_{ji}(r_i, r_j)$. 
3.1. Reductions

Suppose that $\Psi$ be the common operation for $r_i$ and $r_j$. Figure 3.2 shows an expression tree of $e$, where operations $\Psi_1, \Psi_2, \ldots, \Psi_m$ are in the path of argument $r_j$ to the common operation. The operations in the path of $r_i$ are not shown.

Assume that the time for performing operations in the path of $r_j$ is $t_\Psi$. By replacing $r_j$ with reduction expression $\Psi_{ji}(r_i, r_j)$, expression $e$ is transformed to $e'(r_1, \ldots, r_i, \ldots, \Psi_{ji}(r_i, r_j), \ldots, r_n)$. An expression tree of $e'$ is depicted in Figure 3.3.

Assume that the time for performing reduction operation $\Psi_{ji}$ is $t_{\Psi_{ji}}$. Also, after execution of the reduction operation, let the time for performing the operations in the path of $r_j$ be $t_{r_j}'$. Computation of the reduction expression $\Psi_{ji}(r_i, r_j)$ reduces the size of $r_j$ to $r_j'$ where $r_j' \subset r_j$. In this way, reduction of an argument such as $r_j$ may have an impact on the computation times of the operations in the path of $r_j$.

If $t_{\Psi}' < t_\Psi$, then the amount of savings in the computation of operations in the path of $r_j$ is $t_\Psi - t_{\Psi}'$. If the savings cannot compensate the reduction operation time or the savings can merely compensate the reduction operation time (i.e. $t_\Psi - t_{\Psi}' \leq t_{\Psi_{ji}}$), the reduction operation is not beneficial. If $t_\Psi - t_{\Psi}' > t_{\Psi_{ji}}$, then the reduction operation is
3.1. Reductions

Figure 3.3: An expression tree of $e$, where one of the leaf nodes is replaced by reduction expression $\Psi_{ji}(r_i, r_j)$.

beneficial. Specifically, when $t_\Psi$ is much more than $t'_\Psi + t_{\Psi_{ji}}$, the reduction operation is very beneficial.

When the size of argument $r_j$ is reduced by reduction operation $\Psi_{ji}$, those data items of $r_j$ are removed that have no impacts on the evaluation result of $\Psi$ operation. In other words, if the reduction operation were not applied to reduce the size of $r_j$, some parts of those data items of $r_j$ that could be removed in the reduction operation might arrive in the output of operations $\Psi_1, \Psi_2, \ldots$ and finally in the output of operation $\Psi_m$ (see Figure 3.3). Furthermore, if all those data items of $r_j$ arrive in the output of operation $\Psi_m$, they have no impacts on the evaluation result of $\Psi$.

Not all reductions between any two arguments of a data integration expression are beneficial. There may be a case that if reduction between two arguments of an expression is applied, then the computation time of the expression increases because of the extra time for performing the reduction operation.

Example 3.3 In the relational algebra expression $(r_1 \cap r_2) \bowtie (r_3 - r_4)$ whose expression tree is depicted in Figure 3.4, relation $r_3$ can be replaced by $r_3 \times r_2$. Thus, the size of $r_3$ is reduced. As a result, the time for performing the difference operation is reduced. Furthermore, the time for performing the join operation will be reduced because the number of tuples in the output of the difference operation is reduced.

3.1.2 Detection of Reductions

In an expression, an input to an operation may be the output of another operation. In the corresponding expression tree of the expression, the label of the output edge with respect to a leaf node is the class of the operation for the leaf node.
To detect reduction between two arguments of an expression, it is necessary:

- to identify the common operation for the arguments and

- to find the class of the operation whose output is the input to the common operation for each one of the arguments in the corresponding expression tree of the expression.

After identifying the common operation for the two arguments, labelling technique is used to label the paths of the arguments in the corresponding expression tree to find the class of the operation whose output is the input to the common operation node for each one of the arguments. Figure 3.5 shows an expression tree, where $\Psi$ is the common operation for arguments $r$ and $s$. The labels $L_r$ and $L_s$ are the classes of the operations whose outputs are the inputs to $\Psi$ for arguments $r$ and $s$, respectively. In other words, the labels $L_r$ and $L_s$, in order, are the labels of the input edges to $\Psi$ for arguments $r$ and $s$.

For each operation in the database, a table is designated called the reduction expression table. The first row and the first column of the table contain all operation
3.2. Application of Reductions

Reductions between arguments of an expression can be employed as a general technique for database query optimisation.

The problem of query optimisation can be stated as follows: Given an expression with reductions between its arguments, it must be decided how the expression can be computed faster by applying the reductions.

3.2.1 Classical Optimisation

In the classical optimisation, an expression (query or update) can be transformed to an equivalent one by applying some transformational rules. Generally, such transformations are useful because the computation time of the expression is reduced.

By replacing arguments of the expression with reduction expressions, the evaluation result of the expression is not changed. Thus, if there are reductions between arguments of the expression, the beneficial ones can be applied by replacing arguments with the relating reduction expressions. In this way, the computation time of the expression can be decreased.

Example 3.4 Suppose Figure 3.6 shows an expression tree of a relational algebra expression, where operations in the path of leaf node $r_i$ are not shown. Inputs to the first difference operation from bottom of the tree are relations $r_j$ and $r_k$, with 1500 and 1000 tuples, respectively. The operation produces 500 tuples in the output. The second difference operation takes the 500 tuples and the tuples of relation $r_m$ and produces 200 tuples. After the intersection operation is performed, 150 tuples are produced which are the input to the join operation.

Assume that $r_i$ reduces $r_j$, expressed by the reduction expression $\Psi_{ji}(r_i, r_j)$. The reduction operation $\Psi_{ji}$ reduces the number of tuples of $r_j$ from 1500 to 500 tuples (Figure 3.7). Thus, instead of 1500 tuples of relation $r_j$ as an input to the first difference operation, 500 tuples are the input to the operation. It means that the time for
3.2. Application of Reductions

Figure 3.6: An expression tree of a relational expression.

Figure 3.7: An expression tree of a relational expression in which \( r_j \) is replaced by reduction expression \( \Psi_{ij}(r_i, r_j) \).

performing the first difference operation is reduced because relation \( r_j \) is reduced. Similarly, the time for performing the second difference operation, the intersection operation and finally the time for performing the join operation are reduced.

3.2.2 Homogeneous Distributed Database Systems

In homogeneous distributed database systems, the reductions between arguments of an expression can be used in the local processing and in the final processing steps. In the local processing step, the reductions are applied to reduce the size of partial results in the local databases. In the local processing step, beneficial reductions are those whose costs are less than the benefits.

Depending on what parameters are important in the query processing, the costs of
3.2. Application of Reductions

Applying a reduction operation may consist of the cost for projecting the required data in one site (the first site), the cost of transmitting the data to another site (the second site) and the cost of performing the reduction operation in the second site. The benefit of the reduction operation may be the reductions in the size of data in the second site, needed to be transmitted to the first site and the amount of savings in computations. A trivial example of applying reductions in the systems is a semijoin operation.

**Example 3.5** Consider relational expression \( e = r_1 \rightarrow (r_2 \bowtie r_3) \), where relation \( r_2 \) is available in the submission site and relations \( r_2 \) and \( r_3 \) must be obtained from other sites, e.g. sites \( d_2 \) and \( d_3 \), respectively. Relations \( r_2 \) and \( r_3 \) can reduce the size of each other, in order, by the reduction expressions \( r_2 \bowtie \pi_x(r_3) \) and \( r_3 \bowtie \pi_x(r_2) \), where \( x \) are the join attributes. Furthermore, relation \( r_1 \) can reduce the size of relations \( r_2 \) and \( r_3 \). The reduction expressions for \( r_2 \) and \( r_3 \), in order, are \( r_2 \bowtie \pi_y(r_1) \) and \( r_3 \bowtie \pi_z(r_1) \). Attributes \( y \) are common attributes between \( r_1 \) and \( r_2 \) and attributes \( z \) are common attributes between \( r_1 \) and \( r_3 \).

To reduce the size of \( r_2 \) and \( r_3 \) by \( r_1 \), the contents of \( \pi_y(r_1) \) and \( \pi_z(r_1) \), in order, are transmitted to sites \( d_2 \) and \( d_3 \). Also, to reduce the size of \( r_2 \) and \( r_3 \) by each other, \( \pi_x(r_3) \) from site \( d_3 \) and \( \pi_x(r_2) \) from site \( d_2 \), in order, are transmitted to sites \( d_2 \) and \( d_3 \). Those reductions whose costs are less than the benefits can be performed.

In the final processing step, if the size of results obtained from query processing in local databases are not completely reduced, similar to the centralised database systems, the beneficial reductions can be used to reduce the integration time.

### 3.2.3 Heterogeneous Distributed Multidatabase Systems

In HDMDBs, reductions between arguments of a data integration expression are very important not only in the efficient processing of subqueries in component databases but also in the computation of the data integration expression.

After a query in HDMDBs is decomposed into subqueries and then translated to query language of component databases, three strategies can be applied for submission of the subqueries to the component databases as follows:

1. Sequential submission,
2. parallel submission and
3. hybrid (a mixture of sequential and parallel) submission.
If submission of each subquery is delayed until partial results of previously submitted subqueries are available, the strategy is called \textit{sequential submission}. In this strategy, the partial results are used in the processing of the subquery in the relating component database.

If all subqueries are submitted to the component databases at the same time, they are processed in the component databases in parallel and the results may be transmitted in parallel to the submission site. This strategy is called \textit{parallel submission}. \textit{Hybrid submission} of subqueries is when some of the subqueries are submitted in parallel, and the others are submitted sequentially to the component databases.

For the time being, we will focus on how the reductions can be useful in reducing the size of partial results in the component databases and also in the submission site. Merits of the strategies to each other are considered elsewhere.

In the sequential subquery submission strategy, reductions between the arguments of a data integration expression are important factors in the optimisation of query processing. Reductions between arguments are used to discover what subquery should be submitted first and what subquery should use the results of previously submitted subqueries. Furthermore, the reductions determine which subquery can use the results of other subqueries to, at most, reduce the size of query processing result in the component database.

At first, one subquery is selected to be submitted to the relating component database. After a partial result of the subquery is available, the result is used by the second subquery for processing in the pertinent component database. The third subquery uses the obtained partial results for processing and so on.

Figure 3.8 shows how \( n \) subqueries \((q_1, q_2, q_3, \ldots, q_n)\), which are the result of decomposition of a global query, are submitted to the component databases, sequentially. First, subquery \( q_1 \) among the subqueries is chosen and submitted to component database \( DB_{i_1} \), because its results can reduce the size of other partial results the most. After the partial result of \( q_1 \) which is \( r_{i_1} \) is available, the second subquery \( q_{i_2} \) uses \( \Phi(r_{i_1}) \), which is the required data of \( r_{i_1} \) for reduction, and then the subquery is submitted to component database \( DB_{i_2} \). The submission of subquery \( q_{i_n} \) is delayed until partial results of other subqueries are available. Note that except for the partial result of the first subquery, we have:

\[
    r'_{i_2} \subseteq r_{i_2}, \ r'_{i_3} \subseteq r_{i_3}, \ldots, \ r'_{i_n} \subseteq r_{i_n}, \text{ where } r_{i_2}, r_{i_3}, \ldots, r_{i_n} \text{ are partial results of subqueries, if each one of them does not use the partial results of the others for processing. The partial results } r'_{i_2}, r'_{i_3}, \ldots, r'_{i_n} \text{ are the reduced size of partial results after each subquery}
\]
3.2. Application of Reductions

Figure 3.8: Sequential subquery processing.

Figure 3.9: Parallel subquery processing.

uses the partial results of other subqueries.

In the parallel subquery submission strategy (Figure 3.9), all subqueries are submitted to the component databases at the same time. Although they are processed in parallel, their partial results are not available in the submission site at the same time. There are several reasons why some partial results are available earlier and the others are available later. Autonomy of the component databases, network congestion, data transmission speeds, complexity of subqueries processing and the size of partial results are some of the reasons.

During the time that the system is waiting until all partial results are available, it is possible to use reductions between arguments whose corresponding partial results are available to reduce their size. In this way, the sizes of available arguments are reduced before other arguments become available.

Figure 3.9 shows $n$ subqueries ($q_1$, $q_2$, $q_3$, ..., $q_n$), which are the result of the decomposition of a global query, are submitted to the component databases in parallel. Integration of partial results ($r_1$, $r_2$, ..., $r_i$, ..., $r_j$, ..., $r_n$) begins when all of them are available. All subqueries are submitted at the time of $t_0$. The last partial result is available at the time of $t_m$. During the time $t_0$ to $t_m$, reductions are used to reduce the size of those already available partial results.

In the hybrid submission of subqueries, the same mechanisms are used to reduce the size of partial results. For the subqueries that are submitted sequentially, the reductions are used to reduce the size of partial results in component databases. For the subqueries
whose the partial results are available in the submission site, the reductions are used to reduce their size in the submission site.

3.3 Reductions in Relational Algebra Expressions

In this section, reductions between arguments of relational algebra expressions are explored. We will consider reductions between relations, where the common operation for the relations is join, intersection, difference or union operation.

We use two reduction operations, called semijoin and generalised difference whose definitions are given as follows.

**Definition 3.3** A semijoin of relational tables \( r \) and \( s \) is denoted by \( r \times s \). Its results are all tuples of \( r \) that can be joined with tuples of \( s \).

**Definition 3.4** A generalised difference of relational tables \( r \) and \( s \) is denoted by \( r \_k s \). Its results are the tuples of \( r \) that cannot be joined with any tuples of \( s \).

Note that \( k \) are the join attributes. If \( k \) is omitted from the notation of the generalised difference, then the common attributes between \( r \) and \( s \) are the join attributes.

The generalised difference does not extend the expressive power of relational algebra. It is only used as a shorthand notation, because: \( r \_k s = r \setminus (r \times \pi_k(s)) \).

3.3.1 Join Operation

In a relational expression, the common operation for two relational tables \( r \) and \( s \) is join. In the expression tree of the expression, depending on the labels of the input edges for the operation with respect to the relations, the reduction expressions can be obtained from Table 3.1. Note that when there are no reductions between arguments, they are indicated by “NIL”.

**Lemma 3.3.1** In an expression when a common operation between two arguments \( r \) and \( s \) is join and the labels of the input edges for the common operation node in the corresponding expression tree are \( A_x(r) \) and \( A_y(s) \), argument \( r \) can reduce argument \( s \). The reduction expression for \( s \) is \( s \times r \). For \( r \), there is no reduction expression.

**Proof:**
3.3. Reductions in Relational Algebra Expressions

<table>
<thead>
<tr>
<th>⊗</th>
<th>$A_y(s)$</th>
<th>$A_y(s)^+$</th>
<th>$N_y(s)^+$</th>
<th>$E_y(s)$</th>
<th>$E_y(s)^+$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_x(r)$</td>
<td>$r = r \times s$</td>
<td>$s = s \times r$</td>
<td>$s = s \pi_k(r)$</td>
<td>$r = r \pi_k(s)$</td>
<td>$s = s \pi_k(r)$</td>
</tr>
<tr>
<td>$A_x(r)^+$</td>
<td>$r = r \times s$</td>
<td>$s = s \times r$</td>
<td>$s = s \pi_k(r)$</td>
<td>$r = r \pi_k(s)$</td>
<td>$s = s \pi_k(r)$</td>
</tr>
<tr>
<td>$N_x(r)^+$</td>
<td>$r = r \pi_k(s)$</td>
<td>NIL</td>
<td>NIL</td>
<td>$r = r \pi_k(s)$</td>
<td>NIL</td>
</tr>
<tr>
<td>$E_x(r)$</td>
<td>$r = r \pi_k(s)$</td>
<td>$s = s \pi_k(r)$</td>
<td>$s = s \pi_k(r)$</td>
<td>$r = r \pi_k(s)$</td>
<td>$s = s \pi_k(r)$</td>
</tr>
<tr>
<td>$E_x(r)^+$</td>
<td>$r = r \pi_k(s)$</td>
<td>NIL</td>
<td>NIL</td>
<td>$r = r \pi_k(s)$</td>
<td>NIL</td>
</tr>
</tbody>
</table>

Table 3.1: Reduction expressions for join operation ($k = x \cap y$).

- Relation $s$ can be reduced to those tuples that can be joined with tuples of $r$ because all tuples of $r$, with $x$ attributes, are input to the join operation for the edge labelled by $A_x(r)$. Thus, $s$ can be reduced to $s \times r$.
- Relation $r$ cannot be reduced because not only are all tuples of $s$, with $y$ attributes, input to the join operation but also the tuples of other relational table(s) are. There may be a tuple in $r$ that cannot be joined with any tuples of $s$ but it can be joined with the tuples of other relational table(s). Thus, $r$ cannot be reduced.

Proofs of the remaining reduction expressions in Table 3.1 are given in the appendix.

Example 3.6 An expression tree corresponding to relational expression $(r_1 \cap r_2) \times ((r_3 - r_4) \cup r_5)$ is depicted in Figure 3.10. The edges for the paths of $r_1$ and $r_3$ in the tree are labelled. The labels of the input edges to the common operation node are $E(r_1)$ and $E(r_3)^+$. With respect to the labels and Table 3.1, the reduction between arguments $r_1$ and $r_3$ is $r_3 \times r_1$. In fact, the reduction expression for $r_3$ is $r_3 \times r_1$, and for $r_1$, there is no reduction expression.

3.3.2 Difference Operation

In a relational expression, difference is a common operation for two relational tables $r$ and $s$. In the expression tree of the expression, depending on the labels of the input edges for the operation with respect to the relations, the reduction expressions can be obtained from Table 3.2.

Note that difference operation is not commutative. Hence, arguments $r$ and $s$, in order, are considered as the left hand side and the right hand side inputs to the common operation.
3.3. Reductions in Relational Algebra Expressions

![Figure 3.10: A labelled expression tree of relational expression \((r_1 \cap r_2) \setminus ((r_3 - r_4) \cup r_5)\) for paths of \(r_1\) and \(r_3\).]

<table>
<thead>
<tr>
<th>Rule</th>
<th>(A_y(s))</th>
<th>(A_y(s)^+)</th>
<th>(N_y(s)^+)</th>
<th>(E_y(s))</th>
<th>(E_y(s)^+)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A_x(r))</td>
<td>(r = r - s)</td>
<td>(r = r - s)</td>
<td>(s = s \times r)</td>
<td>(s = s \times r)</td>
<td>(s = s \times r)</td>
</tr>
<tr>
<td>(A_x(r)^+)</td>
<td>(r = r - s)</td>
<td>(r = r - s)</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
<tr>
<td>(N_x(r)^+)</td>
<td>either (r = r - s) or (s = s \times r)</td>
<td>either (r = r - s) or (s = s \times r)</td>
<td>(s = s \times r)</td>
<td>NIL</td>
<td>NIL</td>
</tr>
<tr>
<td>(E_x(r))</td>
<td>(s = s \times r)</td>
<td>(s = s \times \pi_k(r))</td>
<td>(s = s \times \pi_k(r))</td>
<td>(s = s \times \pi_k(r))</td>
<td>(s = s \times \pi_k(r))</td>
</tr>
<tr>
<td>(E_x(r)^+)</td>
<td>(r = r -_k s)</td>
<td>(r = r -_k s)</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
</tbody>
</table>

Table 3.2: Reduction expressions for difference operation \((k = x \cap y)\).

**Lemma 3.3.2** In an expression when a common operation between two arguments \(r\) and \(s\) is difference and the labels of the input edges for the common operation node in the corresponding expression tree are \(A_x(r)\) for the left hand side edge and \(A_y(s)^+\) for the right hand side edge, arguments \(r\) and \(s\) can reduce each other. The reduction expression for \(r\) is \(r - s\) and for \(s\) the reduction expression is \(s \times r\).

**Proof:**

- Relation \(r\) can be reduced to \(r - s\) because the output of the difference operation consists of tuples of \(r\) that are not joinable with tuples of \(s\).
- Relation \(s\) can be reduced to \(s \times r\) because those tuples of \(s\) have impacts on the difference operation that are joinable with tuples of \(r\). Thus, \(s\) can be reduced to \(s \times r\).

Proofs of the remaining reduction expressions in Table 3.2 are given in the appendix.
3.3. Reductions in Relational Algebra Expressions

Example 3.7 An expression tree corresponding to relational expression \(((r_1 \cap r_2) - r_3) - ((r_4 \Join r_5) \cap r_6)\) is depicted in Figure 3.11. The edges of paths for \(r_1\) and \(r_6\) in the tree are labelled. The labels of input edges to the common operation node are \(E(r_1)\) and \(E(r_6)\). With regards to Table 3.2 and the labels, reduction for \(r_1\) and \(r_6\) is \(r_6 \prec r_1\). In fact, the reduction expression for \(r_6\) is \(r_6 \prec r_1\), and for \(r_1\), there is no reduction expression in the table.

For arguments \(r_3\) and \(r_6\) of relational expression \(((r_1 \cap r_2) - r_3) - ((r_4 \Join r_5) \cap r_6)\), there are no reductions. This fact is shown by labelling paths for the corresponding arguments in the expression tree depicted in Figure 3.12. The cross of labels of input edges (i.e. \(N(r_3)+\) and \(E(r_6)\)) to the common operation (i.e. \(-\)) in Table 3.2 is NIL.

3.3.3 Union Operation

In a relational expression, union is a common operation for two relational tables \(r\) and \(s\). In the expression tree of the expression, depending on the labels of the input edges for the operation with respect to the relations, the reduction expressions can be obtained from Table 3.3.
3.3. Reductions in Relational Algebra Expressions

Table 3.3: Reduction expressions for union operation ($k = x \cap y$).

<table>
<thead>
<tr>
<th>Operator</th>
<th>$A_y(s)$</th>
<th>$A_y(s) +$</th>
<th>$N_y(s) +$</th>
<th>$E_y(s)$</th>
<th>$E_y(s) +$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\cup$</td>
<td>$A_x(r)$</td>
<td>either</td>
<td>either</td>
<td>$s = s - r$</td>
<td>$s = s - k r$</td>
</tr>
<tr>
<td></td>
<td>$r = r - s$</td>
<td>$r = r - s$</td>
<td>or</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>or</td>
<td>or</td>
<td>$s = s - r$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_x(r) +$</td>
<td>either</td>
<td>either</td>
<td>$s = s - r$</td>
<td>$s = s - k r$</td>
<td>$s = s - k r$</td>
</tr>
<tr>
<td></td>
<td>$r = r - s$</td>
<td>$r = r - s$</td>
<td>or</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>or</td>
<td>or</td>
<td>$s = s - r$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N_x(r) +$</td>
<td>$r = r - s$</td>
<td>$r = r - s$</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
<tr>
<td>$E_x(r)$</td>
<td>$r = r - k s$</td>
<td>$r = r - k s$</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
<tr>
<td>$E_x(r) +$</td>
<td>$r = r - k s$</td>
<td>$r = r - k s$</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
</tbody>
</table>

Lemma 3.3.3 In an expression when a common operation between two arguments $r$ and $s$ is union and the labels of the input edges to the common operation node in the corresponding expression tree are $A_x(r)$ and $A_y(s) +$, either $r$ can reduce $s$ or $s$ can reduce $r$. The reduction expression is either $r - s$ for $r$ or $s - r$ for $s$.

Proof: Duplicate tuples are removed in the output of a union operation. Since labels of input edges to the common operation node are $A_x(r)$ and $A_y(s) +$, it means that the output of the union operation consists of all tuples of $r$, all tuples of $s$ and tuples of other relational table(s), all of which have no duplicate tuples in the output. Thus, duplicate tuples can be removed either from $r$ or $s$. In other words, either $r$ or $s$ can be reduced.

Proofs of the remaining reduction expressions in Table 3.3 are given in the appendix.

Example 3.8 An expression tree of relational expression $r_1 \cup (r_2 - (r_3 - r_4))$ is depicted in Figure 3.13. Edges of paths for $r_1$ and $r_4$ in the tree are labelled. Labels of input edges to the common operation node are $A(r_1)$ and $E(r_4) +$. With respect to Table 3.3 and the labels, reduction for $r_1$ and $r_4$ is $r_4 - r_1$. In fact, the reduction expression for $r_4$ is $r_4 - r_1$ and for $r_1$, there is no reduction expression in the table.

3.3.4 Intersection Operation

In a relational expression, intersection is a common operation for two relational tables $r$ and $s$. In the expression tree of the expression, depending on the labels of the input
3.3. Reductions in Relational Algebra Expressions

Figure 3.13: A labelled expression tree of relational expression \( r_1 \cup (r_2 - (r_3 - r_4)) \) for paths of \( r_1 \) and \( r_4 \).

<table>
<thead>
<tr>
<th>( \cap )</th>
<th>( A_y(s) )</th>
<th>( A_y(s)+ )</th>
<th>( N_y(s)+ )</th>
<th>( E_y(s) )</th>
<th>( E_y(s)+ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_x(r) )</td>
<td>( r = r \times s )</td>
<td>( s = r \times s )</td>
<td>( s = s \times r )</td>
<td>( r = r \times \pi_{k}(s) )</td>
<td>( s = s \times r )</td>
</tr>
<tr>
<td>( A_x(r)+ )</td>
<td>( r = r \times s )</td>
<td>NIL</td>
<td>NIL</td>
<td>( r = r \times \pi_{k}(s) )</td>
<td>NIL</td>
</tr>
<tr>
<td>( N_x(r)+ )</td>
<td>( r = r \times s )</td>
<td>NIL</td>
<td>NIL</td>
<td>( r = r \times \pi_{k}(s) )</td>
<td>NIL</td>
</tr>
<tr>
<td>( E_x(r) )</td>
<td>( r = r \times \pi_{k}(s) )</td>
<td>( s = r \times \pi_{k}(r) )</td>
<td>( s = s \times \pi_{k}(r) )</td>
<td>( r = r \times \pi_{k}(s) )</td>
<td>( s = s \times \pi_{k}(r) )</td>
</tr>
<tr>
<td>( E_x(r)+ )</td>
<td>( r = r \times \pi_{k}(s) )</td>
<td>NIL</td>
<td>NIL</td>
<td>( r = r \times \pi_{k}(s) )</td>
<td>NIL</td>
</tr>
</tbody>
</table>

Table 3.4: Reduction expressions for intersection operation \((k = x \cap y)\).

edges for the operation with respect to the relations, the reduction expressions can be obtained from Table 3.4.

**Lemma 3.3.4** In an expression when a common operation between two arguments \( r \) and \( s \) is intersection and the labels of the input edges to the common operation node in the corresponding expression tree are \( A_x(r) \) and \( A_y(s)+ \), argument \( r \) can reduce argument \( s \). The reduction expression for \( s \) is \( s \times r \).

**Proof:**

- Relation \( s \) can be reduced to those tuples that are joinable with tuples of \( r \). Since all tuples of \( r \) are as one of the input to the intersection operation, \( s \) can be reduced to \( s \times r \).

- Relation \( r \) cannot be reduced because not only all tuples of \( s \) flow to the intersection operation but also tuples of other relational table(s) other than \( s \). There may be a tuple in \( r \) that cannot be joined with any tuples of \( s \) but the tuple can be joined with the tuples of other relational table(s) other than \( s \). Thus, \( r \) cannot be reduced.
3.4 Reduction Operations in Relational Algebra

Proofs of the remaining reduction expressions in Table 3.4 are given in the appendix.

**Example 3.9** An expression tree of relational expression \((r_1 \bowtie r_2) \cap ((r_3 \bowtie r_4) - r_5)\) is depicted in Figure 3.14. The edges in paths of \(r_1\) and \(r_3\) in the tree are labelled. Labels of the input edges to the common operation node are \(E(r_1)\) and \(E(r_3)\). With regards to Table 3.4 and the labels, reductions for \(r_1\) and \(r_3\), in order, are \(r_1 \bowtie r_3\) and \(r_3 \bowtie r_1\).

![Figure 3.14: A labelled expression tree of relational expression \((r_1 \bowtie r_2) \cap ((r_3 \bowtie r_4) - r_5)\) for paths of \(r_1\) and \(r_3\).](image)

3.4 Reduction Operations in Relational Algebra

We will consider reduction operations in the HDMDBS that provides a relational view of its component databases.

Consider a query \(Q\) and its decomposition into subqueries \(r\), \(s\) and \(t\), such that \(Q = r - (s - t)\). The following cases may occur:

1. Assume that subquery \(r\) is computed first and \(v\) denotes the result of \((s - t)\). The result, obtained from the computation of \(r\), can be used to reduce \(v\) into \(\sigma_{\phi(r)}(v)\). Because, selection operation of relational algebra is distributive over the difference operation, both \(s\) and \(t\) can be reduced to \(\sigma_{\phi(r)}(s)\) and \(\sigma_{\phi(r)}(t)\).

2. Assume that subquery \(t\) is computed first. Then, its results can be used to reduce \(s\) into \(\sigma_{\text{not}\phi(t)}(s)\). However, the result of \(t\) has no impact on the reduction of \(r\) because \(t\) contains only some of the tuples that belong to the result of \(r - (s - t)\).

3. Assume that subquery \(s\) is computed first. Its result can be used to reduce \(t\) into \(\sigma_{\phi(s)}(t)\). However, the result of \(s\) has no impact on the reduction of \(r\).
In order to detect the impacts and how the result of one argument of a data integration expression can be used to reduce the size of another argument of the expression, the labelling technique is used and the reductions are detected.

When the data integration expression is an expression of relational algebra and the common operation for two arguments of the expression is join, intersection, difference or union, we have already presented the reductions between the arguments.

In this study, as the basis for detecting the impacts of arguments, we use the reduction expression tables and change the reduction expressions in order to find what reduction operations are to be performed in the component databases.

**Example 3.10** Consider expression \( Q = r - (s - t) \), discussed above. Assume that the result of \( r \) is obtained first. We want to know what impacts the result of \( r \) have on \( t \). By labelling an expression tree of \( Q \) for paths of \( r \) and \( t \), the reduction expression \( t \times r \) is obtained from the reduction expression table for the difference operation. Since \( r \) and \( t \) are union compatible, the semijoin is changed to an intersection operation. To use the result of \( r \) for reducing the size of \( t \) in the relating component database, the reduction expression is changed to \( \sigma_{\phi(r)}(t) \).

Therefore, we need to change the reduction expressions in the reduction expression tables to the reduction operations. After changing, the tables will be called reduction operation tables. Tables 3.5, 3.6, 3.7, 3.8, and 3.9 are the reduction operation tables for join, intersection, union and difference as the common operations between two arguments \( q_i \) and \( q_j \) of a data integration expression. Note that in all reduction operation tables, it is assumed that \( q_i \) is computed first, then its result is used for processing \( q_j \). Also, \( \phi(q_i) \) consists of the common attributes between \( q_i \) and \( q_j \).

Since difference is a non-commutative operation, there are two reduction operation tables for it. In an expression tree, when the common operation is difference, and the label of the input edge for \( q_j \) is on the right hand side of the operation, Table 3.8 is used. If the label is on the left hand side of the operation, Table 3.9 is applied.

## 3.5 Summary

Some applications of the labelling theory in query optimisation in database systems were considered. Labelling an expression tree is a general technique to explore the optimisation of query processing in database systems and particularly in HDMDBSs.
### 3.5. Summary

<table>
<thead>
<tr>
<th>⊗</th>
<th>( A_x(q_i) )</th>
<th>( A_y(q_j) + )</th>
<th>( N_y(q_j) + )</th>
<th>( E_y(q_j) )</th>
<th>( E_y(q_j) + )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_x(q_i) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi}(\tau_{\phi}(q_j))(q_j) )</td>
<td>( \sigma_{\phi}(\tau_{\phi}(q_j))(q_j) )</td>
<td>( \sigma_{\phi}(\tau_{\phi}(q_j))(q_j) )</td>
</tr>
<tr>
<td>( A_x(q_i) + )</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
<tr>
<td>( N_x(q_i) + )</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
<tr>
<td>( E_x(q_i) )</td>
<td>( \sigma_{\phi}(\tau_{\phi}(q_j))(q_j) )</td>
<td>( \sigma_{\phi}(\tau_{\phi}(q_j))(q_j) )</td>
<td>( \sigma_{\phi}(\tau_{\phi}(q_j))(q_j) )</td>
<td>( \sigma_{\phi}(\tau_{\phi}(q_j))(q_j) )</td>
<td>( \sigma_{\phi}(\tau_{\phi}(q_j))(q_j) )</td>
</tr>
<tr>
<td>( E_x(q_i) + )</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
</tbody>
</table>

Table 3.5: Reduction operations for join \((k = x \cap y)\).

<table>
<thead>
<tr>
<th>( \cap )</th>
<th>( A_x(q_i) )</th>
<th>( A_y(q_j) + )</th>
<th>( N_y(q_j) + )</th>
<th>( E_y(q_j) )</th>
<th>( E_y(q_j) + )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_x(q_i) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
</tr>
<tr>
<td>( A_x(q_i) + )</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
<tr>
<td>( N_x(q_i) + )</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
<tr>
<td>( E_x(q_i) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
</tr>
<tr>
<td>( E_x(q_i) + )</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
</tbody>
</table>

Table 3.6: Reduction operations for intersection \((k = x \cap y)\).

<table>
<thead>
<tr>
<th>( \cup )</th>
<th>( A_x(q_i) )</th>
<th>( A_y(q_j) + )</th>
<th>( N_y(q_j) + )</th>
<th>( E_y(q_j) )</th>
<th>( E_y(q_j) + )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_x(q_i) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
</tr>
<tr>
<td>( A_x(q_i) + )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
</tr>
<tr>
<td>( N_x(q_i) + )</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
<tr>
<td>( E_x(q_i) )</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
<tr>
<td>( E_x(q_i) + )</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
</tbody>
</table>

Table 3.7: Reduction operations for union \((k = x \cap y)\).

<table>
<thead>
<tr>
<th>( r- )</th>
<th>( A_x(q_i) )</th>
<th>( A_y(q_j) + )</th>
<th>( N_y(q_j) + )</th>
<th>( E_y(q_j) )</th>
<th>( E_y(q_j) + )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_x(q_i) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
</tr>
<tr>
<td>( A_x(q_i) + )</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
<tr>
<td>( N_x(q_i) + )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
</tr>
<tr>
<td>( E_x(q_i) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
<td>( \sigma_{\phi(q_i)}(q_j) )</td>
</tr>
<tr>
<td>( E_x(q_i) + )</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
</tbody>
</table>

Table 3.8: Reduction operations for difference operation when a reduced argument is on the right hand side of difference \((k = x \cap y)\).
3.5. Summary

<table>
<thead>
<tr>
<th>$l-$</th>
<th>$A_y(q_i)$</th>
<th>$A_y(q_j)$+</th>
<th>$N_y(q_j)$+</th>
<th>$E_y(q_j)$</th>
<th>$E_y(q_j)$+</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_x(q_i)$</td>
<td>$\sigma_{\text{not}}(q_i)(q_j)$</td>
<td>$\sigma_{\text{not}}(q_i)(q_j)$</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
<tr>
<td>$A_x(q_j)$+</td>
<td>$\sigma_{\text{not}}(q_j)(q_j)$</td>
<td>$\sigma_{\text{not}}(q_j)(q_j)$</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
<tr>
<td>$N_x(q_i)$+</td>
<td>$\sigma_{\text{not}}(q_i)(q_j)$</td>
<td>$\sigma_{\text{not}}(q_i)(q_j)$</td>
<td>$\sigma_{\text{not}}(q_i)(q_j)$</td>
<td>NIL</td>
<td>NIL</td>
</tr>
<tr>
<td>$E_x(q_i)$</td>
<td>$\sigma_{\text{not}}(r_x(q_i))(q_j)$</td>
<td>$\sigma_{\text{not}}(r_x(q_i))(q_j)$</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
<tr>
<td>$E_x(q_j)$+</td>
<td>$\sigma_{\text{not}}(r_x(q_j))(q_j)$</td>
<td>$\sigma_{\text{not}}(r_x(q_j))(q_j)$</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
</tr>
</tbody>
</table>

Table 3.9: Reduction operations for difference operation when a reduced argument is on the left hand side of difference ($k = x \cap y$).

In an expression, the impacts of arguments on each other are important for reducing their size before integration. In the expression, an argument may reduce the size of another argument in a reduction expression.

To detect reduction between two arguments of an expression, it is necessary:

- to identify the common operation for the arguments and
- to find the class of the operation whose output is input to the common operation for each one of the arguments in the corresponding expression tree of the expression.

After identifying the common operation for the two arguments, the labelling technique is used to label paths of the arguments in the corresponding expression tree to find the class of the operation whose output is input to the common operation node for each of the arguments.

For each operation in the database, a table is designated called the reduction expression table. The first row and the first column of the table contain all operation classes in the database. In the table, the cross of a row and a column contains the reduction expression(s) relating to the classes of the operation.

Having a common operation for two arguments and the classes of the operations whose outputs are inputs to the common operation for the arguments, reduction between the arguments can be obtained from the reduction expression table for the common operation.

The benefit of applying a reduction depends on how much it reduces the time for performing operations in the path of the argument to the common operation.

Reductions between arguments of an expression can be employed as a general technique for the optimisation of query processing. In the classical optimisation, beneficial reductions between arguments of an expression can be applied by replacing arguments
with the reduction expressions. In homogeneous distributed database systems, beneficial reductions can be employed in the local processing and in the final processing steps to reduce data transmission time between database sites and to reduce the final processing time. In HDMDBSs, the reductions between arguments of a data integration expression are applied to reduce the size of partial results of query processing in the component databases. Also, when partial results are integrated, the reductions can be used to reduce integration time.

After a query in HDMDBSs is decomposed into subqueries and then translated to the query language of component databases, three strategies can be applied for submission of the subqueries to the component databases as follows:

1. Sequential submission,

2. parallel submission and

3. hybrid (a mixture of sequential and parallel) submission.

If the submission of a subquery is delayed until the partial results of previously submitted subqueries are available, the strategy is called sequential submission. In this strategy, the partial results obtained from the processing of other subqueries are used in the processing of a subquery in the relating component database.

If all subqueries are submitted to the component databases at the same time, they are processed in the component databases in parallel and the results may be transmitted in parallel to the submission site. This strategy is called parallel submission. Hybrid submission of subqueries is when some of the subqueries are submitted in parallel and the others are submitted sequentially to the component databases.

In the sequential subquery submission strategy, reductions between arguments of the data integration expression are important factors in identifying which subquery should be submitted first and which should follow. In the parallel subquery submission strategy, the reductions can be applied to reduce the size of the available partial results obtained from the processing of subqueries in the component databases in order to reduce the integration time.

Reduction expressions and operations for arguments of relational expressions are explored when the common operation between the arguments is $\cap$, $\setminus$, $\cup$ or $\cap$. The reduction expressions and operations, in order, are obtained from the reduction expression and the reduction operation tables, by applying the labels of the input edges to the common operation for the arguments in the corresponding expression tree.
Chapter 4

On Optimal Query Processing Strategies

After decomposition of a query into a collection of subqueries and then their translation to the query language of component databases, the subqueries are submitted to the component databases for processing. The order in which the subqueries are submitted is important in the efficiency of query processing.

This chapter deals with the efficient processing of subqueries in the component databases with respect to the impacts of the subquery results on each other. The basis for identifying the impacts is the reductions between the arguments of the data integration expression. Having the reductions, query computation costs in the component databases, data transmission costs and the data conversion costs, an algorithm called HYBRID, is presented to determine the partial results of which subqueries must be obtained first and then which ones must use the results obtained for processing to minimise the response time.

4.1 Autonomy and Heterogeneity

Component database systems participating in the heterogeneous distributed multi-database system (HDMDBS) are autonomous. Some of the components may have complete database operations, while the others may lack even primitive ones. The number of participating component databases may be very large.

Preserving the autonomy of the component databases creates many problems for the processing and optimising of queries in the HDMDBS. Yet another problem is syntax and semantic heterogeneities that are to be resolved in the schema and data integration. In this study, we focus only on the preservation of the autonomy of the component database systems.

Preserving full autonomy of the component database systems is costly and causes restrictions in the query processing:
4.1. Autonomy and Heterogeneity

- Only queries can be sent to the component databases [MeLY93]. To send data to the components, each one of the components must have a utility for interpreting the data. Usually, there are no such utilities in the component databases. Above all, adding such utility to a component database system violates the autonomy.

- Although component database systems are fully connected through the communication networks, they cannot communicate with each other directly. Communication is through the HDMDBS by a message passing system.

- Because of autonomy, the statistical data of component databases, required for query optimisation, may not be available.

- A component database system may terminate its cooperation in the query processing at any time.

- Query processing in the components has no impacts on each other.

In homogeneous distributed database systems, semijoins are used to reduce data transmission time with the assumption that each local database system can accept data as well. There are two important issues in using semijoins in homogeneous distributed database systems:

1. All local database systems are capable of performing the same database operations and they are in one data model. Thus, when data are sent from one local database to another, there is no need to translate the data from one structure to another structure. Furthermore, because of the top-down design approach in homogeneous distributed database systems, there are usually no data inconsistencies.

2. Sending data from one local database system to another is a part of the responsibilities of the query processing subsystem.

In the HDMDBS, it is important that, for obtaining the result of each subquery, the relating component database is accessed once:

- For obtaining the result of a subquery, one access to the component database system creates less interference to the current work of the component.

- Several accesses to a component database for obtaining a particular data need several query executions in the component, several transmissions of the data and several conversions of the data.
4.2. Reductions

A component database may terminate its cooperation on query processing at any time. Thus, if, for obtaining the result of a subquery, the component database is to be accessed more than once, then it may happen that, at the first time, access to the component is successful, while in the later accesses, the component database terminates its cooperation.

Since component database systems are autonomous, they may not be willing to reveal statistical information such as the cardinalities and the selectivities. To obtain such statistical information or, in general, cost models of the components, one of the methods introduced in [DuKS92, ZhuL94a, HaSN96] can be used.

4.2 Reductions

A global query is decomposed into a collection of subqueries such that each needs to be sent to one component database. After translation to the query language of the component databases, the subqueries are sent to the components for execution.

If subqueries are submitted to the component database systems in parallel, then they are processed in parallel and the results may be transmitted to the submission site in parallel. Due to the parallel processing of subqueries and parallel transmission of data, this approach is efficient if the results of subqueries have no significant impacts on each other. Otherwise, the outcome may be the transmission of huge amount of data from each component database system to the submission site.

Traditional reductions, such as semijoins that are intended to reduce the size of transmitted data in homogeneous distributed database systems, are not applicable to the HDMDBS because each component database system accepts query. Another problem is that a global query may include not only join operation but other operations such as union, difference and intersection.

First, we will concentrate on how a reduction can be performed in the HDMDBS, then we will pay attention to the detection of reductions between component databases.

Suppose a global query is $r \bowtie s$, expressed in the global data model which is relational. Data sources of $r$ and $s$, in order, originate from component databases $d_1$ and $d_2$ that may not be relational. Let the set of common attributes between relational tables $r$ and $s$ be $x$. Decomposition of the global query results in two subqueries that after translation to the query language of component databases $d_1$ and $d_2$, in order, are $q_{d_1}(r)$ and $q_{d_2}(s)$.

While only queries can be sent to the component databases, we intend to reduce the
size of query processing result in a component database by using the result of the other
subquery. For the time being, we will focus on the submission order of subqueries, then
we will pay attention to how it is possible to reduce the size of the query processing
result in a component database by sending a query to the component.

Subqueries $q_{d_1}(r)$ and $q_{d_2}(s)$ can be sent to the component databases in one of the
following ways:

(i) Sending both subqueries to the components at the same time. Thus, they are
processed in parallel and their results may be shipped back in parallel.

(ii) Using the result of subquery $q_{d_1}(r)$ for processing subquery $q_{d_2}(s)$ in component
database $d_2$. Thus, subquery $q_{d_1}(r)$ is submitted first.

(iii) Using the result of subquery $q_{d_2}(s)$ for processing subquery $q_{d_1}(r)$ in $d_1$. Thus,
subquery $q_{d_2}(s)$ is to be submitted first.

As mentioned earlier, the first method is efficient as long as a subquery result does
not have an impact on reducing the size of another subquery result in the component
database.

In the second and the third methods, the result of one subquery is used to process
another subquery. In this way, the result of the second subquery is reduced, which
may result in less data transmission time, less data conversion time and less integration
time. Assume that subquery $q_{d_1}(r)$ is sent to the component database $d_1$ first. Since
relations $r$ and $s$ are to be joined, after the result of subquery $q_{d_1}(r)$ is available in the
submission site, the contents of $\pi_x(r)$ are to be transmitted to the component database
site $d_2$ for performing semijoin operation $s \bowtie \pi_x(r)$. While component databases accept
only queries, the following steps can be applied for performing the semijoin operation
in component database $d_2$:

1. Projecting the result of relation $r$ on $x$ attributes.

2. Building a propositional formula $\phi(\pi_x(r))$ for subquery $q_{d_2}(s)$ as a conjunction
of atomic terms $attribute=value$ for each tuple of $\pi_x(r)$. The entire formula is a
disjunction of conjunctions obtained from conversions of the individual tuples.

3. Modifying the subquery $q_{d_2}(s)$ to $q_{d_2}(\sigma_{\phi(\pi_x(r))}(s))$ and sending it to the component
database $d_2$ for execution.
Example 4.1  Let schemes of relations r and s, in order, be \((A, B, C)\) and \((D, E, F)\), where \(x = [(B, D), (C, E)]\) is the common attributes. Three tuples of relation r are given in Table 4.1.

The propositional formula, \(\phi(\pi_x(r))\), is constructed from the following disjunction of conjunctions. \(\{(D = b_1) \land (E = c_1)\} \lor \{(D = b_2) \land (E = c_2)\} \lor \{(D = b_3) \land (E = c_3)\} \lor \ldots\).

If the size of \(\pi_x(r)\) is too large for component database \(d_2\), it must be partitioned into \(\pi_x(r_1), \pi_x(r_2), \ldots, \pi_x(r_k)\), where \(r_1 \cup r_2 \cup \ldots \cup r_k = r\), then each one of the partitioned tables is translated into a propositional formula.

If relation \(r\) is significantly larger than \(s\) and the computation of \(q_{d_1}(\sigma_{\pi_x(s)}(r))\) strongly reduces the size of the result, then it is better to compute \(q_{d_2}(s)\) first. Therefore, the order in which the subqueries are processed has an important impact on the overall performance of the system.

In the example discussed above, parallel query processing strategy provides better results than sequential processing if more time is spent on processing \(q_{d_1}(r)\) and \(q_{d_2}(\sigma_{\pi_x(r)}(s))\) than on data transmission. In such a case, it is better to process \(q_{d_1}(r)\) and \(q_{d_2}(s)\), in order, in \(d_1\) and \(d_2\) in parallel.

Note that if, for obtaining the result of each subquery, more than one access to the component database were allowed, then it would also be possible to apply another method. This is, sending queries \(q_{d_1}(\pi_x(r))\) and \(q_{d_2}(\pi_x(s))\) for obtaining \(x\) attribute values from each component database, performing intersection operation \(\pi_x(r) \cap \pi_x(s)\) in the submission site, then sending queries \(q_{d_1}(\sigma_{\pi_x(r) \cap \pi_x(s)}(r))\) and \(q_{d_2}(\sigma_{\pi_x(r) \cap \pi_x(s)}(s))\) to obtain the results from the component databases.

In the above example, the impact of a subquery result on the result of another subquery can easily be detected. For a collection of subqueries \(\{q_1, \ldots, q_n\}\), the major problems are how to detect an impact that the result of a subquery \(q_i\) has on the
reduction of the results of other subqueries \( \{q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_n\} \) and then in what order the subqueries are to be submitted to the component databases.

For a global query, the impacts of subquery results on each other can be detected by using the reductions between arguments of the data integration expression. The reductions are discovered by labelling an expression tree corresponding to the data integration expression as discussed in chapter 2 and 3. Having the reductions and a cost function for evaluation of subqueries, an algorithm determines in what order (i.e. sequential, parallel, or mixture of sequential and parallel) the subqueries should be submitted to the component databases in order to have a minimum response time.

### 4.3 Minimising Response Time

The objective of this section is to provide an algorithm to minimise the response time of global query processing. Due to autonomy and parallel processing of subqueries in component database systems, we will concentrate on the minimisation of the response time rather than the total time.

Given a global query \( Q \), its decomposition into subqueries \( \{q_1, \ldots, q_n\} \), and a data integration expression \( e(q_1, \ldots, q_n) \), and given the reductions between arguments of the data integration expression, then, after translation of the subqueries to the query language of the component database systems, with the assumption that each component database system is to be accessed once, the problem is that in what order the subqueries should be submitted to the component databases to minimise the response time.

#### 4.3.1 Query Evaluation Parameters

Assume that the partial result obtained from processing of a subquery \( q_i \) in the relating component database system is argument \( r_i, (1 \leq i \leq n) \). If the global data model is relational modelling of data, then argument \( r_i \) is a relational table.

To obtain \( r_i \) in the submission site, subquery \( q_i \) is sent to the relating component database system. The time that is needed to send subquery \( q_i \) for obtaining \( r_i \) in the submission site and integrating it with other partial results consists of the following costs:

1. transmission time of subquery \( q_i \) from submission site to the component database system site,
2. query processing time in the component database system,

3. data transmission time for transmitting the query processing result, $r_i$, from the component database site to the submission site,

4. data conversion time for converting $r_i$ to the global data model format in the submission site and

5. data integration time to integrate $r_i$ with other obtained partial results.

Generally, transmitting a subquery to a component database system has negligible cost because the size of a subquery is usually very small. If the size is large, then transmission of the subquery takes time and must be taken into account. Assume $|q_i|$ is the size of subquery $q_i$ for obtaining $r_i$. Furthermore, it is taken into account whenever the size is large.

Let $E_i'$ be the time for processing subquery $q_i$ in the component database site. The time for transmission of $r_i$ (which is the result of processing subquery $q_i$) to the submission site is $F_i' + D_i \cdot |r_i|$, where $F_i'$ and $D_i$, in order, are the constants relating to the setup time for establishing the connection and the transmission speed, and $|r_i|$ is the size of $r_i$. The time for conversion of the result is also proportional to its size. Let $C_i \cdot |r_i|$ be the conversion time at the submission site for converting $r_i$ to the global data model format, where $C_i$ is a constant of the conversion.

Final integration is performed when all of the partial results are available in the submission site. After conversion to the global data model format, the partial results are integrated. Let the total integration time for integrating all partial results be $I$. If we assume that the integration has the worst time of complexity, then the integration time is proportional to the size of arguments. Thus, integration time, $I$, is $I'' \cdot \Pi_{i=1}^n |r_i|$, where $I''$ is an integration constant and $n$ is the number of partial results.

Accordingly, the time that is required to obtain partial result $r_i$ in the submission site and to integrate it with other partial results, called $cost(r_i)$, estimated from the following formula:

$$cost(r_i) = E_i' + F_i' + D_i \cdot |r_i| + C_i \cdot |r_i| + I' \cdot |r_i|$$

Note that $I'$ is different from $I''$. The variable $I'$ is computed from:

$$I' = \frac{I}{\sum_{i=1}^{n} |r_i|}$$

After simplifying:
4.3. Minimising Response Time

\[\text{cost}(r_i) = E_i' + F_i' + (D_i + C_i + I') \cdot |r_i|\]

The subquery transmission time \(D_i \cdot |q_i|\) can be added to the formula when \(|q_i|\) is large and cannot be neglected. In such a case, with the assumption that the setup time is only necessary once, only one \(F_i'\) is needed.

If there is a reduction between two arguments \(r_i\) and \(r_j\) such that argument \(r_i\) is reduced by \(r_j\), it is represented by reduction expression \(r_i \alpha r_j\). The result of \(\alpha\) operation is a reduction in the size of \(r_i\). The operation \(\alpha\) is the reduction operation and \(r_j\) is a reducer for \(r_i\).

Let \(\rho_j\) represents a selectivity associated with the reduction operation in \(r_i \alpha r_j\). The selectivity \(\rho_j\) is a rational number that ranges from 0 to 1. After execution of \(r_i \alpha r_j\), the size of \(r_i\) becomes \(\rho_j \cdot |r_i|\), where \(|r_i|\) is the size of argument \(r_i\).

To simplify the problem, it is assumed that the effects of reduction operations are independent. For example, after the execution of reduction operations for argument \(r_i\) by a set of reducers in \(R\), \(\{r_i \alpha r_j \mid j \in R\}\), the size of \(r_i\) becomes \(\Pi_{j \in R} \rho_j \cdot |r_i|\). Note that the set \(R\) contains indexes of the reducers.

Reducing the size of an argument results a reduction in the transmission time, a reduction in the conversion time and a reduction in the integration time. A reduction operation is \textit{profitable} if the reductions are greater than all the work that is done for reducing the size of that argument.

For performing reduction operation \(\alpha\) in \(r_i \alpha r_j\), first a query is sent to the site where \(r_j\) is located to obtain it in the submission site. After \(r_j\) is transmitted and then converted, it is scanned for the projection of the relating data required to reduce the size of \(r_i\). Then, the data are converted to a propositional formula and the subquery \(q_i\) with the propositional formula is sent to the site where \(r_j\) is located to reduced its size.

Let the time for all of such work for reducing the size of \(r_i\) by \(r_j\) be \(w^j_i\). Note that the extra time, required for reducing the size of \(r_i\) in its site, is also included in \(w^j_i\). Also, \(w^j_i\) excludes the time that is needed for the final integration of \(r_j\).

A reduction operation is profitable if:

\[\text{cost}(r_i) > w^j_i + E_i' + F_i' + D_i \cdot |r_i'| + C_i \cdot |r_i'| + I' \cdot |r_i'|,\]

where \(r_i'\) is \(r_i\) after it is reduced by \(r_j\), i.e. \(|r_i'| = \rho_j \cdot |r_i|\).

The amount of profit for \(r_i\) whose size can be reduced by \(r_j\) is indicated by \(P_i\). It can be expressed as follows:
4.3. Minimising Response Time

\[ P_i = \text{cost}(r_i) - (w_i' + E_i' + F_i' + D_i \cdot |r_i'| + C_i \cdot |r_i'| + I' \cdot |r_i'|) \]

For each \( r_i \), there is a set of reducers by \( r_j \), where \( 1 \leq j \leq n, j \neq i \). Assume the set of indexes of such reducers for argument \( r_i \) is \( U_i \). Note that the set of reducers for argument \( r_i \) is identified by the reductions between \( r_i \) and the other arguments.

Our method chooses a subset \( B_i \) of \( U_i \) to reduce the size of \( r_i \). Our task is to find \( \{B_1, B_2, \ldots, B_n\} \) and an execution sequence order for \( r_{i(1 \leq i \leq n)} \) in order to minimise the response time, with the restriction that for obtaining each argument, the relating component database can be only accessed once.

Consider reductions for argument \( r_i \). The amount of work for reducing the size of \( r_i \) by \( r_j, j \in B_i \) is \( w_j \). The size of reducers in \( B_i \) may also be reduced by \( r_h, \) where \( h \in B_j \) and \( h \neq i \). Also, The size of reducers in \( B_j \) may be reduced by \( r_g, \) where \( g \in B_h \) and \( g \neq (i, j) \) and so on. We show such a sequence of reducers (i.e. reducers in reduction expressions \( r_i \, \alpha_1 \, r_j, r_j \, \alpha_2 \, r_h, \ldots, r_j \, \alpha_3 \, r_g, \ldots \)) that are also profitable, \( j_s \) for \( j \) in \( B_i \).

Reducing \( r_i \) can be done for any \( m \) in \( B_i \) in parallel as long as \( w_{m_s}^i \leq w_{j_s}^i \) to further reduce its size. Thus all such work for reducing the size of \( r_i \) is at most \( \max_{j_s \in B_i} (w_{j_s}^i) \).

For a set of reducers \( j_s \in B_i \), the profit is represented by the following formula:

\[ P_i = \text{cost}(r_i) - (\max(w_{j_s}^i))_{j_s \in B_i} + E_i' + F_i' + (D_i + C_i + I') \cdot |r_i'| \]

In the formula, \( |r_i'| = \Pi_{j_s \in B_i}(\rho_{j_s}^i) \cdot |r_i| \). By substituting \( \text{cost}(r_i) \) in the formula, we have:

\[ P_i = E_i' + F_i' + (D_i + C_i + I') \cdot |r_i| - (\max(w_{j_s}^i))_{j_s \in B_i} + E_i' + F_i' + (D_i + C_i + I') \cdot \Pi_{j_s \in B_i}(\rho_{j_s}^i) \cdot |r_i| \]

After simplifying:

\[ P_i = |r_i| \cdot (D_i + C_i + I') \cdot (1 - \Pi_{j_s \in B_i}(\rho_{j_s}^i)) - \max(w_{j_s}^i)_{j_s \in B_i} \]

**Computation of \( w_{j_s}^i \)'s**

When the size of an argument such as \( r_i \) is reduced by \( r_j \) and argument \( r_j \) has no reducer, the amount of work for reducing the size of \( r_i \) is \( w_j^i \). If argument \( r_j \) is also reduced by another reducer, then the amount of work for reducing the size of \( r_i \) will not be \( w_j^i \).

As an example, let the sequence set of reducers \( j_s \) in \( U_i \) for \( r_i \) be \( \{j, h, g\} \). Suppose the reduction expressions are \( r_i \, \alpha_1 \, r_j, r_j \, \alpha_2 \, r_h, \) and \( r_h \, \alpha_3 \, r_g \). Assume the amount of work and the selectivity for each one of the reductions, in order, are \( (w_j^i, \rho_j^i), (w_h^i, \rho_h^i) \)
4.3. Minimising Response Time

and \((w_g^h, \rho_g^h)\). The sequence of operations, in order, are \(\alpha_3, \alpha_2\) and then \(\alpha_1\). In other words, first \(r_h\) is reduced by \(r_g\), then \(r_h\) reduces \(r_j\) and finally, \(r_i\) is reduced by \(r_j\).

The amount of work to perform \(r_h \alpha_3 r_g\) is \(w_g^h\). Since \(r_h\) is reduced, the amount of work to perform \(r_j \alpha_2 r_h\) will not be \(w_j^i\) any more. It will be proportional to the size of \(r_h\). Since the size of \(r_h\) is reduced by the factor of \(\rho_g^h\), the amount of work to perform \(r_j \alpha_2 r_h\), will be \(|w_h^j| \cdot \rho_g^h\). Likewise, the amount of work to perform \(r_i \alpha_1 r_j\) will be \(|w_j^i| \cdot \rho_g^h \cdot \rho_j^i\) (see Figure 4.1).

\[
\begin{align*}
  r_g & \xrightarrow{w_g^h} r_h & \xrightarrow{w_h^j \cdot \rho_g^h} r_j & \xrightarrow{w_j^i \cdot \rho_g^h \cdot \rho_j^i} r_i 
\end{align*}
\]

Figure 4.1: The amount of work in each next step is reduced.

Note that the total amount of work for reducing \(r_i\) by the sequence set \(j_s\) is the summation of \(w_g^h, |w_h^j| \cdot \rho_g^h\) and \(|w_j^i| \cdot \rho_g^h \cdot \rho_j^i\). Thus, \(w_{js} = w_g^h + |w_h^j| \cdot \rho_g^h + |w_j^i| \cdot \rho_g^h \cdot \rho_j^i\).

If an argument is reduced by other arguments in parallel, then the total amount of work is computed by taking into account the parallel execution. For instance, suppose reduction expressions are \(r_j \alpha_3 r_g, r_j \alpha_2 r_h\) and \(r_i \alpha_1 r_j\), in order, with the amount of work and the selectivities \((w_g^j, \rho_g^j), (w_h^j, \rho_h^j)\) and \((w_j^j, \rho_j^j)\). Assume the sequence of operations are \(\alpha_3, \alpha_2\) and then \(\alpha_1\). Reduction operations \(\alpha_3\) and \(\alpha_2\) are performed in parallel. The total amount of work to reduce \(r_i\) is the summation of \(\max(|w_g^j|, |w_h^j|)\) and \(|w_j^j| \cdot \rho_g^j \cdot \rho_h^j\). Thus, \(w_{js} = \max(|w_g^j|, |w_h^j|) + |w_j^j| \cdot \rho_g^j \cdot \rho_h^j\).

4.3.2 Formulation

The size of \(r_i\) after reduction operations becomes \(\Pi_{j_s \in B_i} (\rho_{js}^j) \cdot |r_i|\). The transmission and conversion times for sending the reduced \(r_i\) to the submission site is \(F_i + (D_i + C_i) \cdot \Pi_{j_s \in B_i} (\rho_{js}^j) \cdot |r_i|\). The total time to reduce \(r_i\), to transmit it to the submission site and to convert it in the submission site is:

\[
\max_{j_s \in B_i} (w_{js}^i) + E_i' + F_i' + (D_i + C_i) \cdot \Pi_{j_s \in B_i} (\rho_{js}^j) \cdot |r_i| 
\]

Integration begins when all \(r_i\)'s become available, i.e. they are transmitted and then they are converted. Since conversion is performed in the submission site sequentially, it is necessary to compute the total time for the conversion. If there are no reducers for the arguments and before conversion, the arguments arrive at the same time in the submission site, then the total conversion time is: \(\sum_{i=1}^{n} C_i \cdot |r_i|\). The arguments may have reducers and also they may not arrive in the submission site at the same time.
4.3. Minimising Response Time

It is assumed that the amount of work (i.e. $w_{js}$'s) includes the conversion times for reducers in $B_i$'s. Thus, conversion times for the reducer arguments are not recomputed. In general, the arguments may not arrive at the same time and some of them may be reducers.

When the last argument arrives in the submission site, its conversion may be started. Conversion of the last argument may be postponed because conversion of previously arrived arguments may not yet have been completed by the arrival time of the last argument. Let the time that the last argument must wait for starting the conversion be $W_f$.

To compute $W_f$, we need the arrival times of the arguments that are not reducers. Let $A_i$ be the arrival time of argument $r_i$, computed as follows:

$$A_i = w_{js} + E_i' + F_i' + D_i \cdot \Pi_{j \in B_i} (\rho_{js}) \cdot |r_i|$$

Note that conversion time for the argument is $C_i \cdot \Pi_{j \in B_i} (\rho_{js}) \cdot |r_i|$ and it is not included in the formula. If the argument has no reducers, then $w_{js} = 0$ and $\rho_{js} = 1$.

Let sorting of $A_i$'s in an increasing order be $\{A_{i_1}, A_{i_2}, \ldots, A_{i_m}\}$ for $(1 \leq m \leq n)$. In other words, among arguments that are not reducers, argument $r_{i_1}$ is the first argument and argument $r_{i_m}$ is the last one that arrive in the submission site.

When argument $r_{i_1}$ arrives, its conversion begins immediately. The next argument which is $r_{i_2}$ may either arrive in parallel with argument $r_{i_1}$, it may arrive during the conversion or it may arrive after the conversion is completed. In general, conversion of an argument $r_{i_k}$ with $k > 1$ may be delayed because the conversions of previously arrived arguments may not have been completed at the arrival time of the argument.

Assume conversion of argument $r_{i_k}$ can begin at the arrival. Let $T_{i_k}$ be a number that specifies, until arrival of the next argument, whether the conversion of argument $r_{i_k}$ can be carried out or not. It is computed as follows:

$$T_{i_k} = (A_{i_{k+1}} - A_{i_k}) - C_{i_k} \cdot \Pi_{j \in B_{i_k}} (\rho_{js}^{i_k}) \cdot |r_{i_k}|$$

The value of $T_{i_k}$ specifies whether the conversion ends on arrival of the next argument. If the next argument arrives during conversion of argument $r_{i_k}$, then $T_{i_k}$ is negative. When the next argument arrives either at the time that the conversion is ended, or it arrives after the conversion, the value of $T_{i_k}$ is either zero or positive, respectively. If the value of $T_{i_k}$ is zero or positive, conversion of the next argument can be started when it arrives. If the value is negative, conversion of the next argument on the arrival is delayed after the time of $|T_{i_k}|$. 
Accordingly, the waiting time, $W_f$, that the last argument $r_{im}$ must wait for starting the conversion is computed from: $W_f = |\text{Wait}(T_{i_{m-1}})|$, where $\text{Wait}(T_{i_0}) = 0$, and the function $\text{Wait}$ is a recursive function, computed as follows:

$$\text{Wait}(T_{i_k}) = \min(\text{Wait}(T_{i_{k-1}}) + T_{i_k}, 0)$$

(4.1)

Integration begins when all arguments are transmitted and converted. It needs the time of:

$$\max_{1 \leq i \leq n}(\max_{j \in B_i}(w_{j,i}^i) + E_i + F_i^i + D_i \cdot \Pi_{j \in B_i}(\rho_{j,i}^i) \cdot |r_i| + W_f + C_i \cdot \Pi_{j \in B_i}(\rho_{j,i}^i) \cdot |r_i|)$$

The total time for integration is $I'' \cdot \Pi_{i=1}^n(\Pi_{j \in B_i}(\rho_{j,i}^i) \cdot |r_i|)$. The overall response time is expressed as a function of $B_1, B_2, \ldots, B_n$ as follows:

$$\text{RES}(B_1, B_2, \ldots, B_n) = \max_{1 \leq i \leq n}(\max_{j \in B_i}(w_{j,i}^i) + E_i + F_i^i + D_i \cdot \Pi_{j \in B_i}(\rho_{j,i}^i) \cdot |r_i| + W_f + C_i \cdot \Pi_{j \in B_i}(\rho_{j,i}^i) \cdot |r_i| + I'' \cdot \Pi_{i=1}^n(\Pi_{j \in B_i}(\rho_{j,i}^i) \cdot |r_i|)$$

To simplify the expression, let $E_i = E_i' + F_i'$, $I = I'' \cdot \Pi_{i=1}^n|r_i|$, and $B = B_1, B_2, \ldots, B_n$. Minimising the response time can be stated as follows:

Given $A = (I, (E_i, D_i, C_i, |r_i|, \{(w_{j,i}^i, \rho_{j,i}^i) \mid j \in U_i\}), (E_2, D_2, C_1, |r_2|, \{(w_{j,2}^2, \rho_{j,2}^2) \mid j \in U_2\}), \ldots, (E_n, D_n, C_1, |r_n|, \{(w_{j,n}^n, \rho_{j,n}^n) \mid j \in U_n\})), \text{such that } I, E_i, D_i, C_i, w_j^i$ are positive numbers, $|r_i|$ is the size of $r_i$, $0 \leq \rho_{j,i}^i \leq 1$ and $W_f$ is a waiting time to start conversion of the last argument, find an optimal $B = (B_1, B_2, \ldots, B_n)$, and an execution sequence order for $r_i (1 \leq i \leq n)$ such that:

$$\text{RES}(B) = \max_{1 \leq i \leq n}(\max_{j \in B_i}(w_{j,i}^i) + E_i + D_i \cdot \Pi_{j \in B_i}(\rho_{j,i}^i) \cdot |r_i| + W_f + C_i \cdot \Pi_{j \in B_i}(\rho_{j,i}^i) \cdot |r_i|)$$

is minimised.

### 4.3.3 Restriction on Using Reduction Operations

Only one access to each component database system is allowed for obtaining the result of a subquery. Thus, if there are reductions such as $r_i \alpha r_j$ and $r_j \beta r_i$, only one of them can be applied. Although the component databases can be accessed in parallel, the restriction of one access to each of them prohibits the parallel processing for reducing the size of $r_i$ and $r_j$ in the component databases, applying both reduction operations $\alpha$ and $\beta$. Furthermore, for executing reduction operation $\alpha$, first $r_j$ is obtained in the submission site, then the required data of $r_j$ to reduce the size of $r_i$ is used as
4.3. Minimising Response Time

If reduction $r_j \beta r_i$ is applied, the same strategy is used. In this case, the reduction $r_i \alpha r_j$ cannot be applied. If there are no benefits in performing the reduction operations, then to obtain $r_i$ and $r_j$ in the submission site, two queries are sent in parallel to the sites where $r_i$ and $r_j$ are located to obtain them.

Therefore, the restriction of one access to each component database makes one of the reduction operations impractical. The problem of finding out which one of them should be used is very important in reducing the response time.

### 4.3.4 Profitable Reducers

For each argument, there is a set of reducers, identified by the reductions between the argument and other arguments of the data integration expression. A reducer may be profitable for an argument. If the reducer is not profitable, then it may be profitable in a sequence of reductions for the argument. Accordingly, profitable reducers for an argument are divided into direct and indirect profitable reducers. A direct profitable reducer is directly applied to reduce the size of an argument and a profit is obtained. For an indirect profitable reducer, a profit is obtainable in a sequence of reductions.

For example, assume that $r_j$, $r_g$ and $r_h$ are reducers for $r_i$. If $r_j$ reduces the size of $r_i$ by the reduction operation in $r_i \alpha r_j$ and is profitable, then $r_j$ is a direct profitable reducer for $r_i$. Suppose each one of $r_g$ and $r_h$ as reducers of $r_i$ is not a direct profitable reducer, i.e. if each one of $r_g$ and $r_h$ reduces the size of $r_i$, no profit is obtained. If by performing reduction operations in $r_h \alpha r_g$ and then in $r_h \alpha r_i$, a profit for reducing the size of $r_i$ can be obtained, then both $r_h$ and $r_g$ are indirect profitable reducers for $r_i$. Note that if the size of $r_h$ were not reduced by $r_g$, then $r_h$ could not be a profitable reducer for $r_i$.

### 4.3.5 The Search Space

Subqueries can be submitted to component databases either sequentially, in parallel, or in hybrid (i.e. a mixture of sequential and parallel). The search space for sequential execution of subqueries $\{q_1, \ldots, q_n\}$ by itself is $n!$. If the parallel and the hybrid executions of subqueries are also added to the sequential ones, the search space becomes gigantic.

In the worst case that an argument has $n - 1$ reducers, the number of subsets of
4.3. Minimising Response Time

reducers for the argument amounts to $2^{n-1}-1$. Furthermore, the subsets with more than one reducer are to be considered for the sequential and parallel arrangements to find out the most profit for the argument. The problem is NP-hard.

The search space can largely be reduced if only the direct profitable reducers for each argument are taken into account. To find whether a reducer may be a direct profitable reducer or not, we define $w_i^j$ as the weight of an argument $r_i$ for $(1 \leq i \leq n)$ as follows:

$$w_i^j = E'_i + F'_i + (D_i + C_i) \cdot |r_i|$$

The weight for $r_i$ consists of the costs for just obtaining and converting $r_i$ in the submission site. Assume that there is reduction $r_i \prec r_j$ that reduces the size of $r_i$ by $r_j$. It is evident that if $w_j^i > w_i^j$, then the reduction $r_i \prec r_j$ is not profitable. Furthermore, $r_j$ cannot be a direct profitable reducer for $r_i$. If $w_j^i < w_i^j$, the reduction operation may be profitable. In other words, from the set of reducers for each argument, the reducers whose weights are larger than the weight of the argument can be deleted because they cannot be direct profitable reducers for that argument. This may create a problem. A direct reducer that is not profitable for an argument may be an indirect profitable reducer for the argument.

To remedy the problem, it can be assumed that all reducers of arguments are also profitable if there exists at least one argument among other arguments that has a profit for the set of its reducers. Otherwise, reducers of arguments are pruned by comparing the weight of each argument with the weights of its reducers.

4.3.6 HYBRID Algorithm

The greater the profits are the lower the response time will be. Thus, instead of concentrating on reducing the response time, we focus our attention on obtaining the greatest profit in general.

In each step, among arguments of the data integration expression, an argument is chosen to be reduced by the set of its reducers which has the most profit. Then, the global effect of that argument by removing it from the reducers of other arguments are examined for finding out whether the chosen argument is the right choice.

Let $P = \sum_{i=1}^{n} P_i$ be a total profit for all arguments to be reduced by their reducers. Since some of the reductions are not practical (see Subsection 4.3.3), $P$ is not the actual overall profit that will be obtained.

If $r_i$ is the most profitable argument among the others to be reduced, then $P_i > P_{k(1 \leq k \leq n, k \neq i)}$. Assume $r_i$ is chosen because it is the most profitable one to be reduced
by the set of reducers in $U_i$. Since $r_i$ is the most profitable argument, it may not be a profitable reducer for other arguments any more. Thus, if $r_i$ is chosen, then it is removed from the set of reducers in $U_{k(1 \leq k \leq n, k \neq i)}$.

Let $P' = \sum_{i=1}^{n} P_i$ be the total profit when $r_i$ is removed from the reducers in $U_{k(1 \leq k \leq n, k \neq i)}$. If $P - P' < P_i$, then $r_i$ is the right choice because by choosing $r_i$ the most profit is obtained. Furthermore, the effect of $r_i$ as a reducer for obtaining profits, when it is removed from the reducers in $U_{k(1 \leq k \leq n, k \neq i)}$ is not more than the obtained profit. If $P - P' > P_i$, then the effect of $r_i$ as a reducer, when it is not removed from $U_{k(1 \leq k \leq n, k \neq i)}$, is that more profits can be obtained.

If $P - P' = P_i$, then $r_i$ as a reducer may reduce the sizes of some arguments in parallel and thus $r_i$ should not be chosen to be reduced. Therefore, if $P - P' \geq P_i$, then $r_i$ is not chosen to be reduced at this stage.

When $P - P' \geq P_i$, the next profitable argument in the sorted list of profits $P_{k(1 \leq k \leq n)}$ is chosen as a candidate to be reduced by its reducers. The new choice is again examined to see if its profit is greater than $P - P'$.

Note that when $r_i$ is chosen to be reduced, the profitable reducers in $U_i$ are stored in $B_i$, then $r_i$ and $U_i$ are not considered any more. Also, the index $i$ is removed from reducers in $U_{k(1 \leq k \leq n)}$. When a reducer is removed from any reducer set such as $U_k$ and the reducer is an element of the profits, then the profit $P_k$ is updated.

The first argument that is chosen to be reduced is given an execution order of $n$, which means that argument among $n$ arguments is the last one that will be obtained. The next selected argument is given an execution order of $n - 1$ and so on.

The algorithm that is presented below is called HYBRID because it tries to arrange submission of subqueries for obtaining arguments from the component databases for both sequential and parallel executions. If there is no benefit in the sequential execution of the subqueries at all, then the parallel execution is identified by the algorithm.

**HYBRID Algorithm**

**INPUT:** $n, I, \{(E_i, D_i, C_i), (w_j^i, \rho_j^i) \mid 1 \leq i \leq n, j \in U_i\}$

**OUTPUT:** $\{B_1, B_2, \ldots, B_n\}$, and a sequence of execution $S_1, S_2, \ldots, S_n$

$B_i = \emptyset$, $S_i = 0$, $(1 \leq i \leq n)$;

$m = n$;

while ($m > 0$)

begin

compute all $P_i$, $(1 \leq h \leq m)$;
4.3. Minimising Response Time

\[ \text{if } (\text{all}(P_{i,h}'s) < 0) \]
\[ \begin{align*}
&\text{begin} \\
&\quad \text{remove reducers for arguments that the weights < the reducers' weight;} \\
&\quad \text{recompute all } P_{i,h}'s; \\
&\quad \text{if } (\text{all}(P_{i,h}'s) < 0) \text{ return;} \\
&\text{end;} \\
&\text{for}(i = 1 \text{ to } m) \\
&\quad \text{begin} \\
&\quad \text{sort } P_i \text{ as } P_{i_1} \geq P_{i_2} \geq \ldots \geq P_{i_m} ; \\
&\quad P = \sum_{h=1}^{m} P_{i_h} ; \\
&\quad \text{for } (k = i_1 \text{ to } i_m) \\
&\quad \quad \text{begin} \\
&\quad \quad P' = \sum_{h=1}^{m} P_{i_h} , \{U_{i_h}\} - d, d = g(k); \\
&\quad \quad \text{if } (P_k > P - P') \\
&\quad \quad \quad \text{begin} \\
&\quad \quad \quad \quad d = g(k); \\
&\quad \quad \quad \quad B_d = \{U_d\}; \\
&\quad \quad \quad \quad \text{for } (h = 1 \text{ to } m) \\
&\quad \quad \quad \quad \quad \{U_{i_h}\} - d; \\
&\quad \quad \quad \quad S_m = d; \\
&\quad \quad \quad \quad m = m - 1; \\
&\quad \quad \quad \quad \text{break;} \\
&\quad \quad \quad \text{end;} \\
&\quad \quad \text{end;} \\
&\quad \text{end;} \\
&\text{end;} \\
\end{align*} \]

In sorting \( P_i \)'s we need to know which one of \( P_{i_1}, P_{i_2}, \ldots, P_{i_n} \) after sorting belongs to which argument. The function \( g() \) returns the index. When it is necessary to remove an element \( k \) from a reducer set \( U_i \), the notation \( \{U_i\} - k \) is used. The sequence of \( S_1, S_2, \ldots, S_n \) determines the execution order for arguments. Indexes of some arguments may not be assigned to sequences \( S_{i,1 \leq i \leq n} \); that means those arguments have no reducers. In such cases that the arguments have no reducers, they are submitted in parallel for execution.
4.3. Minimising Response Time

<table>
<thead>
<tr>
<th>i</th>
<th>$E_i$</th>
<th>$D_i$</th>
<th>$C_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2</td>
<td>0.003</td>
<td>0.0004</td>
</tr>
<tr>
<td>2</td>
<td>0.4</td>
<td>0.004</td>
<td>0.0008</td>
</tr>
<tr>
<td>3</td>
<td>0.6</td>
<td>0.009</td>
<td>0.0007</td>
</tr>
<tr>
<td>4</td>
<td>0.7</td>
<td>0.005</td>
<td>0.0006</td>
</tr>
</tbody>
</table>

Table 4.2: Values for parameters $E_i$ and $D_i$.

<table>
<thead>
<tr>
<th>i/j</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.3</td>
<td>1.6, 0.8</td>
<td>5.9, 0.7</td>
<td>2.6, 0.3</td>
</tr>
<tr>
<td>2</td>
<td>3.6, 0.5</td>
<td>1.4</td>
<td>6.4, 0.3</td>
<td>2.9, 0.4</td>
</tr>
<tr>
<td>3</td>
<td>4.2, 0.4</td>
<td>2.1, 0.6</td>
<td>5.5</td>
<td>3.4, 0.4</td>
</tr>
<tr>
<td>4</td>
<td>3.8, 0.6</td>
<td>1.8, 0.4</td>
<td>5.3, 0.5</td>
<td>2.4</td>
</tr>
</tbody>
</table>

Table 4.3: Values for $w^i_j$ and $p^i_j$.

4.3.7 Example

Suppose the global data model is relational and four relations $r_1$, $r_2$, $r_3$, $r_4$ are the arguments of the data integration expression. The sizes of the arguments are as follows: $|r_1| = 900$, $|r_2| = 200$, $|r_3| = 500$, $|r_4| = 300$. The total integration time, $I$, is 2.7 and other parameters are given in tables 4.2 and 4.3. Reducers for $r_1$ are $(r_2, r_3, r_4)$, for $r_2$ the reducers are $(r_1, r_3, r_4)$, for $r_3$ the reducers are $(r_1, r_2, r_4)$, and for $r_4$ the reducers are $(r_1, r_2, r_3)$. In other words, $U_1 = \{2, 3, 4\}$, $U_2 = \{1, 3, 4\}$, $U_3 = \{1, 2, 4\}$ and $U_4 = \{1, 2, 3\}$. The weights of the arguments are computed as follows:

$w^1_1 = E_1 + (D_1 + C_1) \cdot |r_1|$

$w^1_2 = 0.2 + (0.003 + 0.0004) \cdot 900 = 3.3$

$w^2_2 = 0.4 + (0.004 + 0.0008) \cdot 200 = 1.4$

$w^3_3 = 0.6 + (0.009 + 0.0007) \cdot 500 = 5.5$

$w^4_4 = 0.7 + (0.005 + 0.0006) \cdot 300 = 2.4$

With the assumption that the reducers for each argument are profitable, first of all, the profit $P_i$ for each argument is computed.

$P_i = |r_i| \cdot (D_i + C_i + I') \cdot (1 - \Pi_{j \in B_i} (p^j_{i,j})) - \max(w^i_j)_{j \in B_i}$

Since the integration time is 2.7, $I'$ is computed from the following formula:

$I' = \frac{\sum_{i=1}^{4} |r_i|}{1900}$

It is assumed that reducers of each arguments are profitable and they are not reduced by other arguments. Thus, $j = j_3$ and $B_i = U_i$. 
4.3. Minimising Response Time

\[ P_1 = |r_1| \cdot (D_1 + C_1 + I') \cdot (1 - \Pi_{j \in U_1}(\rho_j^1)) - \max(w_j^1)_{j \in U_1} \]

\[ U_1 = \{2, 3, 4\} \]

\[ P_1 = 900 \cdot (0.003 + 0.0004 + \frac{2.7}{1900}) \cdot (1 - \Pi_{(k=2,3,4)}(\rho_k^1)) - \max(w_{(2,3,4)}^1) \]

\[ P_1 = 4.34 \cdot (1 - 0.8 \cdot 0.7 \cdot 0.3) - \max(1.6, 5.9, 2.6) = 4.34 \cdot 0.832 - 5.9 = -2.29 \]

Similar to \( P_1 \), the profits \( P_2, P_3, P_4 \) can be computed. To distinguish computation of \( P_i \)'s from one step to another, \( P_i \)'s are represented by a superscript number that is increased at the next step. For example, \( P_1 \) at the first step is represented by \( P_1^0 \).

\[
\begin{align*}
P_1^0 &= -2.29 \\
P_2^0 &= -5.23 \\
P_3^0 &= +1.1 \\
P_4^0 &= -4.45
\end{align*}
\]

Since there is a \( P_i > 0 \) (i.e. \( P_3^0 \)), there is no need to compare the weights for removing some reducers. The sorting of \( P_i \)'s in a decreasing order is:

\[ P_3^0 > P_4^0 > P_1^0 > P_2^0 \]

Since \( P_3^0 \) is the greatest profit of all, the argument \( r_3 \) can be reduced by the set of its reducers with the most profit. Thus the argument \( r_3 \) is selected.

The total profit is:

\[ P = \sum_{i=1}^{4} P_i^0 = -2.29 - 5.23 + 1.1 - 4.45 = -10.87 \]

Also, we need to compute the \( P' \) which is the total profit when \( r_3 \) is removed from reducers of other arguments to find whether \( r_3 \) is the right choice to be reduced at this stage or not.

The profit \( P_i \)'s for arguments without \( r_3 \) as their reducer are:

\[
\begin{align*}
P_1^1 &= +0.70 \\
P_2^1 &= -2.61 \\
P_3^1 &= +1.1 \\
P_4^1 &= -2.2
\end{align*}
\]

\[ P' = \sum_{i=1}^{4} P_i^1 = -3.01 \]

\[ P - P' = -10.87 - (-3.01) = -7.86 \]

Since \( P - P' < P_3^0 \), \( r_3 \) can be chosen without any problem. Thus:

\[
\begin{align*}
B_3 &= \{1, 2, 4\} \\
U_1 &= \{2, 4\} \\
U_2 &= \{1, 4\} \\
U_4 &= \{1, 2\} \\
S_4 &= 3
\end{align*}
\]

Arrangement of \( P_i \)'s is:
4.3. Minimising Response Time

\[ P_1^1 > P_4^1 > P_2^1 \]

The argument \( r_1 \) may be selected. Previous \( P' \) without profit for argument \( r_3 \) is new \( P \) for this step:

\[ P = 0.70 - 2.61 - 2.2 = -4.11 \]

By removing \( r_1 \) from reducers of \( r_4 \) and \( r_2 \), new \( P' \) must be computed.

\[ P_1^2 = 0.70 \]
\[ P_2^2 = -2.16 \]
\[ P_4^2 = -0.13 \]
\[ P' = 0.70 - 2.16 - 0.13 = -1.59 \]
\[ P - P' = -4.11 - (-1.59) = -2.52 \]

Since \( P - P' < P_1^1 \), selection of \( r_1 \) is the right choice at this stage. Thus:

\[ B_1 = \{2, 4\} \]
\[ U_2 = \{4\} \]
\[ U_4 = \{2\} \]
\[ S_3 = 1 \]

Since by removing \( r_1 \) from reducers of \( r_4 \) and \( r_2 \) the profits \( P_2^3 = -2.05 \) and \( P_4^3 = -0.5 \) are negative and there is only one reducer for each one of the arguments, they must be processed without reducer in parallel. Thus, \( B_2 = \emptyset \) and \( B_4 = \emptyset \).

The sequence of execution is that first \( r_2 \) and \( r_4 \) are processed in parallel, since no indexes for them are assigned in the \( S_{i,1\leq i\leq 4} \) (\( S_1 = 0, S_2 = 0 \) and \( B_2 = \emptyset, B_4 = \emptyset \)). Then \( r_1 \) is processed. After processing \( r_1 \), the argument \( r_3 \) is processed. Figure 4.2 shows the sequence of execution.

\[ RES(B) = \max_{1\leq i\leq 4}(\max_{j_i\in B_i}(w_i^{j_i}) + E_i + D_i \cdot \Pi_{j_i\in B_i}(\rho_{j_i}^i) \cdot |r_i| + W_f + C_i \cdot \Pi_{j_i\in B_i}(\rho_{j_i}^i) \cdot |r_i|) + I \cdot \Pi_{j_i\in B_i}(\rho_{j_i}^i) \]

The conversion time for \( r_2, r_4 \) and \( r_1 \), in order, are included in \( w_2^1, w_4^1 \) and \( w_1^3 \). Thus the conversion time is computed for \( r_3 \) only. Since there is only one argument that the conversion time must be computed for it, there is no waiting time for starting the
conversion. Thus, $W_f = 0$.

The size of $r_1$ is reduced to $\Pi_{j=(2,4)} \rho_j^1 \cdot |r_1|$ and the size of $r_3$ is reduced to $\Pi_{j=(2,4,1)} \rho_j^3 \cdot |r_3|$.

$$RES(B) = \max((2.6, 1.6) + (4.2 \times 0.8 \times 0.3)) + 0.6 + (0.009 \times (0.4 \times 0.6 \times 0.4) \times 500) + 0 + (0.0007 \times (0.4 \times 0.6 \times 0.4) \times 500) + 2.7 \times (0.8 \times 0.3 \times 0.4 \times 0.6 \times 0.4)$$

$$RES(B) = (2.6 + 1.01) + 0.6 + 0.432 + 0.03 + 0.06 = 4.73$$

If the size of arguments are not reduced, the response time would be: $5.5 + 2.7 = 8.2$

### 4.3.8 A Near Optimal Solution

In the Hybrid algorithm, first it is assumed that all reducers for each argument are profitable as long as at least an argument can be found to have a positive profit. In each iteration, the most profitable argument is selected to be reduced by its reducers. Since the selected argument cannot be a reducer for other arguments in the next iteration, the global effect of removing it from reducers of other arguments are examined. Reducers of a selected argument may not remain direct reducers of the argument. They may also be reduced by other reducers in the next iterations of the algorithm. These constitute important features of the algorithm that result in a near optimal solution.

If, in the assumption that all reducers are profitable, no argument with a positive profit can be found, then it must be attempted to remove the reducers of each argument whose weights are greater than the weight of the argument. After pruning reducers, if no argument with a positive profit is found, then the algorithm terminates. In this case, the arguments have no reducers (all $B_i = \emptyset$), and they must be processed in parallel.

### 4.4 Summary

After decomposition of a global query into a collection of subqueries and their translation to the query language of component databases, subqueries are submitted to the component databases. The impact of subquery results on each other specifies which subqueries should be submitted earlier and which later. The subqueries that are submitted later use other subquery results for query processing in the component databases in order to reduce the size of results. If the results of the subqueries have no impacts on each other, then subqueries are submitted to the component databases for parallel execution.
The basis for identifying the impact of the subquery results on each other is the reductions between arguments of the data integration expression. The algorithm HYBRID is an attempt to minimise the response time by employing reductions, query computation costs at the components, data transmission and data conversion costs.

Because of preservation of full autonomy of the component database systems, a reduction such as semijoin cannot be used in the HDMDBS as it is used in homogeneous distributed database systems. Since component database systems accept only queries, a reduction operation such as semijoin in the HDMDBS is performed in a different way.

To obtain the result of a subquery, each component database is accessed once. The benefit of one access to each component database is that less interference to the current work of the component is made. Removing the restriction may cause component databases be accessed several times to answer the global query.

The cost of a subquery consists of query transmission time, query processing time in the relating component database, the time for transmission of the query processing result from component database to the submission site, conversion time and integration of the result with other subquery results in the submission site. If the size of the subquery is small, the query transmission time is ignored.

If the arguments of a data integration expression have impacts on each other, the impacts can be used to reduce the size of query processing results in the component databases. For example, if there is a reduction between arguments $r_i$ and $r_j$ in such a way that $r_j$ as a reducer can be used to reduce the size of $r_i$ in the relating component database, the reduction may be applied if it is profitable. Reducing the size of an argument is profitable if all the work that are performed for the reduction is less than the cost of obtaining that argument by itself.

Algorithm HYBRID is based on profitable reducers of arguments. At any time that an argument is chosen to be reduced, the argument is removed from reducers of other arguments. The global effect of removing the argument is examined. This is an important feature of the algorithm that considers the global impact of a reducer argument on the other arguments. That feature results in a near optimal solution.
This chapter deals with efficient integration of partial results obtained from the execution of subqueries in the component databases. After the submission of subqueries to the component databases, partial results of subquery processing in some of the components are available earlier than others. We take advantage of the delays before the arrival of the later partial results to perform operations on the available results.

In the delays, available partial results can gradually be integrated. Also, reductions between arguments of data integration expressions are used to reduce the size of available partial results or prepare them to speed up future computations.

5.1 Query Postprocessing

After decomposition of a global query and then translation to the query language of component databases, subqueries are submitted to the component databases for execution. Subqueries can be submitted to the components sequentially, in parallel, or in a mixture of sequential and parallel, as discussed in chapter 3. To simplify the problem, without loss of generality, it is assumed that after query decomposition and translation, subqueries are sent to the components in parallel.

Generally, integration begins when all of the partial results are available at the submission site. One of the characteristics of heterogeneous distributed multidatabase systems is that partial results, obtained from the processing of subqueries, may not be available in the submission site at the same time. Furthermore, after the submission of subqueries, results of query processing in some of the component databases may be available in the submission site earlier than others. There are many factors that cause a partial result be available earlier or later than others in the submission site.

Owing to autonomy, it is a component database system that decides when to respond to a subquery. Communication with a component database system is through the external user interface. The component database system is not aware which queries
are local user queries and which may be the queries sent by the heterogeneous database system.

The network links between the heterogeneous database system and component database systems varies from local area to wide area networks. Furthermore, some of the links may use low data bandwidth, while the others may employ high data bandwidth. Depending on the kind of link and the bandwidth, data transmission speed between the heterogeneous database system and a component database system may be very fast or it may be very slow.

Computer systems that run component databases may differ from the view point of power. This creates great impacts on query processing in the component databases. Execution of queries by a component database system that is run on a powerful computer system is faster than execution of the same queries by another component database system that is run on a poor computer system.

Subqueries submitted to component databases may vary from simple to complex subqueries. Results obtained from processing of simple subqueries in the components may be available earlier in the submission site than the results from processing complex subqueries.

Component database systems use different techniques and algorithms in the query processing. Query processing techniques employed by a component database may result in a low response time for a subquery, while the techniques applied by another component for the similar query may lead to a high response time.

At the time that subqueries are submitted to the component databases, some of the networks may be very busy, that will result in slow transmission of data from the components to the heterogeneous database system site.

Also of importance are data conversion times. Some partial results obtained from the processing of subqueries may need more data conversion time and more time to deal with the semantic heterogeneities than other partial results.

In such a situation, optimisation of query postprocessing can be critical for the system from which the user expects a fast response time. We take advantage of the delays that partial results are available later at the submission to perform operations on the already available partial result to minimise the response time.
5.2 Optimisation of Query Postprocessing

After the submission of subqueries to the component databases, in a specific moment of time, some partial results, obtained from the processing of the subqueries, are available at the submission site. Until other partial results arrive, it may be possible to:

1. prepare available partial results in order to speed up future computations,
2. integrate some of them to reduce the number of integration operations or to display a part of the final answer to the user.

Partial results can be prepared at the physical and at the logical levels. At the physical level, hashing, indexing and clustering techniques may be used. Thus, when other partial results are available at the submission site later, the time for integration operations may be reduced.

At the logical level, the preparation of available partial results may be possible; this may include reducing their size. Thus, the reductions on partial results reduce the integration time.

Reducing the number of integration operations is beneficial since the time for integration of the partial results is reduced which in turn results in a lower response time. Displaying a part of the final answer not only may be important to the user but also it may be excluded from the final answer.

In general, optimisation of query postprocessing problems can be stated as follows: Given a data integration expression \( e(r_1, \ldots, r_n) \), and some of the arguments, the problems are:

(i) what operations can be performed using available arguments to obtain a part of the final answer or to reduce the number of integration operations,
(ii) how to prepare them for future processing to speed up the computations.

All of the operations on available arguments are done with the assumption that the system has enough time to perform such operations until the next arguments are available. It is also assumed that the global data model of the heterogeneous database system is the relational modelling of data and that a data integration expression is an expression of relational algebra.
5.2. Optimisation of Query Postprocessing

5.2.1 Integration of Available Arguments

Under specific conditions, available arguments of a data integration expression can be integrated. The integration aims at reducing the number of integration operations or displaying a part of the final answer to the user. There are three cases:

1. Available arguments are adjacent, i.e. they are operands of an operation of the data integration expression.

2. Available arguments are non-adjacent but, with a transformation on the data integration expression, they will be adjacent.

3. Available arguments are not adjacent and no transformations can be carried out on the data integration expression to make them adjacent.

In the first case, where available arguments of an operation are adjacent, they can be integrated by performing the operation. In this way, the available arguments constitute a new available argument and the number of integration operations is reduced. Furthermore, if the new argument is part of the final answer, it can be shown to the user.

**Example 5.1** Consider the relational algebra expression $e = (r_1^* \Join r_2^*) \cup (r_3 \Join r_4)$. Arguments superscripted with asterisks are available. The join operation between two arguments $r_1^*$ and $r_2^*$ can be performed and the result can be shown to the user.

In the example, not only is the number of operations of the data integration expression reduced but a part of the final answer is shown to the user.

**Example 5.2** Consider the relational algebra expression $e = ((r_1^* - r_2^*) \Join r_3) - r_4^*$ in which arguments $r_1^*$, $r_2^*$ and $r_4^*$ are available. The difference operation for arguments $r_1^*$ and $r_2^*$ can be computed. The result of the operation does not constitute a part of the final answer. However, the number of operations of the data integration is reduced. Hence, it is beneficial to perform the difference operation.

In the second case, the data integration expression may be transformed so that the available arguments become adjacent. The transformation must produce a new expression which is equivalent to the data integration expression. Note that two equivalent expressions produce the same result.
Example 5.3 Consider the relational algebra expression $e = r_1^* - (r_2^* - r_3^*)$. Arguments $r_1^*$ and $r_2^*$ are available. Since they are not adjacent, no operations can be performed. By replacing the difference operations with intersection operations, applying De Morgan's laws, using the distributive laws of set algebra and the elimination of the complement operation, expression $e$ is transformed to $(r_1^* - r_2^*) \cup (r_1^* \cap r_3^*)$. After transformation, the available arguments become adjacent and the difference operation can be performed. Furthermore, as a part of the final answer, the result of the difference operation can be shown to the user.

It is notable that the transformation produces an equivalent expression which has one operation more than operations of the data integration expression. The benefit of the transformation is that, computation of the difference operation can be done immediately and the result can be shown to the user.

In the third case, where the available arguments are not adjacent and with transformations on the data integration expression they cannot be adjacent, other techniques that are presented later may be applied to perform some operations on them.

5.2.2 Preparation Techniques

Considering the preparation of available partial results, we will focus on the logical level. To prepare partial results at the logical level, it is necessary to detect the impacts of partial results on each other in order to reduce their size. The reductions, discussed in chapter 3, describe how the impacts of partial results on each other can be detected.

Example 5.4 Consider the relational expression $e = r_1^* - \pi_x (r_2^* \Join r_3^*)$. Available arguments are $r_1^*$ and $r_2^*$ which are not adjacent. Also, no transformations can be performed on the expression to make the arguments adjacent. By labelling an expression tree corresponding to $e$, the common operation and the labels of input edges to the operation for the available arguments are identified. The common operation is difference and the labels are $A(r_1^*)$ and $E(r_2^*)$. The reductions are obtained from the reduction expression tables (see chapter 3) for difference operation by applying the labels. The obtained reduction is $r_2^* \Join \pi_x (r_1^*)$. Thus, the argument $r_2^*$ can be reduced.

An optimisation technique based on marking tuples of available arguments is important when a data integration expression has two or more identical subexpressions. For instance, consider the skeleton of a data integration expression given in Figure 5.1. A subtree $T$ is common for the operations $\alpha$ and $\beta$ but only in the case $\beta$ does there ex-
5.2. Optimisation of Query Postprocessing

Figure 5.1: A skeleton of an expression tree with the common subtrees.

exists a reduction between \( r \) and \( s \). If we assume that a reduction between \( r \) and \( s \) allows replacement of \( r \) with \( f(r, s) \) then the right hand side subtree \( T \) becomes different from the left hand side subtree \( T \). Such optimisation is not correct because it is no longer possible to compute \( T \) once. The solution to this problem is in the marking of tuples in \( r \) with (+) and (-) markers and clustering \( r \) according to the markers possessed by the individual tuples. The tuples that belong to \( f(r, s) \) are marked with (+) while the other tuples, i.e. \( r - f(r, s) \), are marked with (-). Furthermore, computation of the subexpression \( T \) should be done in such a way that it returns two results, one for the evaluation of \( \alpha \) and the other for the evaluation of \( \beta \). The result of \( T \) which is later used as an argument of \( \alpha \) contains all tuples independent of their markers. On the other hand, every tuple in the result of \( T \) which becomes an argument of \( \beta \), such that it consists of a part of or the entire tuple marked with (-), should be discarded before computation of \( \beta \). Such a tuple does not affect the result of \( \beta \).

Example 5.5 Consider the data integration expression \( e = (s^* \cup t) - (r^* \bowtie w) \). Labelling the paths from the leaf nodes \( s^* \) and \( r^* \) to the root node (i.e. -) in the corresponding expression tree provides the labels \( A(s^*)+ \) and \( E(r^*) \). A reduction between \( r \) and \( s \) does not exist. According to other optimisation techniques, the expression can be transformed into a union of subexpressions \( (s^* - (r^* \bowtie w)) \cup (t - (r^* \bowtie w)) \). Labelling in the first subexpression indicates that \( r^* \) can be replaced with the result of \( r^* \bowtie s^* \). Obviously, the reduction of \( r^* \) is not correct because it destroys the common subexpressions. Therefore, \( r^* \) is marked and clustered into \( r^{**} = r^* \bowtie s^* \) and \( r^{***} = r^* - r^{**} \). When \( w \) is available, the entire result of \( r^* \bowtie w^* \) is stored in order to compute the second subexpression while only \( r^{**} \bowtie w \) is kept for computation of the first subexpression.
5.3 Query Postprocessing Algorithm

This section provides an algorithm for query postprocessing in heterogeneous, distributed multidatabase systems. It is assumed that parallel submission strategy is used to distribute subqueries. Moreover, we assume that it is possible to estimate how much time is needed to compute the subqueries within the respective component databases. Integration of the partial results is performed accordingly to the following algorithm.

5.3.1 Algorithm

Input:

An expression tree of the data integration expression \( e(r_1, \ldots, r_k) \) and a sequence of pairs \( < t_1, r_1 >, \ldots, < t_k, r_k > \) ordered in the ascending order of the values \( t_1, \ldots, t_k \). Each pair \( < t_i, r_i > \) for \( i = 1, \ldots, k \) represents an estimation of the time elapsed from the submission of a subquery \( q_i \) till the arrival of its partial result \( r_i \).

Output:

The result of \( e(r_1, \ldots, r_n) \).

Method

Let \( R \) be a set of partial results available for data integration. Initially \( R = \emptyset \).

When a partial result \( r_i \) arrives after time \( t_i \), it is inserted into \( R \). The system periodically inspects the set of partial results. If \( R \) is not empty, i.e. it contains the partial results \( r_i, r_{i+1}, \ldots, r_j \), then the following actions take place.

(i) All operations that have all of their arguments available are immediately computed and the partial results \( r_i, r_{i+1}, \ldots, r_j \) are removed from \( R \). The expression tree is modified after the computations. The system inspects the set \( R \) and if it is not empty, step (i) is repeated for the new arrivals. At any moment the system is able to determine the amount of time \( t_a \) which is left before the arrival of the next partial result \( r_{j+1} \).

(ii) The system identifies an appropriate syntax-based optimisation technique to re-shuffle the arguments to compute a part of the final answer and to eliminate some of the operations. All optimisations that require less time than \( t_a \) are performed. Then, the system inspects \( R \) for new arrivals. If it is not empty, step (i) is repeated. Otherwise the system continues optimisation.
5.3. Query Postprocessing Algorithm

Table 5.1: Arrival times of arguments of expression e.

<table>
<thead>
<tr>
<th>argument</th>
<th>arrival time</th>
</tr>
</thead>
<tbody>
<tr>
<td>r₁</td>
<td>3</td>
</tr>
<tr>
<td>r₂</td>
<td>4</td>
</tr>
<tr>
<td>r₃</td>
<td>7</td>
</tr>
<tr>
<td>r₄</td>
<td>4</td>
</tr>
<tr>
<td>r₅</td>
<td>12</td>
</tr>
<tr>
<td>r₆</td>
<td>4</td>
</tr>
<tr>
<td>r₇</td>
<td>7</td>
</tr>
</tbody>
</table>

(iii) The system identifies the reductions and all possible reductions of the arguments. All optimisations that can be done before the arrival of the next partial result are performed. If \( R \) is still empty, the system continues optimisations, otherwise it returns to step (i).

(iv) The system identifies all cases when reductions of the partial results conflicts with computation of the common subexpressions. In such cases, the tuples from the respective arguments are marked. If a set \( R \) is still empty the system waits for new partial results, as no other optimisations can be done. Otherwise, the system returns to step (i).

Example 5.6 Consider the relational expression \( e = ((r₁ \Join r₂) \Join r₃) \cup ((r₄ \Join r₅) - (r₆ \Join r₇)) \). Arrival times of the arguments are given in Table 5.1. At the time of 4, available arguments are \( r₁, r₂, r₄ \) and \( r₆ \). The join operation whose arguments are available can be computed. Let \( s₁ \) be evaluation result of \( (r₁ \Join r₂) \). Thus, expression \( e \) after computation of the join is equivalent to \( (s₁^* \Join r₃) \cup ((r₄^* \Join r₅) - (r₆^* \Join r₇)) \). Available arguments before the time of 7 are superscripted by asterisks. We assume that after computation of the join operation, and until arrival of the next arguments, there is enough time to perform some operations on the available arguments. Labelling an expression tree of \( e \) and analysis of the labels for the common operations of available arguments (see chapter 3) indicates that argument \( r₆^* \) can be reduced by \( r₄^* \). The reduction expression is \( r₆^* \Join x(r₄^*) \), where \( x \) is the join attributes between \( r₆^* \) and \( r₄^* \). When the reduction of argument \( r₆^* \) is completed, arguments \( r₃ \) and \( r₇ \) arrive. Now, both join operations of the expression can be computed, because their arguments are available. Let \( s₂ \) and \( s₃ \), in order, be evaluation results of \( (s₁^* \Join r₃^*) \) and \( (r₆^* \Join r₇^*) \). After computation of the join operations, expression \( e \) is equivalent to \( s₂^* \cup ((r₄^* \Join r₅^*) - s₃^*) \). The result of \( s₂ \) can be sent to the user as a part of the final answer.
5.4 Summary

We addressed optimisation of query postprocessing in heterogeneous distributed multi-database systems. We assumed that subqueries are submitted to the component databases in parallel. After submission, partial results of query processing in some of the component databases may be available in the submission site later than the others. The reasons may be:

- Autonomy of the component database systems.
- Data transmission speeds.
- Power of computer systems that run the component databases.
- Complexity of subqueries.
- Query processing techniques in the component databases.
- Network congestion at the time that subqueries are submitted and at the time that the results are transmitted to the heterogeneous distributed multidatabase system site.

Until other partial results arrive, it may be possible to:

1. prepare available partial results in order to speed up future computations,
2. integrate some of them to reduce the number of integration operations or to display a part of the final answer to the user.

The approach to optimisation of query postprocessing takes advantage of the time that the system is waiting until all partial results are available. Three optimisation techniques are proposed. The first optimisation technique is based on the transformation of data integration expressions making it possible to compute a part of the...
5.4. Summary

final answer from incomplete partial results; thus it is possible to eliminate some of the operations. Another optimisation technique reduces the size of available arguments through detection of the reductions between arguments of the data integration expressions. The third optimisation technique performs marking of the partial results allowing future computations to be performed at a higher speed.
Chapter 6

Static and Dynamic Query Optimisation

So far, in the optimisation of query postprocessing, it has been assumed that the heterogeneous distributed multidatabase system has enough time in the waiting times to perform all possible optimisation opportunities on available partial results obtained from the processing of subqueries in the component databases. Moreover, it has been assumed that the partial results arrive at predictable times at the submission site. Generally, the assumptions may not always be valid. First, the waiting times may be too small to perform all of the possible optimisations, or they may be too inconsiderable to perform any query optimisations at all. Second, some of the partial results may arrive earlier or later than the estimated times; thus it may not be possible to predict exactly at what times they are available.

This chapter deals with optimisation at the query postprocessing stage when partial results are available at the submission site at either predictable or unpredictable times. We will present algorithms for two different environments. One algorithm is proposed for a static environment in which the arrival times of partial results are predictable. Another is for a dynamic environment in which the arrival times of partial results cannot be predicted. Both algorithms take advantage of the waiting times to reduce the size of available partial results by applying the reductions between arguments of the data integration expression.

6.1 Static and Dynamic Environments

The estimated time at which a partial result will be available at the submission site is based on the time for query processing at the component database, data transmission time from the component to the submission site and the data conversion time. In a static environment, we assume that query processing time at the component databases and the data transmission time are not changed. Thus, they are reliable and can be trusted to be used as important parameters in the optimisation of query postprocessing.
In a dynamic environment, depending on the time that a subquery is sent to the component database, some of the cost parameters may change. For example, because of autonomy and workload, the component database system may respond to the subquery very fast or very slowly. Furthermore, at that time, the communication network may be idle or busy; that may, consequently, cause the data transmission time be low or high.

Generally, to make the estimated time better, an average of the parameters is estimated by taking into account different times for query processing and data transmission. As a result, with the estimated times, it is expected that some of the partial results will be available at the submission site earlier or later than the estimated times. Therefore, the estimated times cannot still be relied on for query optimisation in the dynamic environment.

Efficient processing of queries in both environments is very important. In the environments, algorithms are proposed to take the most advantage of delays to reduce the size of available partial results.

### 6.2 Assumptions and Formulation

A query $Q$ is decomposed and translated into a collection of subqueries and a data integration expression $e(r_1, \ldots, r_n)$, where any argument such as $r_i$ is the partial result of a subquery from query processing in the relating component database system. The order for integration of partial results is specified in the data integration expression.

To obtain the partial results, $n$ subqueries are sent to the component database systems in parallel. Assume that the time for transmission of each subquery to the relating component database system is negligible.

Query processing at the component databases and data transmission from component databases to the submission site may be performed in parallel. Parallel submission of subqueries is important where no other optimisation strategies can be applied to reduce the size of results in the component databases.

Integration of partial results begins when all $r_i$'s are transmitted and then they are converted to the global data model format. Conversion of partial results is carried out in the submission site, sequentially.

The arrival time of argument $r_i$ is represented by $A_i$. It was introduced in chapter 4. In the case that $r_i$ is not reduced before transmission, the arrival time is computed
as follows:

\[ A_i = E_i' + F_i' + D_i \cdot |r_i| \]

\( E_i' \) is the processing time of the subquery at the component database to obtain \( r_i \). \( F_i' \) and \( D_i \), in order, are the setup time and the data transmission speed from the component database system site to the submission site.

When \( r_i \) is arrived at and then converted, it is available. The argument \( r_i \) may not be converted at the arrival time because, at that time, conversion of other arguments may not yet have been completed.

Let sorting of \( A_i \)'s in an increasing order be \( \{A_{i_1}, A_{i_2}, \ldots, A_{i_n}\} \). In other words, argument \( r_{i_1} \) is the first argument and argument \( r_{i_n} \) is the last to arrive in the submission site. Similar to chapter 4, such notations are used when the sorting is important and necessary, otherwise the previous notations are applied.

Conversion of argument \( r_{i_1} \) is started immediately after its arrival. Conversion of an argument \( r_{i_k} \), with \( k > 1 \), may be delayed because conversions of previously arrived arguments may not be completed by the arrival time of the argument.

First we focus on when an argument can be available, i.e. when the conversion can be completed. We have already introduced \( T_{i_k} \) in chapter 4. It is the difference between arrival time of an argument with the arrival and conversion times of a previously arrived argument. In the case that all subqueries are sent to the component database systems in parallel, it is computed as follows:

\[ T_{i_k} = (A_{i_k+1} - A_{i_k}) - C_{i_k} \cdot |r_{i_k}|, k < n \]

If the conversion of argument \( r_{i_k} \) begins immediately after the arrival, and the conversion is completed before the arrival of the next argument, then \( T_{i_k} \) is a positive number. When the conversion is completed at the time that the next argument arrives, \( T_{i_k} \) is zero. If during the conversion of argument \( r_{i_k} \), the next argument arrives, \( T_{i_k} \) is a negative number.

The waiting time, \( W_{i_k} \) (also introduced in chapter 4), is the time that an argument \( r_{i_k} \) must wait until its conversion begins. The waiting time is computed from \( W_{i_k} = |Wait(T_{i_k-1})| \), where \( Wait(T_{i_0}) = 0 \). The function \( Wait() \) is computed as follows:

\[ Wait(T_{i_k}) = \min(\text{Wait}(T_{i_k-1}) + T_{i_k}, 0) \]

With the assumption that the conversion of an argument is carried out in the order that it arrives in the submission site, when the last argument which is \( r_{i_n} \) arrives, its conversion is delayed by the time of \( W_{i_n} \). Thus, the last argument is available at the time of \( A_{i_n} + W_{i_n} + C_{i_n} \cdot |r_{i_n}| \).
6.2. Assumptions and Formulation

After the last argument is available, integration begins. The time for integration is $I''$. It is computed from $I'' \cdot \prod_{k=1}^{n} |r_{ik}|$, where $I''$ is an integration constant and $|r_{ik}|$ is the size of argument $r_{ik}$.

The response time of query $Q$ is expressed as:

$$RESP(Q) = A_{in} + W_{in} + C_{in} \cdot |r_{in}| + I'' \cdot \prod_{k=1}^{n} |r_{ik}|$$

$W_{in}$ and $C_{in}$, in order, are the waiting time and the conversion constant for the argument that its arrival time is the maximum.

6.2.1 Gap between Arguments

Conversion of an argument such as $r_{ik}$ may be completed while the next argument (i.e. $r_{ik+1}$) has not yet arrived. The time between the end of the conversion and the arrival time of the next argument is called a gap and it is represented by $G_{ik}^{i+1}$. If conversion of $r_{ik}$ begins on its arrival, $G_{ik}^{i+1}$ is equal to $T_{ik}$. When conversion of the argument commences after its arrival, because conversions of other arguments are being carried out, the value of $G_{ik}^{i+1}$ is $T_{ik} - W_{ik}$. A positive value of $G_{ik}^{i+1}$ indicates that there is a gap between the argument $r_{ik}$ and $r_{ik+1}$. Negative or zero values of $G_{ik}^{i+1}$ denote that no gap exists between the arguments. Thus, to find the existence and the size of a gap, $G_{ik}^{i+1}$ is computed as follows:

$$G_{ik}^{i+1} = \max((T_{ik} - W_{ik}), 0)$$

The gap between two arguments $r_{ik}$ and $r_{ij}$ where $j > k$ and $j \leq n$ is computed as follows:

$$G_{ik}^{ij} = \sum_{h=k}^{j-1} G_{ih}^{i+1}$$

At the time that conversion of argument $r_{ik}$ begins, previously arrived arguments (i.e. $r_{i1}, r_{i2}, \ldots, r_{ik-1}$) are available. After the last argument (i.e. $r_{in}$) is available, integration can begin immediately. From the time that argument $r_{ik}$ is available until the last argument arrives, the the gap is $G_{ik}^{in}$. The gap between an argument $r_{ik}(k<n)$ and the last argument $r_{in}$ is called the total gap for argument $r_{ik}$.

In general, for $j > k$ and $j \leq n$, the total gap for argument $r_{ik}$ and the total gap for argument $r_{ij}$ have the relation $G_{ik}^{in} \geq G_{ij}^{in}$. On the one hand, the total gap for argument $r_{ik}$ may be more than the total gap for $r_{ij}$ with $j > k$. On the other hand, the number of available arguments at the arrival time of argument $r_{ik}$ is less than the number of available arguments at the arrival time of argument $r_{ij}$. This important fact denotes
6.2 Assumptions and Formulation

Table 6.1: Values for different parameters and available arguments at the arrival times.

| k | $A_{ik}$ | $C_{ik} - |r_{ik}|$ | $T_{ik}$ | $W_{ik}$ | $G^{ik+1}_{ik}$ | $G^{in}_{ik}$ | available arguments |
|---|---|---|---|---|---|---|---|
| 1 | 10 | 5 | -3 | 0 | 0 | 4 | — |
| 2 | 12 | 7 | +6 | 3 | 3 | 4 | after 3 UoT $r_{i1}$ |
| 3 | 25 | 2 | +1 | 0 | 1 | 1 | $r_{i1}, r_{i2}$ |
| 4 | 28 | 10 | -8 | 0 | 0 | 0 | $r_{i1}, r_{i2}, r_{i3}$ |
| 5 | 30 | 5 | Non | 8 | Non | Non | $r_{i1}, r_{i2}, r_{i3}$, after 8 UoT $r_{i4}$ |

Table 6.1 shows the arrival times, conversion times, waiting times, gaps between two arguments, the total gaps and available arguments at the arrival times for five arguments. In the second row of the table, the first argument (i.e. $r_{i1}$) arrives at the time of ten. The waiting time, $W_{i1}$, is zero which means after its arrival, conversion of $r_{i1}$ is started immediately. Thus, conversion of argument $r_{i1}$ is started at the time of ten. When argument $r_{i2}$ arrives, still three units of time (UoT) for conversion of $r_{i1}$ must elapse. There is a gap of three UoT between arguments $r_{i1}$ and $r_{i2}$. In fact, if conversion of argument $r_{i2}$ could be started on its arrival time, the gap would be six UoT. The total gap for argument $r_{i1}$ and $r_{i2}$ is four UoT. For argument $r_{i3}$, the total gap is one UoT. The last column represents what arguments are available on the arrival of an argument.

Integration can begin immediately when all arguments are available. The time for integration is $I'' \cdot \Pi_{i=1}^{n} |r_{i}|$. If the sizes of arguments before integration can be reduced, then the integration time can be decreased as well. The gaps between arguments are wasted times. If the gaps can be used to reduce the size of arguments, then the integration time is reduced without any extra time.

6.2.2 Response Time

An argument $r_{i}$ may have a set of reducers that are identified by reductions between the argument and other arguments of the data integration expression. Let the set of indexes of such reducers for argument $r_{i}$ be $U_{i}$. Assume only argument $r_{i}$ is available. Although the argument has a set of reducers, no reduction operations to reduce its size are possible because the reducers are not available. Assume that after argument $r_{i}$, argument $r_{j}$ is available and $j \in U_{i}$. Thus, argument $r_{j}$ reduces argument $r_{i}$. Suppose
the reduction expression is \( r_i \alpha_{(i,j)} r_j \), that results in a reduction to the size of \( r_i \). In the reduction operation, argument \( r_j \) is a reducer for argument \( r_i \). Let indexes of such reduction operations when argument \( r_j \) which is the \( j^{th} \) argument that has arrived and then is converted be a set \( V_j \). The ordered pair \((i, j)\) is a member of \( V_j \) and we assume that the reduction operation is known. Note that \( V_j \) is a subset of reduction operations for arguments \( r_i \) and \( r_j \) that are also feasible. A feasible reduction operation is an operation of a reduction expression whose operands are available.

Suppose a selectivity associated with the operation in \( r_i \alpha_{(i,j)} r_j \) is \( \rho^j_i \) and the time that is needed to perform \( \alpha_{(i,j)} \) operation, called reduction time, be \( o^j_i \). The selectivity ranges from 0 to 1. We assume that a reduction operation is performed in the worst time of complexity. With this assumption, reduction time \( o^j_i \) is computed from \( o^j_i \cdot |r_i| \cdot |r_j| \), where \( o^j_i \) is a constant for performing the reduction operation. After the operation, the size of \( r_i \) is reduced to \( \rho^j_i \cdot |r_i| \).

We also assume that effects of reduction operations are independent. Thus, for a subset of reduction operations in \( V_j \) that reduces the size of argument \( r_i \), after performing the reduction operations, the size of \( r_i \) is reduced to \( \Pi_{(i,j) \in V_j} \rho^j_i \cdot |r_i| \).

After the size of argument \( r_i \) is reduced, for those reduction operations in which argument \( r_i \) is involved, the reduction times are also reduced by \( \rho^j_i \). For example, after the reduction operation, reduction times \( o^k_i \) and \( o^h_i \) (if they exist), in order, are reduced to \( \rho^j_i \cdot o^k_i \) and \( \rho^j_i \cdot o^h_i \).

At specific moments, some reducers of argument \( r_i \) may not be feasible because the reducers are not available or there may be a time constraint on the reduction operations. Specifically, when only one argument is available, the set of feasible reduction operations is empty because no other arguments are available. Also, when an argument is the last available argument, the set of feasible reduction operations is empty because integration must begin immediately and thus there is no time to perform any reduction operation.

Our task is to find a subset \( B_i \) of feasible reduction operations \( V_i \) to perform the reduction operations in the gap between an argument \( r_i \) and the next argument that will be available after \( r_i \) for \((1 \leq i \leq n)\).

To express the response time in terms of reduction operations and gaps between arguments, the sorting notations are applied. For example, \( V_{ik} \) is the set of feasible reduction operations for arguments \( \{r_{i_1}, r_{i_2}, \ldots, r_{ik}\} \) and argument \( r_{ik} \) is the \( k^{th} \) argument that arrives in the submission site. Also, \( B_{ik} \), which must be found, is a subset of reduction operations in \( V_{ik} \). Note that both \( V_{i} \) and \( V_{in} \) are empty sets and thus \( B_{i} \).
and \( B_i \) are empty sets as well.

Thus, for \( n \) arguments, the overall response time is expressed as a function of
\[
(B_{i_1}, G_{i_1}^{i_1}), (B_{i_2}, G_{i_2}^{i_2}), \ldots, (B_{i_{n-1}}, G_{i_{n-1}}^{i_{n-1}})
\] as follows:
\[
RES((B_{i_1}, G_{i_1}^{i_1}),(B_{i_2}, G_{i_2}^{i_2}),\ldots,(B_{i_{n-1}}, G_{i_{n-1}}^{i_{n-1}})) = E_i' + F_i' + D_i \cdot |r_{i-1}| + W_{i-1} +
\]
\[
C_i \cdot |r_{i-1}| + \max\left(\sum_{h=2}^{n-1} (\sum_{(k,j) \in B_{ih}} \rho_{ij}^{ih}) - G_{ih}^{ih+1}, 0\right) + I'' \cdot \Pi_{h=1}^{n-1} |r_{ih}| \cdot \Pi_{h=2}^{n-1} (k,j) \in B_{ih} \rho_{ij}^{ih}
\]
To simplify the expression, let
\[
E_i = E_i' + F_i' + I'' \cdot \Pi_{h=1}^{n-1} |r_{ih}| \quad \text{and} \quad (B, G) = (B_{i_1}, G_{i_1}^{i_1}),(B_{i_2}, G_{i_2}^{i_2}),\ldots,(B_{i_{n-1}}, G_{i_{n-1}}^{i_{n-1}}).
\]

Minimising the response time is stated as follows:
Given \( Q = (I,(E_1,D_1,C_1,|r_1|,\{o_j^1,\rho_j^1|j \in U_1\}),(E_2,D_2,C_2,|r_2|,\{o_j^2,\rho_j^2|j \in U_2\}),\ldots,(E_n,D_n,C_n,|r_n|,\{o_j^n,\rho_j^n|j \in U_n\})\), such that \( I, E_i, D_i, C_i, o_j^i \) are positive numbers, \( |r_i| \)

is the size of \( r_i \), \( 0 \leq \rho_j^i \leq 1 \) and \( W_{ih} \) is a waiting time to start conversion of the last argument, find an optimal \( B \) for a static environment in time \( G \) in which \( E_1 \) and \( D_1 \) are reliable and for a dynamic environment in time varying \( G \) in which \( E_1 \) and \( D_1 \) are not reliable such that:
\[
RES(B,G) = E_i + D_i \cdot |r_{ih}| + W_{ih} + C_i \cdot |r_{ih}| +
\]
\[
\max\left(\sum_{h=2}^{n-1} (\sum_{(k,j) \in B_{ih}} \rho_{ij}^{ih}) - G_{ih}^{ih+1}, 0\right) + I \cdot \Pi_{h=1}^{n-1} (k,j) \in B_{ih} \rho_{ij}^{ih}
\]
is minimised.

### 6.3 Minimising Response Time

At the time that argument \( r_{ih} \) arrives, if the waiting time, \( W_{ih} \), is zero, then available arguments are \( \{r_{i_{h+1}}, \ldots, r_{i_{h-1}}\} \). Furthermore, if the gap \( G_{ih}^{ih+1} > 0 \), then not only are arguments \( \{r_{i_{h+1}}, \ldots, r_{i_{h-1}}\} \) available but there is time of \( G_{ih}^{ih+1} \) to perform reduction operations on the arguments. We can extend the gap to the total gap \( G_{ih}^{in} \). Using the total gap is justified by the fact that the gaps between the next arguments (i.e. \( r_{ih+1}, \ldots, r_{ih} \)) are included in the total gap; this assumes there are no gaps between them. The benefit of the total gap is that, since \( G_{ih}^{in} \geq G_{ih}^{ih+1} \), there may be more time for performing reduction operations on the available arguments. Another important benefit of the total gap is to avoid fragmentation of the gaps. For argument \( r_{ih} \), the total gap is \( \sum_{h=2}^{n-1} G_{ih}^{ih+1} \). It may happen that no reduction operations can be performed in \( G_{ih}^{ih+1} \), but in the total gap some reduction operations on the available arguments be possible.
6.3. Minimising Response Time

In contrast to these benefits, there are some disadvantages of using the total gap. For example, if the total gap is spent entirely on reducing the size of the available arguments, then no time remains for reducing the arguments that will be available later. Also, for a dynamic environment, where arguments may arrive earlier, the total gap is not a good parameter in scheduling an algorithm to reduce the response time. However, it will be shown that the total gap is very important in a static environment even if it is spent entirely on some available arguments. In a dynamic environment, the gaps between arguments and the total gap have no applications because at any time as a result of earlier or later arrival of arguments, the gaps cannot be relied on.

When the first argument, \( r_{i_1} \), arrives, after conversion, it is available. Generally, with one available argument, only physical optimisation such as hashing and indexing is possible. Since the aim is optimisation at the logical level, the physical optimisation is not considered. However, the gap between one argument and the next is a wasted time, thus it may be used for a physical optimisation.

Suppose the next argument, \( r_{i_2} \), has arrived and is then converted. The total gap for argument \( r_{i_1} \) indicates how much time exists for reducing the size of available arguments \( r_{i_1} \) and \( r_{i_2} \). The total gap may be related to the gaps between the next arguments. Thus, if the total gap for argument \( r_{i_2} \) is equal to the total gap for argument \( r_{i_3} \), there is no gap between arguments \( r_{i_2} \) and \( r_{i_3} \). In this case, argument \( r_{i_3} \) can be assumed to be an available argument and so on. If the total gaps for arguments \( r_{i_2} \) and \( r_{i_3} \) are not equal, there is a gap between the arguments.

Assume the total gaps for arguments \( r_{i_2} \) and \( r_{i_3} \) are not equal. Thus, the total gap for argument \( r_{i_2} \) is greater than the total gap for argument \( r_{i_3} \). Available arguments are \( r_{i_1} \) and \( r_{i_2} \) and the time for performing reduction operations can be up to the total gap \( G_{i_2}^{in} \). Suppose both arguments can reduce each other. Let the set of feasible reduction operations, \( V_{i_2} \), be \{\( (1, 2) \), \( (2, 1) \)\}. Assume that the reduction times for the corresponding reduction operations be \( o_{i_2}^{i_1} \) and \( o_{i_1}^{i_2} \) with selectivities of \( \rho_{i_2}^{i_1} \) and \( \rho_{i_1}^{i_2} \), respectively. Let \( o_{i_1}^{i_2} < o_{i_1}^{i_1} \). Suppose argument \( r_{i_2} \) with reduction expression \( r_2 \alpha_{(2,1)} r_1 \) is chosen to be reduced. Three cases may occur:

1. The reduction operation can be performed within the gap, i.e. \( o_{i_1}^{i_2} \leq G_{i_2}^{in} \).

2. The reduction operation can be done within the total gap, i.e. \( G_{i_2}^{in} < o_{i_1}^{i_2} \leq G_{i_2}^{in} \).

3. The reduction operation cannot be performed within the total gap, i.e. \( o_{i_1}^{i_2} > G_{i_2}^{in} \).

In the first and the second cases where there is enough time, reduction operation \( \alpha_{(2,1)} \) is performed and the size of argument \( r_{i_2} \) is reduced to \( \rho_{i_1}^{i_2}, |r_{i_2}| \). In the third case,
6.3. Minimising Response Time

the reduction operation may be started and depending on how great its effect is on
the integration time, it may be continued or terminated. However, there may be other
reduction operations with shorter reduction times and lower selectivities which, when
the next arguments are available, become feasible. Also, in a dynamic environment
where the next arguments may arrive earlier or later, reduction operations should be
chosen that have the most opportunity for completion. Furthermore, in contrast to a
static environment where arrival times of arguments are important factors for query
optimisation, in a dynamic environment, the gaps and total gaps are not reliable.

6.3.1 Static Environment Case

In a static environment, the arrival time of each argument can be computed in advance.
In this environment, it is assumed that each argument arrives exactly at the arrival
time. After two arguments have arrived and have been converted, optimisation begins.

In each gap, some reduction operations are feasible. If the operations are performed,
they may not be completed within the gap. If feasible reduction operations in every
gap could be performed and completed, integration would be carried out in the shortest
time. The gap between two arguments may be very small, or even zero, while the gap
between another two arguments may be very large.

When the $k^{th}$ argument ($k < n$) has arrived and then has been converted, the
total gap is $G_{ik}^n$. The set of feasible reduction operations is $V_{ik}$. Some of the feasible
reduction operations might be feasible before the arrival of the argument because their
arguments were available earlier. The total gap for previously available arguments is
likely more than the total gap for the arguments that are available later. Thus, for
each feasible reduction operation, the maximum total gap is assigned. In other words,
at the first moment that arguments of a reduction operation are available, the total
gap is assigned to the operation.

A reduction operation is profitable if it can be performed within the total gap. The
profit for reduction operation $\alpha_{(k,j)}$ is computed from:

$$P_{(k,j)} = \min((G_{im}^n - o_{ij}^k), 0) + (I - I \cdot p_{ij}^k), \quad m = \max(k, j)$$

The total gap is computed whenever both arguments $r_{ij}$ and $r_{ik}$ are available. Thus,
$m = \max(k, j)$. A profitable reduction operation may have a large reduction time and
the most profit in comparison to other reduction operations. Thus, the profit should
represent how much time is spent on the reduction operation. Also, it is important that
the profit can be gained as soon as arguments of the reduction operation are available.
6.3. Minimising Response Time

Thus, dividing the profit into the reduction time and multiplying it by the total gap represents how much profit, in comparison with others, is applicable. Applicability of the profit, represented by \( A_P(k,j) \) is computed from:

\[
A_P(k,j) = \frac{P_{\{k,j\}}}{\gamma_{ij}} \cdot G_{im}^n
\]

The feasible reduction operations should be chosen that are more applicable than the others. Accordingly, profits and then applicabilities of all feasible reduction operations with regards to the assigned total gaps are computed. Based on the applicabilities, a reduction operation is chosen that is the most applicable, i.e. it has the biggest value. Afterwards, the reduction operation is removed from the sets of feasible reduction operations and it is inserted into set \( B_{im} \) \((B_{im} = B_{im} + (k,j), m = max(j,k))\). Reduction times of other reduction operations that include the reduced argument and integration time are multiplied by the selectivity of the reduction operation. After performing a reduction operation, the total gaps are updated. The reduction time is taken out of the total gaps from the point before the chosen reduction operation, i.e. for \( h, (1 < h < m), m = max(j,k) \) and \( G_{ih}^n = max((G_{ih}^n - o_{ij}^k), 0) \). A function that updates the total gaps for arguments before the point, called \textit{update1()}, is as follows:

\[
\text{update1()}
\begin{align*}
\text{begin} \\
\text{for}(h = 2 \text{ to } m-1) \\
G_{ih}^n &= \max((G_{ih}^n - o_{ij}^k), 0) \\
\text{end};
\end{align*}
\]

The total gaps for reduction operations at the point and after the point are updated as follows:

\[
\text{RemainGap} = \max((G_{im}^n - o_{ij}^k), 0), \ m = max(j,k)
\]

The \textit{RemainGap} must be distributed among arguments \( r_{im} \) to \( r_{im-1} \), starting from the \((n - 1)^{th}\) argument to the \(m^{th}\) argument. The reason for such distribution is to use the gaps from the point that the reduction operation starts until the gaps between the next arguments are consumed by the reduction time. The following function updates the total gaps for the arguments at the point of reduction operation and the next arguments:

\[
\text{update2()}
\begin{align*}
\text{begin} \\
\text{RemainGap} &= \max((G_{im}^n - o_{ij}^k), 0) \\
\text{for}(h = n-1 \text{ to } m \text{ step -1})
\end{align*}
\]
begin
\[ G_{ih}^n = \min(\max((\text{RemainGap} - G_{ih}^n), 0), G_{ih}^n) ; \]
\[ \text{RemainGap} = \max((\text{RemainGap} - G_{ih}^n), 0) ; \]
end;

In the next step, if there is at least one total gap that is not zero, profits and the applicabilities for reduction operations are computed and the most applicable reduction operation is chosen. Similarly to the above, the procedure continues until all the total gaps become zero or no positive profits are obtained.

Algorithm STATIC

INPUT: \( n, I, \{(E_i, D_i, C_i, |r_i|), (o^i_j, p^i_j) | 1 \leq i \leq n, j \in U_i \} \)
OUTPUT: \( B_2, B_3, \ldots, B_{(n-1)} \)
\( B_{ik} = \emptyset, (1 \leq k \leq n) ; \)
compute the gaps and the total gaps for arguments \( r_{ik}, (1 < k < n) ; \)
while(1)
begin
\[ \text{TotalGaps} = 0; \]
for(\( h = 2 \) to \( n - 1 \))
\[ \text{TotalGaps} = \text{TotalGaps} + G_{ih}^n ; \]
if(\( \text{TotalGaps} == 0 \)) return;
for(\( h = 2 \) to \( n - 1 \))
\[ P(k,j) = \min((G_{ih}^n - o^i_j), 0) + (I - I \cdot p^i_j) | \]
\[ \{(j, k) \in V_{ih} \}, \{(j, k) \notin V_{ih}, 2 \leq hh < h \}; \]
if(all \( P(k,j) == 0 \)) return;
for(\( h = 2 \) to \( n - 1 \))
if(\( P(k,j) > 0 \)) \( A_{P(k,j)} = \frac{P(k,j)}{o^i_j} \cdot G_{ih}^n | \)
\[ \{(j, k) \in V_{ih} \}, \{(j, k) \notin V_{ih}, 2 \leq hh < h \}; \]
select \( \max(A_{P(k,j)}) ; \)
update1(); update2(); \( I = I \cdot p^i_j ; \)
\( B_{km} = B_{km} + (k, j) | \{m = \max(k, j)\} ; \)
for(\( h = 2 \) to \( n - 1 \))
begin
\[ o^i_{kk} = o^i_{kk} \cdot p^i_j | \{kk = k or jj = j\}, \{(kk, jj) \in V_{ih} \}; \]
\[ V_{ih} = V_{ih} - (k,j); \]
An Optimal Solution in STATIC Algorithm

In algorithm STATIC, an attempt is made to reduce the size of available arguments so that the maximum profits can be obtained from them by taking into account the reduction times and the total gaps. At first, the maximum total gaps are assigned to reduction operations, then the profits and the applicabilities are computed. By choosing a reduction operation that is the most applicable, the required time from the gaps is dedicated to the reduction time. In this way, those reduction operations are performed in the gaps that can reduce integration time the most.

In the worst case, all the arguments can be reducers of each other and, in all steps, the profits and the total gaps are positive. For \( n \) arguments, \((n - 1)(n - 2)\) reduction operations can be feasible. Note that the last argument cannot be reduced because after the arrival and the conversion, integration must begin immediately. Computation of the profits and the applicabilities, and then obtaining a maximum value of the applicabilities, in the worst case, are required to be performed \((n - 1)\) times.

6.3.2 Dynamic Environment Case

In a static environment, gaps between arguments are used as an important factor in minimising the response time. In a dynamic environment where the next arguments may arrive at any moment, the gaps are not applicable. Thus, the reduction operations should be chosen that have small reduction times. In this way, it is much more probable that the reduction operations will be completed before the arrival of the next arguments. Therefore, when a new argument is available, among available arguments, the argument is chosen to be reduced that has the smallest reduction time. When a new argument arrives during a reduction operation, continuation or termination of the operation is to be evaluated.

Assume that two arguments \( r_i \) and \( r_j \) are available and that they can reduce each other. The reduction operations are \( \alpha_{(i,j)} \) and \( \alpha_{(j,i)} \) with reduction times \( \sigma^j_i \) and \( \sigma^i_j \), respectively. The selectivities associated with the operations are \( \rho^j_i \) and \( \rho^i_j \). The reduction operations \( \alpha_{(i,j)} \) and \( \alpha_{(j,i)} \), in order, reduce the size of argument \( r_i \) and \( r_j \). Let \( \sigma^j_i < \sigma^i_j \). In the environment, completion of reduction operation \( \alpha_{(i,j)} \) is more probable. Thus, it is chosen to be performed. If, during the reduction operation, the next argument arrives, then the number of available arguments is increased and more reduction operations can be feasible.
6.3. Minimising Response Time

operations may be feasible. Thus, a new reduction operation may have lower reduction
time than if operation \( \alpha_{(i,j)} \) is terminated. If a new reduction operation is started
immediately, it is more probable that it will be completed.

Assume during reduction operation \( \alpha_{(i,j)} \), the next argument arrives. Let the set
of reduction operations on arrival of the next argument be \( V_k \). Suppose the remaining
time to complete the reduction operation be \( m^i_j \). If the smallest reduction time relating
to the reduction operations in \( V_k \) is greater than \( m^i_j \), reduction operation \( \alpha_{(i,j)} \) must be
continued. If it is equal to \( m^i_j \), but it has larger selectivity than \( \rho^i_j \), reduction operation
\( \alpha_{(i,j)} \) must also be continued because after completion of reduction operation \( \alpha_{(i,j)} \)
the integration time with smaller selectivity is further reduced. When the smallest
reduction time relating to the reduction operations in \( V_k \) is smaller than \( m^i_j \), reduction
operation \( \alpha_{(i,j)} \) should be terminated and the new reduction operation should be started
immediately.

If the next argument is the last one that arrives during the reduction operation,
then another strategy is used to continue or to terminate the operation. In this case,
continuation of reduction operation \( \alpha_{(i,j)} \) depends on how much benefit can be obtained.
Let the integration time before and after the reduction operation, in order, be \( I_0 \) and
\( I_1 \). Note that if the reduction operation is performed, then \( I_1 = I_0 \cdot \rho^i_j \). If \( m^i_j < I_0 \)
- \( I_1 \), the reduction operation should be continued, otherwise it should be terminated.
When \( m^i_j \) is equal to \( I_0 - I_1 \), termination of the reduction operation is preferable
because immediate integration is much more important than the reduction operation.
Moreover, it may happen that during integration, some parts of the final result are
ready and then are immediately shown to the user.

An important assumption is that whenever an argument arrives, if it is during
a reduction operation, the operation is interrupted and conversion of the argument
starts immediately. After conversion, the reduction operation by considering the new
argument may be resumed.

When a reduction operation is completed, the size of the reduced argument and
those reduction times that the reduced argument is involved in are updated. Then, the
next available argument that has the lowest reduction time is chosen to be reduced. If
there are no reduction operations for available arguments, then the system must wait
for a new argument to arrive.

In general, for a set of feasible reduction operations in \( V_k \), a reduction operation is
chosen that has the lowest reduction time. In the set, there may be reduction operations
with smaller selectivities but higher reduction times. Since new arguments may arrive
at any time, an attempt is made to choose the reduction operations that have the lowest reduction times. In this way, it is more probable that a reduction operation even with higher selectivity will be completed rather than a reduction operation that has higher reduction time and possibly lower selectivity. Thus, for a set of feasible reduction operations, choice of the next reduction operation is based on the ascending order of reduction times. If two reduction times are equal, the reduction operation is chosen that has smaller selectivity.

It can be observed that in the dynamic environment estimated times for arrival of arguments and the gaps are not applicable because at any time they may be destroyed. The strategy tries to use the gaps as much as possible. On the contrary, in a static environment the estimated times play critical roles in the optimisation.

Assumptions in DYNAMIC Algorithm

Since the environment is dynamic, in an algorithm that will be presented, the estimated times for query processing in a component and the time for data transmission from the component to the submission site are not applied. Thus, the arrival time of each argument is based on the time that the argument arrives at a real time.

A function, called $\text{watch}()$, watches for the arrival of new arguments. Whenever the function is called, if new arguments arrive, the function returns a true value, otherwise it returns a false value. The function $\text{watch}()$ takes care of the number of available arguments ($NoAvailArgs$) and conversion of newly arrived arguments as well. By calling the function $\text{watch}()$, whenever a new argument arrives, any operation is interrupted until the conversion of the argument is completed. After conversion, the number of available arguments is increased.

The logical optimisation begins when at least two arguments are available. Afterwards, the pairs of feasible reduction times and the associated selectivities (i.e. $\{\alpha_k^j, \rho_k^j|\ (j,k) \in V_{\text{NoAvailArgs}}\}$) are sorted in an ascending order. The list $L_O$ keeps the sorting. The number of elements of the list is represented by $|L_O|$. If there is enough time before a new argument arrives, reduction operations with the order of reduction times in the list of $|L_O|$ are performed one by one.

A reduction operation is performed in a number of time slices or it may be performed in its entirety. The function $\text{perform}(f(m), \text{TimeSlice})$ performs a part of reduction operation $f(m)$ in one time slice, where the function $\text{perform}(f(m))$ performs the reduction operation entirely. The function $f(m)$ specifies that which reduction operation is associated with which element in the list of $L_O$. After a time slice, the function $\text{watch}()$ is
called. If during the operation in the time slice a new argument arrives, continuation or termination of the operation is evaluated. The function $p(L_m)$ in the algorithm returns the selectivity relating to the reduction time in the list of $L_0$.

If a reduction operation is completed, the size of the reduced argument, the reduction times that the reduced argument is involved in and integration time are updated. Also, the set of optimal reduction operation (i.e. $B_i$) is updated by the completed reduction operation. The function update() does the updating. In updating the size of an argument that is reduced, the function $r(m)$ returns the index of the argument whose the reduction operation for it is completed.

**Algorithm DYNAMIC**

**INPUT:** $n, I, \{(C_i, |r_i|), (o^i_j, p^i_j) \mid 1 \leq i \leq n, j \in U_i\}$

**OUTPUT:** $B_2, B_3, \ldots, B_{(n-1)}$

$\text{NoAvailArgs} = 0; \ B_i = \emptyset, (1 \leq i \leq n)$;

while ($\text{NoAvailArgs} < 2$) watch();

repeat

\[ \text{sort}\{o^j_k, p^j_k \mid (j, k) \in V_{\text{NoAvailArgs}}\} \text{ in an increasing order of:} \]

\[ (1) \ o^j_k \text{ and (2) } p^j_k \text{ in a list of } L_0 = \{o_1, o_2, \ldots\}; \]

for ($m = 1$ to $|L_0|$)

begin

$\text{RemainTime} = 0; \ \text{StartDate} = 0; \ \text{Argument} = \text{NoAvailArgs};$

for ($\text{Start} = 0$ to $O_m$, $\text{TimeSlice}$)

begin

$\text{perform}(f(m), \text{TimeSlice});$

if (watch())

begin

$\text{RemainTime} = O_m - \text{Start};$

if (($\text{NoAvailArgs} == n)$ and ($\text{RemainTime} < I - I \cdot p(L_m)$))

begin

$\text{perform}(f(m));$

$\text{update}(I, B_{\text{Argument}}, \{o^j_k \mid (j, k) \in V_n\});$

return;

end;

$\text{MinRedTime} = \min(\{o^j_k \mid (j, k) \in V_{\text{NoAvailArgs}}\});$

if (($\text{RemainTime} > \text{MinRedTime}$) or (($\text{RemainTime} == \text{MinRedTime}$) and
6.3. Minimising Response Time

\[(p(L_m) > p(\text{MinRedTime}))\];

\begin{verbatim}
begin
StartOver = 1; break;
end;
end;
if(StartOver) break;
update(I, B_Argument, \{\sigma_k | (j,k) \in V_n\});
end;
while (Argument == NoAvailArgs) watch();
until(NoAvailArgs == n);
\end{verbatim}

An Optimal Solution in DYNAMIC Algorithm

In DYNAMIC algorithm, it is assumed that the times for performing various operations such as comparing, sorting and obtaining a minimum value are negligible. However, if the times are considerable, it may affect the efficiency of the algorithm.

Any feasible solution that results in reducing integration time is an optimal or a near optimal solution. If an attempt is made to schedule a plan based on performing the reduction operations that are much more beneficial in the gaps, then since the environment is dynamic, the plan can be destroyed at any time. As a result, no reduction operations may be completed.

The strategy of the algorithm is to perform as many reduction operations as possible in the gaps. It is preferable to complete a reduction operation with a lesser benefit than another reduction operation with a larger benefit that cannot probably be completed.

Except for the first and the last arguments, the sorting is needed when a new argument is available. In the worst case that all available arguments can be reducers for each other, the number of elements for \(m\) arguments that must be sorted is \(\frac{m!}{(m-2)!}\). With two arguments, the number of elements is 2. With \((n-1)\) arguments, the number is \((n-1)(n-2)\) elements. In general, the sorting takes the time of \(\sum_{i=2}^{n-1} i \cdot (i-1) \log(i \cdot (i - 1))\). Assuming that the minimum value of elements can be obtained from the sorting, it has no cost. In the worst case, with \((n-1)\) arguments and \((n-1)(n-2)\) values for reduction times, \((n-1)^2(n-2)\) iterations are required in the algorithm.
6.3. Minimising Response Time

<table>
<thead>
<tr>
<th>(i)</th>
<th>(E_i)</th>
<th>(D_i)</th>
<th>(C_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2</td>
<td>0.003</td>
<td>0.0004</td>
</tr>
<tr>
<td>2</td>
<td>0.4</td>
<td>0.004</td>
<td>0.0008</td>
</tr>
<tr>
<td>3</td>
<td>0.7</td>
<td>0.009</td>
<td>0.0007</td>
</tr>
<tr>
<td>4</td>
<td>0.6</td>
<td>0.005</td>
<td>0.0006</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
<td>0.006</td>
<td>0.0003</td>
</tr>
</tbody>
</table>

Table 6.2: Values for parameters \(E_i\) and \(D_i\).

\[
\begin{array}{ccccc}
  i/j & 1 & 2 & 3 & 4 & 5 \\
  \hline
  1 & 4.4, 0.6 & 2.1, 0.9 & \square & 4.9, 0.8 \\
  2 & 4.4, 0.7 & \square & \square & \square & 4.6, 0.7 \\
  3 & 2.2, 0.8 & \square & \square & 1.8, 0.8 & 2.3, 0.9 \\
  4 & \square & \square & 1.7, 0.9 & \square & \square \\
  5 & 6.4, 0.9 & \square & 2.2, 0.9 & \square & \square \\
\end{array}
\]

Table 6.3: Values for \(o_j^i\) and \(\rho_j^i\).

6.3.3 Example

Assume the data integration expression includes five arguments with the following sizes: \(|r_1| = 500\), \(|r_2| = 400\), \(|r_3| = 100\), \(|r_4| = 300\), \(|r_5| = 600\). The values for \(E_i\), \(D_i\) and \(C_i\) are given in Table 6.2. The values for reduction times and associated selectivities are given in Table 6.3. If there are no reductions between arguments, it is indicated by "\(\square\)" in the table. The integration time, \(t\), is 7.2 units of time (UoT). In a dynamic environment, the values for \(E_i\) and \(D_i\) can be ignored. Thus, in the dynamic environment, as soon as an argument arrives, the arrival time is registered. At first, \(B_i = \emptyset\), \((i = 1, 5)\).

Arrival times of arguments are computed as follows:

\[
A_i = E_i + D_i \cdot |r_i|
\]

\[
\begin{align*}
A_1 &= 0.2 + 0.003 \times 500 = 1.7 \\
A_2 &= 0.4 + 0.004 \times 400 = 2.0 \\
A_3 &= 0.8 + 0.009 \times 100 = 1.7 \\
A_4 &= 0.6 + 0.005 \times 300 = 2.1 \\
A_5 &= 0.5 + 0.007 \times 600 = 4.7
\end{align*}
\]

Conversion times of arguments are:

\[
C_1 \cdot |r_1| = 0.0004 \times 500 = 0.2
\]
6.3. Minimising Response Time

Table 6.4: Values for different parameters and available arguments at the arrival times.

| k | $A_{ik}$ | $C_{ik}$ | $|r_{ik}|$ | $T_{ik}$ | $W_{ik}$ | $G_{ik}^{i+1}$ | $G_{ik}^{in}$ | available arguments |
|---|---|---|---|---|---|---|---|---|
| 1 | 1.7 | 0.2 | -0.2 | 0 | 0 | 2.23 | — | after 0.2 + 0.07 $UoT r_1$ and $r_3$ |
| 2 | 1.7 | 0.07 | +0.23 | 0.2 | 0.03 | 2.23 | $r_1$, $r_3$ | $r_1$, $r_3$, after 0.4 $UoT r_2$, $r_4$ |
| 3 | 2.0 | 0.32 | -0.22 | 0 | 0 | 2.20 | — | $r_1$, $r_3$, $r_2$, $r_4$ |
| 4 | 2.1 | 0.18 | +2.42 | 0.22 | 2.20 | 2.20 | — | — |
| 5 | 4.7 | 0.18 | Non | 0 | Non | Non | — | — |

$C_2 \cdot |r_2| = 0.0008 \times 400 = 0.32$  
$C_3 \cdot |r_3| = 0.0007 \times 100 = 0.07$  
$C_4 \cdot |r_4| = 0.0006 \times 300 = 0.18$  
$C_5 \cdot |r_5| = 0.0003 \times 600 = 0.18$

The values for parameter $T_{ik}$, the waiting times, the gaps, the total gaps and the available arguments at the arrival time of the next argument are given in Table 6.4. The arrival times of two arguments $r_1$ and $r_3$ are equal. It is assumed that conversion of argument $r_1$ begins first and conversion of $r_3$ starts next.

**Static Environment Case**

First of all, for the reduction operations, the maximum total gaps are assigned. The total gap for reduction operations that involve both arguments $r_1$ and $r_2$ is 2.23 units of time. For reduction operations that involve $r_3$ and either of arguments $r_1$ or $r_2$, the total gap is 2.20 units of time. The total gap for reduction operations that include argument $r_4$ is also 2.20 units of time. The total gap for reduction operations that involve argument $r_5$ is zero.

With regard to Table 6.3, feasible reduction operations until the arrival of argument $r_5$ are: {$(1,3), (3,1)$} with the total gap of 2.23 and {$(1,2), (2,1), (3,4), (4,3)$} with the total gap of 2.20 units of time. The profits are computed as follows:

\[
P_{(k,i)} = \min((G_{im}^{in} - o_{ik}^{j,k}), 0) + (I - I \cdot \rho_{ik}^{j,k}) = \min((G_{im}^{in} - o_{ik}^{j,k}), 0) + I \cdot (1 - \rho_{ik}^{j,k})
\]

\[
P_{(1,3)} = \min((2.23 - 2.1), 0) + 7.2 \times (1 - 0.9) = 0.72
\]

\[
P_{(3,1)} = \min((2.23 - 2.2), 0) + 7.2 \times (1 - 0.8) = 1.44
\]

\[
P_{(1,2)} = \min((2.20 - 4.4), 0) + 7.2 \times (1 - 0.6) = 0.68
\]

\[
P_{(2,1)} = \min((2.20 - 4.4), 0) + 7.2 \times (1 - 0.7) = -0.04
\]

\[
P_{(3,4)} = \min((2.20 - 1.8), 0) + 7.2 \times (1 - 0.8) = 1.44
\]

\[
P_{(4,3)} = \min((2.20 - 1.7), 0) + 7.2 \times (1 - 0.9) = 0.72
\]

Applicabilities of the profits are computed as follows:
6.3. Minimising Response Time

\[ A_{R(k,j)} = \frac{P(k,j)}{G_{m}^{n}} \]

\[ A_{R(1,3)} = (0.72 / 2.1) * 2.23 = 0.76 \]
\[ A_{R(3,1)} = (1.44 / 2.2) * 2.23 = 1.46 \]
\[ A_{R(1,2)} = (0.68 / 4.4) * 2.20 = 0.34 \]
\[ A_{R(3,4)} = (1.44 / 1.8) * 2.20 = 1.76 \]
\[ A_{R(4,3)} = (0.72 / 1.7) * 2.20 = 0.93 \]

The maximum of the applicabilities is \( A_{R(3,4)} = 1.76 \), thus reduction operation \((3,4)\) with reduction time 1.8 and the selectivity of 0.8 is chosen. Thus, \( B_{4} = (3,4) \), i.e. when argument \( r_{4} \) is available, the reduction operation begins. The total gaps are updated. For reduction operations \{\( (1,3), (3,1) \)\} the total gap is changed to \((2.23 - 1.8) = 0.43\) and for reduction operations \{\( (1,2), (2,1), (4,3) \)\}, it is modified to \((2.20 - 1.8) = 0.40\) unit of time. The reduction times for those reduction operations that involve the reduced argument (i.e. \( r_{3} \)) are also updated. Thus, \( o_{3}^{3} = 2.1 * 0.8 = 1.68 \), \( o_{1}^{3} = 2.2 * 0.8 = 1.76 \) and \( o_{4}^{3} = 1.7 * 0.8 = 1.36 \) units of time. Integration time is altered to \( I = (7.2 * 0.8) = 5.76 \) units of time.

In the next iteration of the algorithm, the profits and applicabilities with new values are computed as follows:

\[ P(1,3) = \min((0.43 - 1.68), 0) + 5.76 * (1 - 0.9) = -0.67 \]
\[ P(3,1) = \min((0.43 - 1.76), 0) + 5.76 * (1 - 0.8) = -0.18 \]
\[ P(1,2) = \min((0.40 - 4.4), 0) + 5.76 * (1 - 0.6) = -1.7 \]
\[ P(2,1) = \min((0.40 - 4.4), 0) + 5.76 *(1 - 0.7) = -2.27 \]
\[ P(4,3) = \min((0.40 - 1.36), 0) + 5.76 *(1 - 0.9) = -0.38 \]

Since the profits are negative, the algorithm stops. The response time is computed as follows:

\[ RES = (4.7 + 0.18) + \max((1.8 - 2.20), 0) + 7.2 * 0.8 = 4.88 + 0 + 5.76 = 10.64 \]

The response time without using the gaps is: \( (4.7 + 0.18) + 7.2 = 12.08 \)

**Dynamic Environment Case**

In the dynamic environment, after the subqueries are submitted, the system waits for the arrival of the arguments. It is assumed that the arrival times of the arguments cannot be estimated. At the time of 1.7 two arguments, \( r_{1} \) and \( r_{3} \), arrive and then they are converted. With the assumption that argument \( r_{2} \) has not yet arrived, and arguments \( r_{1} \) and \( r_{3} \) are available, a reduction operation can be initiated. With regard to Table 6.3, there are reductions for the arguments with reduction times and selectivities of \((2.1, 0.9)\), and \((2.2, 0.8)\) for reducing the size of \( r_{1} \) and \( r_{3} \), respectively.
6.3. Minimising Response Time

Thus, the reduction operation with shorter reduction time is chosen to be performed. Therefore, reduction operation (1,3) with reduction time 2.1 is initiated. During the reduction operation, argument \( r_2 \) arrives. The reduction operation is interrupted and the conversion of \( r_2 \) begins while 0.03 unit of time has been spent on the reduction operation. The conversion of \( r_2 \) is completed at the time of \((2 + 0.32) = 2.32\) while argument \( r_4 \) has arrived before completion of the conversion. After the conversion of \( r_2 \), the conversion of \( r_4 \) begins. The conversion of \( r_4 \) is completed at the time of \((2.32 + 0.18) = 2.50\).

At the time of 2.50, available arguments are \( r_1, r_2, r_3 \) and \( r_4 \). While only 0.03 unit of time has been spent on the reduction operation (3,1), its continuation or termination must be evaluated.

Sorting of reduction operations based on ascending order of reduction times and the selectivities is: \{\((4,3), (3,4), (3,1), (1,2), (2,1)\}\}. The list of reduction times \( L_0 \) with the selectivities is \{\((1.7, 0.9), (1.8, 0.8), (2.2, 0.8), (4.4, 0.6), (4.4, 0.7)\)\}. Since the remaining reduction time for reduction operation (1,3) is \((2.1 - 0.03) = 2.07\), which is greater than the minimum reduction time of available arguments, it must be terminated. After termination, the reduction operation is added to the list of reduction operations and the first reduction operation from the list is chosen to be performed. Thus, reduction operation (4,3) is chosen. After completion, those reduction times in which argument \( r_4 \) is involved are updated. The new reduction time for (3,4) is \( o_3^4 \cdot \rho_3^4 = (1.8 \cdot 0.9) = 1.62 \) units of time. Also, reduction operation (4,3) is added to the set \( B_4 \). The integration time \( I \) is changed to \((7.2 \cdot 0.9) = 6.48 \) units of time.

The time after the operation is \((2.5 + 1.7) = 4.2\) and no new arguments have yet been arrived. The sorted list after completion of operation is: \{\((3,4), (1,3), (3,1), (1,2), (2,1)\)\}. The list of reduction times, \( L_0 \), with the selectivities is \{\((1.62, 0.8), (2.1, 0.9), (2.2, 0.8), (4.4, 0.6), (4.4, 0.7)\)\}. Note that when a reduction operation is terminated, it is added to the list and when completed, it is removed from the list.

Since no new arguments arrive, reduction operation (3,4) with the reduction time and the selectivity of \((1.62, 0.8)\) is initiated. During the reduction operation at the time of 4.7 argument \( r_5 \) arrives. The reduction operation is interrupted and the conversion of the argument begins. When the conversion is completed, it must be decided whether to resume the interrupted reduction operation or not. The remaining time to complete the operation is \(1.62 - (4.7 - 4.2) = 1.12\) unit of time. The integration time is 6.48 units of time. If the reduction operation is completed, integration time is reduced to \((6.48 \cdot 0.8) = 5.18\) units of time. Since the remaining time 1.12 is less than \((6.48 - 5.18) =\)
1.30, the reduction operation should be resumed and completed. By completion of the reduction operation, it is added to the set $B_4$ and the optimisation terminated.

The overall response time is:

$$RES = (4.7 + 0.18) + \max((1.7 + 1.62) - 2.20), 0) + (7.2 * 0.9 * 0.8) = 11.18$$

units of time. If the gaps between arguments were not used, the overall response time would be $(4.7 + 0.18) + 7.2 = 12.08$ units of time.

### 6.4 Summary

Two algorithms, called STATIC and DYNAMIC, were proposed for optimisation of queries in the heterogeneous distributed multidatabase systems at the postprocessing stage. In the STATIC algorithm, arrival times of partial results in the submission site can be predicted, whereas in the DYNAMIC algorithm, the arrival times are based on the actual time that they arrive.

It is assumed that subqueries are submitted to the component database systems in parallel. After submission, partial results may not arrive in the submission site at the same time. When partial results arrive at different times, optimisation of query postprocessing plays an important role in minimising the response time.

Query processing at the component databases and data transmission from components to the submission site, for a collection of subqueries that are submitted in parallel, are performed in parallel. Data conversion is done in the submission site after an argument arrives. Conversion of an argument may begin immediately when it arrives. If at the arrival time of an argument, conversion of previously arrived arguments have not been completed, conversion of the new argument is delayed. The delay is called the waiting time for the argument to begin the conversion operation.

A gap between two arguments, if it exists, is the time between end of conversion of an argument till the arrival of the next argument. Generally, to begin integration, all arguments must have arrived and the conversions must be completed. Since the gaps between arguments are the wasted times, they can be used to reduce the size of available arguments. As a result, integration time is reduced.

To reduce the size of available arguments, it is necessary not only that the reductions exist between available arguments but also that there is enough time to perform the reduction operations. In a static environment, the arrival times of arguments can be predicted and, based on the arrival times, the gaps between arguments are computed. To obtain the maximum benefit of the gaps, the total gaps for arguments are used.
The maximum of the total gaps is assigned to each feasible reduction operation. Then the profit of the reduction operations are computed. To choose the best reduction operation, the applicability of reduction operations are used. At any step, the reduction operation that is the most applicable is chosen.

In a dynamic environment an attempt is made to initiate a reduction operation that has the minimum reduction time. Since the next arguments may arrive at any time, it is more probable that the reduction operation with the minimum reduction time is completed. If, during a reduction operation, a new argument arrives, continuation or termination of the operation depends on how much time remains to complete the operation. When new arguments arrive, the number of feasible reduction operations may increase and thus there may be reduction times that are less than the remaining time for completion of the initiated reduction operation. If, during a reduction operation, the last argument arrives, continuation or termination of the operation depends on how much the integration time is affected. If by completing the reduction operation in the remaining time, the integration time is reduced more, the reduction operation is continued.

Based on the provided example, in a static environment, the most beneficial reduction operations that can be performed in the gaps are chosen because the gaps and the total gaps can be computed. In a dynamic environment, lack of such information results in the selection of reduction operations that have lower reduction times and that can thus be completed.
Chapter 7

Summary and Future Work

7.1 Summary

In this thesis we have addressed the optimisation of query processing in heterogeneous distributed multidatabase systems. In the systems, a query is decomposed into a collection of subqueries and also it is translated to a data integration expression. After translation to the query language of component databases, subqueries are submitted to the component databases for processing. Partial results, obtained from the processing of subqueries, are converted to a canonical data model format and they are then integrated.

The efficiency of query processing in the systems depends on detecting the impacts of the partial results on each other. The impacts are applied to efficiently submit subqueries to the component databases and to efficiently integrate the partial results obtained from query processing in the component databases.

Initially, to be independent of any particular database model, we introduced and adopted a generalised database model. In the database, queries are represented by algebraic expressions. Expression trees are used for two dimensional representation of corresponding expressions.

We proposed a technique to label expression trees corresponding to the database expressions for detecting optimisation of query processing. Depending on the output of an operation for just one of the inputs, operations in the database are classified. In general, operations are divided into five classes as follows:

1. Property decreasing operation.
2. Cardinality decreasing operation.
3. Weak cardinality decreasing operation.
4. Cardinality increasing operation.
5. Negative operation.

Classification of operations is important for detecting optimisation of query processing. In an expression tree, the class of a composite operation node with respect to one of the leaf nodes is identified by labelling the edges in the path of the leaf node to the operation.

In a data integration expression, a labelling technique is used to label an expression tree corresponding to the data integration expression for discovering the impacts of arguments on each other. The impacts are important for reducing the size of arguments before integration.

To detect reductions between arguments of an expression, the common operation for the arguments is identified. Then the labelling technique is used to find the labels of the input edges to the common operation. The labels of the input edges are the classes of the operations whose outputs are the inputs to the common operation for the arguments. Having the labels, reductions between the arguments are obtained from the reduction expression table that is designated to the common operation.

As a general technique, labelling is not only important and applicable to query optimisation in heterogeneous distributed multidatabase systems but also to the centralised and to the homogeneous distributed database systems.

Preserving the full autonomy of the underlying component database systems is a principle assumption in the query processing. With that assumption, the component databases accept only queries from heterogeneous distributed multidatabase systems. We proposed a method for reducing the size of query processing results in component databases using the results obtained from query processing in other component databases, while only queries can be sent to the components.

Optimisation of query processing in heterogeneous distributed multidatabase systems was identified and addressed at two stages. At the first stage, after a query is decomposed into a collection of subqueries, we focused on the submission of the subqueries to the component databases by considering the impacts of partial results on each other.

Three strategies can be applied for submission of the subqueries to the component databases as follows:

1. Sequential submission,
2. parallel submission and
3. hybrid (a mixture of sequential and parallel) submission.
If the submission of each subquery is delayed until the partial results of previously submitted subqueries is available, the strategy is called sequential submission. In this strategy, the partial results are used in the processing of the subquery in the relating component database.

If all subqueries are submitted to the component databases at the same time, they are processed in the component databases in parallel and the results may be transmitted in parallel to the submission site. This strategy is called parallel submission. Hybrid submission of subqueries is when some of the subqueries are submitted in parallel and the others are submitted sequentially to the component databases.

To obtain the result of a global query, the restriction of just one access to the component database for each subquery causes less interference to the component database. We proposed an algorithm for the submission order of subqueries to the component databases. In the algorithm, for each subquery, the component database is restricted to be accessed once and only queries can be sent to it. The response time is minimised by taking into account the following parameters:

- The impacts of subquery results on each other, i.e. the reductions between arguments of the data integration expression,
- query processing costs in component databases,
- data transmission costs from component database sites to the submission site,
- data conversion costs and
- data integration costs.

The impacts of subquery results can be detected by labelling an expression tree of the data integration expression. Moreover, query processing costs in the component databases can be obtained from one of the methods such as the database calibration and query classification.

The search space for finding an optimal submission of subqueries is extremely large. The problem of searching for the optimal solution is NP-complete. We applied heuristics to prune the search space. We attempted to choose the most profitable argument among the others to be reduced by the set of its reducers. The effect of this local optimum is also examined with relation to the global profits. This optimum result leads to a near optimal solution.

At the second stage, we concentrated on the efficiency of query postprocessing. Assuming that subqueries are submitted to the component databases in parallel, some
7.2. Future Work

Partial results of query processing in the component databases are available in the submission site earlier than others. Integration of partial results begins when all of the partial results are available. We took advantage of the delays in the arrival times of partial results before their integration. In the waiting time, we proposed different techniques to speed up future computations. Integration of available partial results is important when a part of the final answer can be shown to the user or when it can reduce the number of integration operations. Available partial results can be prepared by applying the reductions between them.

Two different environments of query postprocessing were considered. In the static environment, arrival times of partial results are predictable. Based on that assumption, we proposed an algorithm to minimise the response time by taking the maximum advantage of the delays. In the dynamic environment in which arrival times of partial results are not reliable, we presented another algorithm to perform as many reduction operations as possible in the delays.

We employed relational algebra operations and expressions as a particular case of the generalised database model. In the database, a relation is as a data container, a tuple of the relation is as a data item of the data container and an attribute of the tuple is a property of the data item. We presented some tables that demonstrate reduction expressions and reduction operations between any two arguments of a relational expression for the operations of relational algebra.

7.2 Future Work

The generalised database model

Owing to heterogeneity, HDMDBSs may need special operations. For example, when the global data model is relational modelling of data, join of two relations that are obtained from two component databases may be required to be changed to an outer-join because of inconsistencies in the mismatched keys of the join attributes. Lack of an operation such as outerjoin does not decrease expressive power of the generalised database model. However, such operations may be necessary to be defined in the model for such exclusive operations in HDMDBSs.

Labelling technique

We identified five classes of operations, and based on the classes, an expression tree is labelled for detecting optimisation of query processing. In the current situation,
the operation classes seem to be complete in a sense that they are sufficient to detect the reductions between arguments of an expression. Is it necessary to define other operation classes? This requires more research.

Application of labelling technique

We proposed labelling as a general technique for detecting optimisation of query processing in HDMDBSs, in centralised database systems, and in homogeneous distributed database systems. More comprehensive work on application of labelling in centralised and parallel database systems is needed. Specifically, for a system that is multiprocessor and each reduction operation can be given to independently be performed by one or more processors, more work is necessary to be done.

Submission order of subqueries

The search space for finding an optimal subquery submission is gigantic. The problem of searching for the optimal solution is NP-complete. We applied heuristics to prune the search space. We made an attempt to choose the most profitable argument among others to be reduced by the set of its reducers. The effect of this local optimum is also examined to the global profits that can be obtained. It may be a case that no profitable arguments that can be reduced be found. However, by applying a set of parallel and sequential reducers, an argument may become a profitable one. More research on this issue is needed.

Optimisation of query postprocessing

In the dynamic environment, arrival times of partial results are not predictable. Based on the assumption, in the waiting times we have made an attempt to initiate the feasible reduction operation with the least reduction time. By arrival of a new argument, the reduction operation is interrupted and it may be terminated. There may be cases that a sequence of reduction operations are initiated and then terminated by arrival of new arguments while one or more of the reduction operations could be completed with some profits. More work is needed, specifically, by taking into account the probability of arrival time of an argument.
Chapter 8

Appendix

This appendix contains proofs of reduction expressions that express the reductions between arguments of relational algebra expressions when the common operation for the arguments is join, difference, intersection, or union.

Recall that in a path of a leaf node in an expression tree, the label of the output edge of an operation node represents the class of the operation for an argument corresponding to the leaf node.

In an expression tree, the labels of the input edges to a commutative operation such as join, union or intersection can also commute. Thus, only one proof for each pair labels is required. For example, when the common operation is a commutative operation, proof of each reduction expression for labels $A_x(r)$ and $E_y(s)$ is similar to the proof for labels $A_y(s)$ and $E_x(r)$.

8.1 Join Operation

Proof of reduction expression for labels $A_x(r)$ and $A_y(s)$ of the input edges to join as the common operation has already been given. We will continue to present proofs of the reduction expressions for other labels.

Lemma 8.1.1 In an expression when a common operation between two arguments $r$ and $s$ is join and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)$ and $A_y(s)$, arguments $r$ and $s$ can reduce each other. The reduction expression for $r$ is $r \bowtie s$, and for $s$, the reduction expression is $s \bowtie r$.

Proof: All tuples of $r$ with $x$ attributes, and all tuples of $s$ with $y$ attributes flow to the operation. Tuples of relation $r$ can be reduced to the tuples that can be joined with tuples of $s$. Tuples of relation $s$ can be reduced to the tuples that can be joined with tuples of $r$.
Lemma 8.1.2 In an expression when a common operation between two arguments \( r \) and \( s \) is join and the labels of the input edges for the common operation node in the corresponding expression tree are \( A_x(r) \) and \( N_y(s)^+ \), argument \( r \) can reduce argument \( s \). The reduction expression for \( s \) is \( s \bowtie \pi_k(r) \), where \( k = x \cap y \). For \( r \), there is no reduction expression.

Proof: None of the tuples of \( s \) flow to the common operation because the input edge to the operation is labelled by \( N_y(s)^+ \). Tuples that flow to the operation are all tuples of \( r \) with \( x \) attributes for the edge labelled by \( A_x(r) \) and tuples of other relational table(s) other than \( s \) that have \( y \) attributes for the edge labelled by \( N_y(s)^+ \).

Let \( T_{Nys^+} \) be the tuples that flow to the common operation for the edge labelled by \( N_y(s)^+ \). Also, let \( t_i \) be a tuple in \( \pi_y(s) \). Then, \( t_i \notin T_{Nys^+} \). Assume \( \beta \) is an operation whose output edge is labelled by \( N_y(s)^+ \) (Figure 8.1).

- Relation \( s \) can be reduced to \( s \bowtie \pi_k(r) \). Suppose that \( t_i \) cannot be joined with any tuples of \( \pi_x(r) \). If a tuple such as \( t_i \) is removed from \( \pi_y(s) \), then the effect may be such that, in the output of operation \( \beta \), tuple \( t_i \) appears. Since tuple \( t_i \) cannot be joined with any tuples of \( r \), then it has no impact on the output of the join operation. Hence, a tuple such as \( t_i \in \pi_y(s) \) that cannot be joined with \( r \) can be removed from \( s \). In other words, \( s \) can be reduced to \( s \bowtie \pi_k(r) \).

- Tuples of \( r \) cannot be reduced with regards to tuples of \( s \) because \( T_{Nys^+} \) consists of tuples of other relational table(s) other than tuples of \( \pi_y(s) \). Thus, for \( r \) there is no reduction expression.
Lemma 8.1.3 In an expression when a common operation between two arguments $r$ and $s$ is join and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)$ and $E_y(s)$, arguments $r$ and $s$ can reduce each other. The reduction expression for $r$ is $r \bowtie \pi_k(s)$ and for $s$ the reduction expression is $s \bowtie \pi_k(r)$, where $k = x \cap y$.

Proof: All tuples of relational table $r$ with $x$ attributes flow to the common operation for the edge labelled by $A_x(r)$. Some tuples of $s$ with $y$ attributes flow to the common operation for the other input edge that is labelled by $E_y(s)$.

Let $s'$ be those tuples of $s$ that flow to the operation, i.e. $s' \subseteq s$.

- Relation $r$ can be reduced to $r \bowtie \pi_k(s)$. If $s' \equiv s$, the tuples of $r$ can be reduced to those tuples that can be joined with tuples of $s$ with $y$ attributes. However, it is assumed that only some tuples of $s$ with $y$ attributes flow to the operation. Relation $r$ can be reduced to $r \bowtie \pi_k(s')$ because those tuples of $r$ are removed that have no impacts on the join operation. We need to prove that $r$ can also be reduced to $r \bowtie \pi_k(s)$.

Those tuples in $r$ that can be joined with tuples in $s'$ are also joinable with tuples in $s$. Hence, $(r \bowtie \pi_k(s')) \subseteq (r \bowtie \pi_k(s))$. Thus, $r$ can be reduced to $r \bowtie \pi_k(s)$ without any impact on the output of the join operation.

- Relation $s$ can be reduced to $s \bowtie \pi_k(s)$. Since $s' \subseteq s$, then $(s' \bowtie \pi_k(r)) \subseteq (s \bowtie \pi_k(r))$. Thus, $s$ can be reduced to $s \bowtie \pi_k(s)$ without any effect on the output of the join operation.

Lemma 8.1.4 In an expression when a common operation between two arguments $r$ and $s$ is join and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)$ and $E_y(s)\+$, argument $r$ can reduce argument $s$. The reduction expression for $s$ is $s \bowtie \pi_k(r)$, where $k = x \cap y$. For $r$, there is no reduction expression.

Proof: All tuples of relational table $r$ with $x$ attributes flow to the common operation for the edge labelled by $A_x(r)$. Some tuples of $s$, in addition to tuples of other relational table(s), all with $y$ attributes, flow to the common operation for the other input edge labelled by $E_y(s)\+$.

Let $s'$ be those tuples of $s$ that flow to the operation for the edge labelled by $E_y(s)\+$. Thus, $s' \subseteq s$. Also, let $T_{sy\+}$ be the tuples except tuples of $s'$ that flow to the operation for this edge.
8.1. Join Operation

• Relation $s$ can be reduced to $s \bowtie \pi_k(r)$. Since $s' \subseteq s$, then $(s' \bowtie \pi_k(r)) \subseteq (s \bowtie \pi_k(r))$. Thus, if $s$ is reduced to $s \bowtie \pi_k(r)$, the output of the join operation is not affected.

• Relation $r$ cannot be reduced. Let $t_i$ be a tuple in $r$ that cannot be joined with any tuples of $s$ but which can be joined with tuples of $T_{sy+}$. The tuple $t_i$ cannot be removed from $r$ for the reason that it cannot be joined with tuples of $s$. The tuple $t_i$ is a joinable tuple with tuples of $T_{sy+}$. Thus, $r$ cannot be reduced.

Lemma 8.1.5 In an expression when a common operation between two arguments $r$ and $s$ is join and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)+$ and $A_y(s)+$, neither $r$ nor $s$ can reduce each other.

Proof: All tuples of relational table $r$, in addition to tuples of other relational table(s), all with $x$ attributes, flow to the common operation for one edge, and all tuples of $s$, in addition to tuples of other relational table(s), all with $y$ attributes, flow to the common operation for the other input edge to the operation.

• Relation $r$ cannot be reduced. Let $t_i$ be a tuple in $r$. If tuple $t_i$ cannot be joined with any tuples of $s$, it cannot be removed from $r$. The tuple $t_i$ may be a joinable tuple with tuples of other relational table(s) other than $s$ that flow to the operation. Hence, tuples of $r$ cannot be reduced only with regards to tuples of $s$. Thus, $r$ cannot be reduced.

• Using the same approach as above, relation $s$ cannot be reduced.

Lemma 8.1.6 In an expression when a common operation between two arguments $r$ and $s$ is join and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)+$ and $N_y(s)+$, neither $r$ nor $s$ can reduce each other.

Proof: All tuples of relational table $r$, in addition to tuples of other relational table(s), all with $x$ attributes, flow to the common operation for one edge. Also, tuples of other relational table(s) except tuples of $s$, flow to the common operation for the other input edge to the operation.

Not only does there exist no information regarding tuples of $s$ but also flow of tuples of other relational table(s) to the operation does not permit to remove any tuple from $r$ or $s$. Thus, neither $r$ nor $s$ can be reduced.
Lemma 8.1.7 In an expression when a common operation between two arguments \( r \) and \( s \) is join and the labels of the input edges for the common operation node in the corresponding expression tree are \( A_x(r) + \) and \( E_y(s) \), argument \( s \) can reduce argument \( r \). The reduction expression for \( r \) is \( r \times \pi_k(s) \), where \( k = x \cap y \). For \( s \), there is no reduction expression.

**Proof:** All tuples of relational table \( r \), in addition to tuples of other relational table(s), all with \( x \) attributes, flow to the common operation for one edge and some tuples of \( s \) with \( y \) attributes flow to the common operation for the other input edge to the operation.

Let \( s' \) be those tuples of \( s \) that flow to the operation. Thus, \( s' \subseteq s \). Let \( t_j \in s \).

- Relation \( r \) can be reduced to \( r \times \pi_k(s) \). Since \( s' \subseteq s \), then \( (r \times \pi_k(s')) \subseteq (r \times \pi_k(s)) \). Thus, \( r \) can be reduced to \( r \times \pi_k(s) \) without any impact on the output of the join operation.

- Relation \( s \) cannot be reduced. If a tuple such as \( t_j \) in \( s \) cannot be joined with any tuples of \( r \), then it cannot be removed from \( s \). The tuple \( t_j \) may be a joinable tuple with tuples of other relational table(s) other than \( r \). Thus, \( s \) cannot be reduced.

Lemma 8.1.8 In an expression when a common operation between two arguments \( r \) and \( s \) is join and the labels of the input edges for the common operation node in the corresponding expression tree are \( A_x(r) + \) and \( E_y(s) + \), neither \( r \) nor \( s \) can reduce each other.

**Proof:** All tuples of relational table \( r \), in addition to tuples of other relational table(s), all with \( x \) attributes, flow to the common operation for one edge, and some tuples of \( s \), in addition to tuples of other relational table(s), all with \( y \) attributes, flow to the common operation for the other input edge to the operation.

Neither \( r \) nor \( s \) can be reduced. If a tuple in \( r \) is removed because it cannot be joined with any tuples of \( s \), then the tuple may be joined with tuples of other relational table(s) other than \( s \). Thus, relation \( r \) cannot be reduced. For the same reason, \( s \) cannot be reduced by \( r \) because tuples of other relational table(s) other than \( r \) also flow to the operation for the edge labelled by \( A_x(r) + \).

Lemma 8.1.9 In an expression when a common operation between two arguments \( r \) and \( s \) is join and the labels of the input edges for the common operation node in the
Lemma 8.1.10 In an expression when a common operation between two arguments r and s is join and the labels of the input edges for the common operation node in the corresponding expression tree are $N_x(r)+$ and $E_y(s)$, argument s can reduce argument r. The reduction expression for r is $r \Join \pi_k(s)$, where $k = x \cap y$. There is no reduction expression for s.

Proof: None of the tuples of relational table r except tuples of other relational table(s) with x attributes flow to the common operation for the edge labelled by $N_x(r)+$. For the other input edge, some tuples of s with y attributes flow to the operation.

Let $T_{Nxr+}$ be all those tuples for the edge labelled by $N_x(r)+$ and $s'$ be those tuples of s that flow to the operation for the other edge. Also, let $t_i$ be a tuple in r. Then, $s' \subseteq s$ and $\pi_x(t_i) \notin T_{Nxr+}$. Assume $\beta$ is an operation whose output edge is labelled by $N_y(s)+$ (Figure 8.2).

- Relation r can be reduced to $r \Join \pi_k(s)$. Suppose that $t_i$ cannot be joined with any tuples of s. Since $s' \subseteq s$, $t_i$ is not joinable with any tuples of $s'$. If a tuple
such as $t_i$ is removed from $r$, then the effect may be such that, in the output of operation $\beta$ (i.e. in $T_{N_xr+}$), tuple $\pi_x(t_i)$ appears. Since tuple $t_i$ cannot be joined with any tuples of $s$ and thus $s'$, then it has no impact on the output of the join operation. Hence, the tuples such as $t_i \in r$ that cannot be joined with $s$ can be removed from $r$. In other words, $r$ can be reduced to $r \Join \pi_k(s)$.

- Tuples of $s$ cannot be reduced with regards to tuples of $r$. If tuples of $s$ are reduced to those tuples that can be joined with tuples of $r$, then there exists a tuple in $s$ that cannot be joined with tuples of $r$ but can be joined with a tuple in $T_{N_xr+}$. Thus, for $s$ there is no reduction expression.

**Lemma 8.1.11** In an expression when a common operation between two arguments $r$ and $s$ is join and the labels of the input edges for the common operation node in the corresponding expression tree are $N_x(r)+$ and $E_y(s)+$, neither $r$ nor $s$ can reduce each other.

**Proof:** None of the tuples of $r$ except tuples of other relational table(s) with $x$ attributes flow on the common operation for the edge labelled by $N_x(r)+$. For the other input edge, some tuples of $s$, in addition to tuples of other relational table(s), all with $y$ attributes, flow to the operation.

Since there are tuples of other relational tables that flow to the operation, neither $r$ nor $s$ can be reduced. Thus, there are no reduction expressions for $r$ and $s$.

**Lemma 8.1.12** In an expression when a common operation between two arguments $r$ and $s$ is join and the labels of the input edges for the common operation node in the corresponding expression tree are $E_x(r)$ and $E_y(s)$, arguments $r$ and $s$ can reduce each other. The reduction expression for $r$ is $r \Join \pi_k(s)$ and for $s$ is $s \Join \pi_k(r)$, where $k = x \cap y$, $x$ are attributes of $r$ and $y$ are attributes of $s$.

**Proof:** Some tuples of relational table $r$ with $x$ attributes flow to the common operation for one edge and some tuples of $s$ with $y$ attributes flow to the common operation for the other input edge to the operation node.

Let $r'$ be the tuples of $r$ that flow to the common operation for the edge labelled by $E_x(r)$. Let $s'$ be the tuples of $s$ that flow to the operation for the edge labelled by $E_y(s)$. Thus, $r' \subseteq r$ and $s' \subseteq s$

- Relation $r$ can be reduced to $r \Join \pi_k(s)$. There exists $t_i \in r$, such that $\pi_x(t_i)$ cannot be joined with any tuples in $\pi_y(s)$. Since $s' \subseteq s$, then $\pi_x(t_i)$ cannot be joined with any tuples in $\pi_y(s')$. 


8.1. Join Operation

If \( t_i \in r \) but \( t_i \not\in r' \), then, by removing \( t_i \) from \( r \), the evaluation result of the join operation is not affected.

If \( t_i \in r' \), then \( t_i \) must be in \( r \). If \( \pi_x(t_i) \) cannot be joined with any tuples in \( \pi_y(s) \), then \( t_i \) can be removed from \( r \) without any impact on the evaluation result of the join operation. Thus, \( r \) can be reduced to those tuples that can be joined with \( s \). Hence, the reduction expression for \( r \) is \( r \bowtie \pi_k(s) \).

- Using the same approach as above, relation \( s \) can be reduced to \( s \bowtie \pi_k(s) \).

Lemma 8.1.13 In an expression when a common operation between two arguments \( r \) and \( s \) is join and the labels of the input edges for the common operation node in the corresponding expression tree are \( E_x(r) \) and \( E_y(s) \), argument \( r \) can reduce argument \( s \). The reduction expression for \( s \) is \( s \bowtie \pi_k(r) \), where \( k = x \cap y \). For \( r \), there is no reduction expression.

Proof: Some tuples of relational table \( r \) with \( x \) attributes flow to the common operation for one edge and some tuples of \( s \), in addition to tuples of other relational table(s), all with \( y \) attributes flow to the common operation for the other input edge.

Let \( r' \) be the tuples of \( r \) that flow to the common operation for the edge labelled by \( E_x(r) \) and \( s' \) be the tuples of \( s \) that flow to the operation for the edge labelled by \( E_y(s) \). Thus, \( r' \subset r \) and \( s' \subset s \).

- Relation \( s \) can be reduced to \( s \bowtie \pi_k(r) \). There exists \( t_i \in s \), such that \( \pi_y(t_i) \) cannot be joined with any tuples in \( \pi_x(r) \). Since \( r' \subset r \), then \( \pi_y(t_i) \) cannot be joined with any tuples in \( r' \).

  If \( t_i \in s \) but \( t_i \not\in s' \), then, by removing \( t_i \) from \( s \), evaluation result of the join operation is not changed.

  If \( t_i \in s' \), then \( t_i \) must be in \( s \). However, since \( \pi_y(t_i) \) cannot be joined with any tuples in \( \pi_x(r) \), then \( t_i \) can be removed from \( s \) without any impact on the evaluation result of the join operation. Thus, \( s \) can be reduced to those tuples that can be joined with \( r \). Hence, the reduction expression for \( s \) is \( s \bowtie \pi_k(r) \).

- Relation \( r \) cannot be reduced with respect to tuples of \( s \). If \( r \) is reduced, then there may be a tuple in \( r \) such that before removal can be joined with tuples of other relational table(s) other than \( s \). Thus, \( r \) cannot be reduced.
Lemma 8.1.14 In an expression when a common operation between two arguments $r$ and $s$ is join and the labels of the input edges for the common operation node in the corresponding expression tree are $E_x(r)+$ and $E_y(s)+$, neither $r$ nor $s$ can reduce each other.

Proof: Some tuples of relational table $r$, in addition to tuples of other relational table(s), all with $x$ attributes, flow to the common operation for one edge. Some tuples of $s$, in addition to tuples of other relational table(s), all with $y$ attributes, flow to the common operation for the other input edge.

Neither $r$ nor $s$ can be reduced. If $r$ is reduced with respect to tuples of $s$, then there may be a tuple in $r$ that is removed by a reduction operation. However, the tuple may be a joinable tuple with tuples of other relational table(s) other than $s$. Thus, $r$ cannot be reduced.

Using the same approach as above, $s$ cannot be reduced.

8.2 Difference Operation

Proof of reduction expressions for labels $A_x(r)$ and $A_y(s)+$ of the input edges to a difference as the common operation has already been given. We will continue to present the proofs of reduction expressions for other labels.

Since difference is a non-commutative operation, the labels of the input edges to a difference operation node in an expression tree are distinguished from each other such that each one of the labels is on the right hand side or on the left hand side of the operation. As a convention, we assume that for each pair of the labels, the first one represents the label of the left hand side input edge and the second one represents the label of the right hand side input edge to the difference operation node.

Lemma 8.2.1 In an expression when a common operation between two arguments $r$ and $s$ is difference and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)$ and $A_y(s)$, arguments $r$ and $s$ can reduce each other. The reduction expression for $r$ is $r \rightarrow_k s$, where $k = x \cap y$. For $s$, the reduction expression is $s \leftarrow_k r$.

Proof: All tuples of $r$ with $x$ attributes, and all tuples of $s$ with $y$ attributes flow to the operation. Output of the operation consists of the tuples in $\pi_x(r)$ that cannot be joined with any tuples in $\pi_y(s)$.
8.2. Difference Operation

Since all tuples of \( r \) and all tuples of \( s \), in order, with \( x \) and \( y \) attributes flow to the common operation, \( r \) can be reduced to \( r - s \).

Tuples of relation \( s \) can be reduced to the tuples that can be joined with tuples of \( r \) because the joinable tuples in \( s \) have impacts on the difference operation. Thus, \( s \) can be reduced to \( s \bowtie r \).

Note that when attributes \( x \) and \( y \) are the same, the generalised difference is a difference operation, and the semijoin is an intersection operation.

**Lemma 8.2.2** In an expression when a common operation between two arguments \( r \) and \( s \) is difference and the labels of the input edges for the common operation node in the corresponding expression tree are \( A_x(r) \) and \( N_y(s)+ \), argument \( r \) can reduce argument \( s \). The reduction expression for \( s \) is \( s \bowtie r \) and for \( r \) there is no reduction expression.

**Proof:** All tuples of relational table \( r \) with \( x \) attributes flow to the operation for the edge labelled by \( A_x(r) \). No tuples of \( s \) flow to the operation for the other edge. However, there are tuples of other relational table(s), with \( y \) attributes, other than \( s \) that flow to the operation for this edge.

Let \( T_{N_y+s+} \) be all those tuples that flow to the common operation for the edge labelled by \( N_y(s)+ \). Thus, \( T_{N_y+s+} \cap \pi_y(s) = \emptyset \). Output of the difference operation consists of the tuples in \( \pi_x(r) \) that are not joinable with any tuples in \( T_{N_y+s+} \).

- Relation \( s \) can be reduced to \( s \bowtie r \). Assume that \( \beta \) is an operation whose output edge is labelled by \( N_y(s)+ \) (Figure 8.3).
Let \( t_j \in \pi_y(s) \). Assume \( t_j \) cannot be joined with any tuples of \( r \). If \( t_j \) has an impact on the output of the \( \beta \) operation such that a tuple \( t'_j \) does not appear in the output, then, by removing \( t_j \) from \( s \), tuple \( t'_j \) will appear in the output of \( \beta \). The tuple \( t'_j \) has no impact on the output of the difference operation because it appears in the output of \( \beta \) as the result of removing \( t_j \) from \( s \) and thus it cannot be joined with any tuples of \( r \). Hence, those tuples in \( s \) that cannot be joined with \( r \) can be removed. Thus, \( s \) can be reduced to \( s \bowtie r \).

- Relation \( r \) cannot be reduced with respect only to tuples of \( s \). If a tuple \( t_i \) in \( r \) is removed by \( s \) in a reduction operation, then tuple \( t_i \) may not be joinable with any tuples in \( T_{NVS+} \). It produces incorrect output for the common operation. Thus, \( r \) cannot be reduced with respect to tuples of \( s \).

**Lemma 8.2.3** In an expression when a common operation between two arguments \( r \) and \( s \) is difference and the labels of the input edges for the common operation node in the corresponding expression tree are \( A_x(r) \) and \( E_y(s) \), argument \( r \) can reduce argument \( s \). The reduction expression for \( s \) is \( s \bowtie r \), and for \( r \), there is no reduction expression.

**Proof:** All tuples of relational table \( r \) with \( x \) attributes flow to the operation for the edge labelled by \( A_x(r) \). Some tuples of \( s \) with \( y \) attributes flow to the operation for the other edge. Let \( s' \) be those tuples of \( s \) that flow to the operation for the edge labelled by \( E_y(s) \). Thus, \( s' \subseteq s \).

The evaluation result of the common operation consists of those tuples of \( r \) that are not joinable with any tuples in \( s' \).

- Relation \( s \) can be reduced to \( s \bowtie r \). Let \( t_i \in s' \). If \( \pi_y(t_i) \) cannot be joined with any tuples in \( r \), then \( t_i \) can be removed from \( s \) without any impact on the evaluation result of the common operation.

If tuple \( t_i \) in \( s \) but \( t_i \not\subseteq s' \), then \( t_i \) can be removed from \( s \) because it does not affect the evaluation result of the common operation. Thus, \( s \) can be reduced to \( s \bowtie r \).

- Relation \( r \) cannot be reduced because \( r \) can only be reduced by all tuples in \( s \). There exists a tuple \( t_i \) in \( s \) such that \( t_i \not\subseteq s' \). Suppose \( t_i \) can be joined with a tuple \( t_j \) in \( r \). The tuple \( t_j \) cannot be removed from \( r \) because it is joinable with tuple \( t_i \) which is in \( s \) and it is not in \( s' \). Thus, \( r \) cannot be reduced.
Lemma 8.2.4 In an expression when a common operation between two arguments \( r \) and \( s \) is difference and the labels of the input edges for the common operation node in the corresponding expression tree are \( A_x(r) \) and \( E_y(s) \), argument \( r \) can reduce argument \( s \). The reduction expression for \( s \) is \( s \times r \), and for \( r \), there is no reduction expression.

Proof: All tuples of relational table \( r \) with \( x \) attributes flow to the operation for the edge labelled by \( A_x(r) \). In addition to some tuples of \( s \), tuples of other relational table(s) other than \( s \), all with \( y \) attributes, flow to the operation for the other edge.

The proof is that same as that for labels \( A_x(r) \) and \( E_y(s) \). Note that the tuples that, in addition to tuples of \( s \), flow to the common operation for the edge labelled by \( E_y(s) \) do not create additional dimension for the proof.

Lemma 8.2.5 In an expression when a common operation between two arguments \( r \) and \( s \) is difference and the labels of the input edges for the common operation node in the corresponding expression tree are \( A_x(r) \) and \( A_y(s) \), argument \( s \) can reduce argument \( r \). The reduction expression for \( r \) is \( r - s \) and for \( s \), there is no reduction expression.

Proof: All tuples of relational table \( r \), in addition to tuples of other relational table(s), all with \( x \) attributes, flow to the operation for the edge labelled by \( A_x(r) \). All tuples of \( s \) with \( y \) attributes flow to the operation for the other edge.

Let \( T_{rx+} \) be tuples of other relation(s) except tuples of \( r \) that flow to the operation for the edge labelled by \( A_x(r) \). The evaluation result of the common operation consists of tuples in \( (\pi_x(r) \cup T_{rx+}) \) that are not joinable with any tuples in \( s \).

- Relation \( r \) can be reduced to \( r - s \).
  Let \( t_i \in \pi_x(r) \) and \( t_i \not\in T_{rx+} \). If \( t_i \) can be joined with any tuples in \( s \), then \( t_i \) can be removed from \( r \) without any impact on the evaluation result of the common operation. Let \( t_i \in \pi_x(r) \) and \( t_i \in T_{rx+} \). If \( t_i \) can be joined with any tuples in \( s \), then \( t_i \) can also be removed from \( r \) without any impact on the output, because \( t_i \in T_{rx+} \) is finally removed by the common operation. Thus, \( r \) can be reduced to \( r - s \).

- Tuples of \( s \) cannot be reduced with respect only to tuples of \( r \), because there exists a tuple \( t_j \) in \( s \) such that \( t_j \) cannot be joined with any tuples in \( r \), but \( t_j \) can be joined with a tuple such as \( t_k \) in \( T_{rx+} \). If tuple \( t_j \) is removed from \( s \), because it is not joinable with any tuples in \( r \), then, since \( t_j \) is joinable with a tuple in \( T_{rx+} \),...
removal of $t_j$ that can be joined with $t_k$ in $T_{rx+}$ is not correct. Thus, $s$ cannot be reduced.

**Lemma 8.2.6** In an expression when a common operation between two arguments $r$ and $s$ is difference and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)+$ and $A_y(s)+$, argument $s$ can reduce argument $r$. The reduction expression for $r$ is $r - s$, and for $s$, there is no reduction expression.

**Proof:** All tuples of relational table $r$, in addition to tuples of other relational table(s), all with $x$ attributes, flow to the operation for the edge labelled by $A_x(r)+$. Also, all tuples of $s$, in addition to tuples of other relational table(s), all with $y$ attributes, flow to the operation for the other edge.

Let $T_{rx+}$ be those tuples of other relation(s) except tuples of $r$ that flow to the operation for the edge labelled by $A_x(r)+$. Thus, $T_{rx+} \cup \pi_x(r)$ are all tuples that flow to the operation for the edge labelled by $A_x(r)+$. Also, let $T_{sy+}$ be those tuples of other relation(s) except $s$ that flow to the operation for the edge labelled by $A_y(s)+$. Thus, $T_{sy+} \cup \pi_y(s)$ are all tuples that flow to the operation for this edge. The evaluation result of the common operation consists of those tuples that are in $T_{rx+} \cup \pi_x(r)$ and they cannot be joined with any tuples in $T_{sy+} \cup \pi_y(s)$.

- Relation $r$ can be reduced to $r - s$.
  Let $t_i$ be a tuple in $r$ such that $\pi_x(t_i) \not\in T_{rx+}$. If $\pi_x(t_i)$ can be joined with a tuple in $\pi_y(s)$, then $t_i$ can be removed from $r$ without any impact on the evaluation result of the common operation. If $\pi_x(t_i)$ also be in $T_{rx+}$, then $t_i$ can still be removed from $r$ because tuple $\pi_x(t_i)$ in $T_{rx+}$ will be removed in the common operation. Thus, $r$ can be reduced to $r - s$.

- Tuples of $s$ cannot be reduced with respect to tuples of $r$ because there exists $t_i$ in $s$ such that $t_i$ cannot be joined with any tuples in $r$ but $t_i$ can be joined with a tuple in $T_{rx+}$. If tuple $t_i$ is removed from $s$ for the reason that $t_i$ is not joinable with any tuples in $r$, then, since $t_i$ can be joined with a tuple in $T_{rx+}$, the evaluation result of the common operation will not be correct. Thus, $s$ cannot be reduced.

**Lemma 8.2.7** In an expression when a common operation between two arguments $r$ and $s$ is difference and the labels of the input edges for the common operation node in
the corresponding expression tree are $A_x(r)^+\text{ and } N(s)^+$, neither $r$ nor $s$ can reduce each other.

**Proof:** All tuples of relational table $r$, in addition to tuples of other relational table(s), all with $x$ attributes, flow to the operation for the edge labelled by $A_x(r)^+$. Tuples of other relational table(s) except $s$ flow to the operation for the other edge.

Let $T_{rx^+}$ be those tuples of other relational table(s) except tuples of $r$ that flow to the operation for the edge labelled by $A_x(r)^+$. Thus, $T_{rx^+} \cup \pi_x(r)$ are all tuples that flow to the operation for the edge labelled by $A_x(r)^+$.

- Relation $r$ cannot be reduced with respect to $s$ because none of the tuples in $s$ flow to the common operation node.
- Nor can tuples of $s$ be reduced with respect to tuples in $r$, because for the edge labelled by $A_x(r)^+\text{ not only tuples in } r \text{ flow but also tuples in } T_{rx^+}$. Furthermore, there exists $t_i$ in $T_{rx^+}$ such that if $s$ is reduced with respect only to tuples of $r$, then a tuple in $s$ which is reduced by $r$ can be joined with tuple $t_i$. Thus, relation $s$ cannot be reduced.

**Lemma 8.2.8** *In an expression when a common operation between two arguments $r$ and $s$ is difference and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)^+\text{ and } E_y(s)$, neither $r$ nor $s$ can reduce each other.*

**Proof:** All tuples of relational table $r$, in addition to tuples of other relation(s), all with $x$ attributes, flow to the operation for the edge labelled by $A_x(r)^+$. For the other edge, some tuples of $s$ with $y$ attributes flow to the operation.

Let $T_{rx^+}$ be those tuples of other relational table(s) other than $r$ that flow to the operation for the edge labelled by $A_x(r)^+$. Thus, $T_{rx^+} \cup \pi_x(r)$ are all tuples that flow to the operation for the edge labelled by $A_x(r)^+$. Also, let $s' \subseteq s$ be those tuples of $s$ that flow to the common operation for the edge labelled by $E_y(s)$.

- Relation $r$ cannot be reduced. There exists a tuple $t_i$ in $s$ such that $t_i \notin s'$. If $r$ is reduced with respect only to tuples of $s$, then a tuple such as $t_i$ that is in $s$ but is not in $s'$ may have an impact on tuples of $r$ for the reduction. Since the tuple $t_i \notin s'$, $r$ cannot be reduced.
8.2. Difference Operation

- Tuples of $s$ cannot be reduced with respect only to tuples of $r$ because there exists a tuple $t_i$ in $s'$ such that $t_i$ may have an impact on tuples of $T_{r,z+s}$. However, tuple $t_i$ is removed from $s$ by tuples of $r$. It is not correct. Thus, $s$ cannot be reduced.

**Lemma 8.2.9** In an expression when a common operation between two arguments $r$ and $s$ is difference and the labels of the input edges for the common operation node in the corresponding expression tree are $A_r(r)+$ and $E_y(s)+$, neither $r$ nor $s$ can reduce each other.

**Proof:** All tuples of $r$, in addition to tuples of other relational table(s), all with $x$ attributes, flow to the operation for the edge labelled by $A_r(r)+$. Also some tuples of $s$, in addition to tuples of other relational table(s), all with $y$ attributes, flow to the operation for the other edge. Neither $r$ nor $s$ can be reduced.

The proof is that same as that for labels $A_x(r)+$ and $E_y(s)$.

**Lemma 8.2.10** In an expression when a common operation between two arguments $r$ and $s$ is difference and the labels of the input edges for the common operation node in the corresponding expression tree are $N_x(r)+$ and $A_y(s)$, either $r$ can reduce $s$ or $s$ can reduce $r$. The reduction expression is either $r - s$ for $r$ or $s - r$ for $s$.

**Proof:** Tuples of other relational table(s) except tuples of $r$ flow to the operation for the edge labelled by $N_x(r)+$. For the other edge of the operation, all tuples of $s$ with $y$ attributes flow. Either relation $r$ can be reduced to $r - s$ or relation $s$ can be reduced to $s - r$.

Let $T_{N_x r+s}$ be all those tuples of the other relational table(s) except tuples of $r$ that flow to the operation for the edge labelled by $N_x(r)+$.

- Let $t_i \in r$. If $t_i \notin (r - s)$, then $t_i$ can be joined with tuples in $s$. If by the reduction operation a tuple such as $t_i$ appears in $T_{N_x r+s}$, then, since $t_i$ can be joined with a tuple such as $t_j$ in $s$, tuple $t_i$ in $T_{N_x r+s}$ will be removed by $t_j$ in the common operation. Thus, $r$ can be reduced to $r - s$. However, $s$ must not be reduced because tuple $t_j$ in $s$ can remove tuple $t_i$ that appears in $T_{N_x r+s}$ as the result of the reduction operation. If $s$ is also reduced by $r$, then tuple $t_j$ will be removed from $s$ which will produce incorrect output for the common operation.

- Let $t_i \in s$. If $t_i \notin (s - r)$, then $t_i$ can be joined with a tuple such as $t_j$ in $r$. Since none of the tuples of $r$ flow to the common operation, removing a tuple such as $t_i$ from $s$ will not affect the evaluation result of the common operation. Thus, $s$ can
be reduced to \( s \leftarrow r \). However, \( r \) cannot be reduced. If \( t_j \) is removed from \( r \) by a reduction operation, then \( t_j \) may appear in \( T_{N_x+} \). Since \( t_i \) is already removed from \( s \), an incorrect output for the common operation will be produced.

**Lemma 8.2.11** In an expression when a common operation between two arguments \( r \) and \( s \) is difference and the labels of the input edges for the common operation node in the corresponding expression tree are \( N_x(r)+ \) and \( A_y(s)+ \), either \( r \) can reduce \( s \) or \( s \) can reduce \( r \). The reduction expression is either \( r \leftarrow s \) for \( r \) or \( s \leftarrow r \) for \( s \).

**Proof:** All tuples of other relational table(s) except tuples of \( r \) flow to the operation for the edge labelled by \( N_x(r)+ \). For the other input edge to the operation, all tuples of \( s \), in addition to tuples of other relational table(s), all with \( y \) attributes, flow.

The proof is the same as that for labels \( N_x(r)- \) and \( A_y(s) \). Note that tuples of other relational table(s) for the edge labelled by \( A_y(s)- \) that flow to the operation do not make an incorrect output if either \( r \) or \( s \) is reduced.

**Lemma 8.2.12** In an expression when a common operation between two arguments \( r \) and \( s \) is difference and the labels of the input edges for the common operation node in the corresponding expression tree are \( N_x(r)- \) and \( A_y(s)- \), argument \( r \) can reduce argument \( s \). The reduction expression is \( s \leftarrow r \) for \( s \). For \( r \), there is no reduction expression.

**Proof:** All tuples of other relational table(s) except tuples of \( r \) flow to the operation for the edge labelled by \( N_x(r)+ \). For the other input edge to the operation, tuples of other relational table(s) except tuples of \( s \) flow.

Let \( T_{N_x+} \) and \( T_{N_y+} \), in order, be those tuples of other relations other than \( r \) and \( s \) that flow to the operation.

- Relation \( s \) can be reduced to \( s \leftarrow r \). Let \( t_i \in s \), then \( \pi_y(t_i) \notin T_{N_y+} \). Assume that tuple \( t_i \) is removed from \( s \) in the reduction operation. The impact of removing \( t_i \) from \( s \) may be such that \( \pi_y(t_i) \) appears in \( T_{N_y+} \). If this happen, then, since there is no such a tuple in \( T_{N_x+} \) that can be joined with \( t_i \) because \( T_{N_x+} \) consists of tuples of other relational table(s) other than \( r \), removing \( t_i \) from \( s \) has no impact on the output of the common operation. Thus, \( s \) can be reduced to \( s \leftarrow r \).

- Relation \( r \) cannot be reduced. Let \( t_j \in r \), then \( \pi_x(t_j) \notin T_{N_x+} \). Assume that a tuple \( t_j \) is removed from \( r \) in a reduction operation. The impact of removing \( t_j \) from \( r \) may be such that \( \pi_x(t_j) \) appears in \( T_{N_x+} \). If this happen, then, since there
is no such a tuple in $T_{Nys+}$ that can be joined with $t_j$ because $T_{Nys+}$ consists of tuples of other relational table(s) other than $s$, removing $t_j$ from $r$ produces an incorrect output for the common operation. Thus, $r$ cannot be reduced.

**Lemma 8.2.13** In an expression when a common operation between two arguments $r$ and $s$ is difference and the labels of the input edges for the common operation node in the corresponding expression tree are $N_x(r)+$ and $E_y(s)$, neither $r$ nor $s$ can reduce each other.

**Proof:** All tuples of other relational table(s) except tuples of $r$ flow to the operation for the edge labelled by $N_x(r)+$. For the other input edge to the operation, some tuples of $s$ with $y$ attributes flow.

Let $T_{Nsr+}$ be tuples of other relational table(s) other than $r$ that flow to the operation for the edge labelled by $N_x(r)+$. Let $s'$ be some tuples of $s$ that flow to the operation for the other edge. Thus, $s' \subset s$.

- Relation $r$ cannot be reduced. Let $t_i \in r$. If tuple $t_i$ is removed by $s$ in a reduction operation, then the impact may be such that $\pi_x(t_i)$ appears in $T_{Nsr+}$. Assume that this happens. The tuple $\pi_x(t_i)$ may not be removed from $T_{Nsr+}$ because the corresponding tuple is in $s$ and may not be in $s'$. Thus, $r$ cannot be reduced.

- Relation $s$ cannot be reduced. Since no tuples of $r$ are in $T_{Nsr+}$, reducing $s$ by $r$ will produce an incorrect output for the common operation. Let $t_j \in s$. Assume that tuple $t_j$ is removed by $r$ in a reduction operation. Then, $t_j$ will not be in $s'$ any more to remove the the tuple(s) in $T_{Nsr+}$ in the common operation. Thus, $s$ cannot be reduced.

**Lemma 8.2.14** In an expression when a common operation between two arguments $r$ and $s$ is difference and the labels of the input edges for the common operation node in the corresponding expression tree are $N_x(r)+$ and $E_y(s)+$, neither $r$ nor $s$ can reduce each other.

**Proof:** Tuples of other relational table(s) except tuples of $r$ flow to the operation for the edge labelled by $N_x(r)+$. For the other input edge to the operation, some tuples of $s$, in addition to tuples of other relational table(s), all with $y$ attributes, flow.

The proof is the same as that for labels $N_x(r)+$, $E_y(s)$. Note that the tuples of other relational table(s) other than $s$ that flow to the operation do not produce an additional dimension for the proof.
**Lemma 8.2.15** In an expression when a common operation between two arguments $r$ and $s$ is difference and the labels of the input edges for the common operation node in the corresponding expression tree are $E_x(r)$ and $A_y(s)$, arguments $r$ and $s$ can reduce each other. The reduction expression for $r$ is $r - k s$ where $k = x \cap y$. For $s$, the reduction expression is $s \simeq r$.

**Proof:** Some tuples of relational table $r$ with $x$ attributes flow to the operation for the edge labelled by $E_x(r)$. For the other edge of the operation, all tuples of $s$ with $y$ attributes flow.

Let $r' \subseteq r$ be those tuples of $r$ that flow to the operation for the edge labelled by $E_x(r)$.

- Relation $r$ can be reduced to $r - k s$. Any tuple in $r$ that can be removed by $s$ in the reduction operation will not also be in $r'$. Since the output of the common operation consists of those tuples of $r'$ that are not joinable with tuples in $s$, if $r$ is already reduced by reduction operation $r - k s$, the output will still consist of those tuples in $r'$ that are not joinable with tuples in $s$. Thus, $r$ can be reduced to $r - k s$.

- Relation $s$ can be reduced to $s \simeq r$. Those tuples of $s$ have impacts on the common operation that are joinable with tuples in $\pi_x(r')$. Let $t \in s$. If $t \in s$ is removed from $s$ by reduction operation $s \simeq r$, then $t$ is not joinable with any tuples in $\pi_x(r)$. Thus, $t$ is not joinable with any tuples in $\pi_x(r')$. If $t$ is not removed from $s$ by the reduction operation, then $t$ in $s$ is a joinable tuple with a tuple such as $t_j$ in $\pi_x(r)$. If $t_j$ is not among tuples of $\pi_x(r')$, then $t$ in $s$ has no impact on the output of the common operation. If $t_j$ is in $\pi_x(r')$, then $t$ in $s$ will remove the corresponding tuple $t_j$ in $\pi_x(r')$ in the output of the common operation. Thus, $s$ can be reduced to $s \simeq r$.

**Lemma 8.2.16** In an expression when a common operation between two arguments $r$ and $s$ is difference and the labels of the input edges for the common operation node in the corresponding expression tree are $E_x(r)$ and $A_y(s)\uparrow$, arguments $r$ and $s$ can reduce each other. The reduction expression for $r$ is $r - k s$ and for $s$ the reduction expression is $s \simeq \pi_k(r)$, where $k = x \cap y$.

**Proof:** Some tuples of relational table $r$ with $x$ attributes flow to the operation for the edge labelled by $E_x(r)$. For the other input edge to the operation, all tuples of $s$, in addition to tuples of other relational table(s), all with $y$ attributes flow.
8.2. Difference Operation

Let \( r' \) be those tuples of \( r \) that flow to the operation for the edge labelled by \( E_x(r) \). Let \( T_{sy+} \) be tuples of other relational table(s) other than \( s \) that flow to the operation for the other edge.

- Relation \( r \) can be reduced to \( r \rightarrow_k s \). Output of the common operation consists of tuples of \( \pi_x(r') \) that are not joinable with tuples in \( \pi_y(s) \) and \( T_{sy+} \). Since all tuples in \( r' \) are also in \( r \), if relation \( r \) is reduced by the reduction operation to those tuples that are not joinable with any tuples in \( s \), then \( r' \) will consist of the tuples that are not joinable with any tuples in \( s \) as well. Thus, \( r \) can be reduced to \( r \rightarrow_k s \).

- Relation \( s \) can be reduced to \( s \times \pi_k(r) \). Let \( t_i \in s \). If tuple \( t_i \) is removed by the reduction operation from \( s \), then \( t_i \) is not in \( r \) and thus \( t_i \) is not also in \( r' \). If tuple \( t_i \) is not removed by the reduction operation, then \( t_i \) is a joinable tuple with a tuple such as \( t_j \) in \( r \). If tuple \( t_j \) is also in \( r' \), it will be removed by \( t_i \) in the common operation. Thus, \( s \) can be reduced to \( s \times \pi_k(r) \).

Lemma 8.2.17 In an expression when a common operation between two arguments \( r \) and \( s \) is difference and the labels of the input edges for the common operation node in the corresponding expression tree are \( E_x(r) \) and \( N_y(s)+ \), argument \( r \) can reduce argument \( s \). The reduction expression for \( s \) is \( s \times \pi_k(r) \), where \( k = x \cap y \). For \( r \), there is no reduction expression.

Proof: Some tuples of relational table \( r \) with \( x \) attributes flow to the operation for the edge labelled by \( E_x(r) \). For the other input edge to the operation, tuples of other relational table(s) except tuples of \( s \) flow.

Let \( r' \) be those tuples of \( r \) that flow to the common operation for the edge labelled by \( E_x(r) \). Thus, \( r' \subseteq r \). Let \( T_{Nys+} \) be the tuples that flow to the common operation for the edge labelled by \( N_y(s)+ \).

- Relation \( r \) cannot be reduced by \( s \), because no tuples of \( s \) flow to the common operation for the edge labelled by \( N_y(s)+ \).

- Relation \( s \) can be reduced to \( s \times \pi_k(r) \). Let \( t_i \in s \). If tuple \( t_i \) is removed by the reduction operation from \( s \), the impact may be such that tuple \( \pi_y(t_i) \) appears in \( T_{Nys+} \). Since tuple \( t_i \) is not a joinable tuple with any tuples in \( r \), it is not also joinable with any tuples in the subset of \( r \) which is \( r' \). Therefore, the appearance of the tuple in \( T_{Nys+} \) will not affect the output of the common operation.
Lemma 8.2.18 In an expression when a common operation between two arguments \( r \) and \( s \) is difference and the labels of the input edges for the common operation node in the corresponding expression tree are \( E_x(r) \) and \( E_y(s) \), argument \( r \) can reduce argument \( s \). The reduction expression for \( s \) is \( s \bowtie \pi_k(r) \), where \( k = x \cap y \). For \( r \), there is no reduction expression.

Proof: Some tuples of relational table \( r \) with \( x \) attributes flow to the operation for the edge labelled by \( E_x(r) \). For the other input edge to the operation, some tuples of \( s \) with \( y \) attributes flow.

Let \( r' \) and \( s' \) be the tuples that flow to the operation, in order, for the edge labelled by \( E_x(r) \) and \( E_y(s) \). Thus, \( r' \subseteq r \) and \( s' \subseteq s \).

- Relation \( r \) cannot be reduced by \( s \). Let \( t_i \in r' \). Assume that the tuple \( t_i \) is removed from \( r \) in a reduction operation by \( s \). Therefore, tuple \( t_i \) is joinable with tuples in \( s \) and after the reduction operation the tuple is not in \( r' \) any more. Tuple \( t_i \) is joinable with tuples in \( s \) and it may not be joinable with tuples in \( s' \). Thus, the reduction of \( r \) by \( s \) is not correct.

- Relation \( s \) can be reduced to \( s \bowtie \pi_k(r) \). Let \( t_i \in s' \). Thus, tuple \( t_i \) is also in \( s \). If tuple \( t_i \) is removed from \( s \) by the reduction operation, then tuple \( t_i \) is not joinable with any tuples in \( r \) and thus in \( r' \). If tuple \( t_i \) is not removed from \( s \) by the reduction operation, then it is joinable with a tuple such as \( t_j \) in \( r \). If tuple \( t_j \) is not in \( r' \), then it will not affect the output of the common operation. Moreover, those tuples of \( s \) have impacts on the common operation that are joinable with tuples of \( r' \). Since \( r' \subseteq r \), after the reduction operation, the reduced \( s \) still contains all the tuples that are joinable with \( r' \). Thus, \( s \) can be reduced to \( s \bowtie \pi_k(r) \).

Lemma 8.2.19 In an expression when a common operation between two arguments \( r \) and \( s \) is difference and the labels of the input edges for the common operation node in the corresponding expression tree are \( E_x(r) \) and \( E_y(s) \), argument \( r \) can reduce argument \( s \). The reduction expression for \( s \) is \( s \bowtie \pi_k(r) \). For \( r \), there is no reduction expression.

Proof: Some tuples of relational table \( r \) with \( x \) attributes flow to the operation for the edge labelled by \( E_x(r) \). For the other input edge to the operation, some tuples of \( s \), in addition to tuples of other relational table(s), all with \( y \) attributes, flow. Relation
8.2. Difference Operation

$s$ can be reduced to $s \prec \pi_k(r)$. The proof is the same as that for labels $E_x(r)$ and $E_y(s)$.

Lemma 8.2.20 In an expression when a common operation between two arguments $r$ and $s$ is difference and the labels of the input edges for the common operation node in the corresponding expression tree are $E_x(r)^+$ and $A_y(s)^+$, argument $r$ can reduce argument $s$. The reduction expression for $r$ is $r \triangleleft_k s$. For $s$, there is no reduction expression.

Proof: Some tuples of relational table $r$, in addition to tuples of other relational table(s), all with $x$ attributes, flow to the operation for the edge labelled by $E_x(r)^+$. For the other input edge to the operation, all tuples of $s$ with $y$ attributes flow.

Let $r'$ and $T_{rx^+}$, in order, be those tuples of $r$ and tuples of other relational table(s) other than $r$ that flow to the operation for the edge labelled by $E_x(r)^+$. Thus, $r' \subseteq r$.

- Relation $r$ can be reduced to $r \triangleleft_k s$. If relation $r$ is reduced to those tuples that are not joinable with tuples in $s$, then $r'$ is also reduced to the tuples that are not joinable with any tuples in $s$. Thus, $r$ can be reduced to $r \triangleleft_k s$.

- Relation $s$ cannot be reduced by $r$. If $s$ is reduced in a reduction operation by $r$, then there may be a tuple in $s$ that is removed in the reduction operation and this tuple may be a joinable tuple with tuples in $T_{rx^+}$. Thus, $s$ cannot be reduced.

Lemma 8.2.21 In an expression when a common operation between two arguments $r$ and $s$ is difference and the labels of the input edges for the common operation node in the corresponding expression tree are $E_x(r)^+$ and $A_y(s)^+$, argument $r$ can reduce argument $s$. The reduction expression for $r$ is $r \triangleleft_k s$, where $k = x \cap y$. For $s$, there is no reduction expression.

Proof: Some tuples of relational table $r$, in addition to tuples of other relational table(s), all with $x$ attributes, flow to the operation for the edge labelled by $E_x(r)^+$. For the other input edge to the operation, all tuples of $s$, in addition to tuples of other relational table(s), all with $y$ attributes, flow.

The proof is the same as that for labels $E_x(r)^+$ and $A_y(s)$.

Lemma 8.2.22 In an expression when a common operation between two arguments $r$ and $s$ is difference and the labels of the input edges for the common operation node in the corresponding expression tree are $E_x(r)^+$ and $N_y(s)^+$, neither $r$ nor $s$ can reduce each other.
8.2. Difference Operation

Proof: Some tuples of relational table $r$, in addition to tuples of other relational table(s), all with $x$ attributes, flow to the operation for the edge labelled by $E_x(r)+$. For the other input edge to the operation, tuples of other relational table(s) except tuples of $s$ flow.

Neither $r$ nor $s$ can be reduced, since none of the tuples of $s$ flow to the common operation.

Lemma 8.2.23 In an expression when a common operation between two arguments $r$ and $s$ is difference and the labels of the input edges for the common operation node in the corresponding expression tree are $E_x(r)+$ and $E_y(s)$, neither $r$ nor $s$ can reduce each other.

Proof: Some tuples of relational table $r$, in addition to tuples of other relational table(s), all with $x$ attributes, flow to the operation for the edge labelled by $E_x(r)+$. For the other input edge to the operation, some tuples of $s$ with $y$ attributes flow.

Let $r'$ and $T_{rx+}$, in order, be those tuples of $r$ and tuples of other relational table(s) that flow to the common operation for the edge labelled by $E_x(r)+$. Also, let $s'$ be those tuples of $s$ that flow to the operation for the other edge. Thus, $r' \subseteq r$ and $s' \subseteq s$.

- Relation $r$ cannot be reduced by $s$. Let $t_i \in r'$. Assume in a reduction operation, tuple $t_i$ is removed from $r$ by $s$. If tuple $t_i$ is joinable with tuples in $s'$, the reduction operation is correct. If it is in $s$ but it is not in $s'$, then the reduction operation is not correct. Thus, $r$ cannot be reduced.

- Relation $s$ cannot be reduced by $r$. If $s$ is reduced in a reduction operation by $r$, then there may be a tuple in $r$ such that it is not in $r'$ and that tuple reduces a tuple in $s'$. Thus, $s$ cannot be reduced by $r$.

Lemma 8.2.24 In an expressions when a common operation between two arguments $r$ and $s$ is difference and the labels of the input edges for the common operation node in the corresponding expression tree are $E_x(r)+$ and $E_y(s)+$, neither $r$ nor $s$ can reduce each other.

Proof: Some tuples of relational table $r$, in addition to tuples of other relational table(s), all with $x$ attributes, flow to the operation for the edge labelled by $E_x(r)+$. For the other input edge to the operation, some tuples of $s$, in addition to tuples of other relational table(s), all with $x$ attributes, flow.

The proof is the same as that for labels $E_x(r)+$, $E_y(s)$. 
8.3 Union Operation

Proof of reduction expressions for labels $A_x(r)$ and $A_y(s)$ of the input edges to union as the common operation has already been given. We will continue to present the proofs of reduction expressions for other labels.

Lemma 8.3.1 In an expression when a common operation between two arguments $r$ and $s$ is union and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)$ and $A_y(s)$, arguments $r$ and $s$ can reduce each other. The reduction expression for is either $r - s$ for $r$, or $s - r$ for $s$.

Proof: All tuples of $r$ with $x$ attributes, and all tuples of $s$ with $y$ attributes flow to the operation. In the output of the union operation duplicate tuples are removed. Thus, either tuples of $r$ can be reduced to the tuples that cannot be joined with tuples of $s$ or tuples of $s$ can be reduced to the tuples that cannot be joined with tuples of $r$.

Note that when attributes $x$ and $y$ are the same, the join operation is changed to an intersection operation.

Lemma 8.3.2 In an expression when a common operation between two arguments $r$ and $s$ is union and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)$ and $N_y(s)$, argument $r$ can reduce argument $s$. The reduction expression for $s$ is $s - r$. For $r$, there is no reduction expression.

Proof: None of the tuples of $s$ flow to the common operation because the input edge to the operation is labelled by $N_y(s)$. Tuples that flow to the operation are all tuples of $r$ with $x$ attributes for the edge labelled by $A_x(r)$ and tuples of other relational table(s) other than $s$ for the edge labelled by $N_y(s)$.

Let $T_{N_y}$ be the tuples for edge labelled by $N_y(s)$.

- Relation $r$ cannot be reduced by $s$. Since none of the tuples of $s$ flow to the operation, reducing relation $r$ by $s$ causes tuples of $r$ that may not be joinable with any tuples in $T_{N_y}$ to be removed from $r$. Thus, $r$ cannot be reduced.

- Relation $s$ can be reduced to $s - r$. Let $t_i \in s$. If tuple $t_i$ is removed by the reduction operation from $s$, then tuple $t_i$ is joinable with a tuple such as $t_j$ in $r$. If the impact of removing tuple $t_i$ from $s$ is such that a tuple such as $t_k$ appears in $T_{N_y}$, then both tuples $t_j$ and $t_k$, that are joinable tuples with each other, flow to the common operation for each edge. Since the common operation is union, one of the tuples is removed in the output of the common operation. Note that
tuple $t_k$ is a joinable tuple with $t_j$ because $t_k$ appears as the result that $t_j$ is joinable with $t_i$. Thus, $s$ can be reduced to $s - r$.

Lemma 8.3.3 In an expression when a common operation between two arguments $r$ and $s$ is union and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)$ and $E_y(s)$, argument $r$ can reduce argument $s$. The reduction expression for $s$ is $s - k r$, where $k = x \cap y$. For $r$, there is no reduction expression.

Proof: All tuples of relational table $r$ with $x$ attributes flow to the common operation for one edge and some tuples of $s$ with $y$ attributes flow to the common operation for the other input edge to the operation node.

Assume $s'$ are those tuples of $s$ that flow to the operation for the edge labelled by $E_y(s)$. Thus, $s' \subset s$.

- Relation $r$ cannot be reduced. Let $t_i \in r$. Relation $r$ can be reduced by $s$ with the assumption that if a tuple $t_i$ is removed from $r$ in a reduction operation, the tuple is joinable with a tuple $t_j$ in $s$. Since tuples in $s'$ are a subset of tuples in $s$, then tuple $t_j$ may not be in $s'$. Thus, $r$ cannot be reduced.

- Relation $s$ can be reduced to $s - k r$. Let $t_i \in s'$. If the tuple $t_i$ is removed by $r$ in the reduction operation from $s$, then $t_i$ is joinable with tuples in $r$. Therefore, removing a tuple such as $t_i$ from $s$ will not affect the evaluation result of the common operation. Thus, $s$ can be reduced to $s - k r$.

Lemma 8.3.4 In an expression when a common operation between two arguments $r$ and $s$ is union and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)$ and $E_y(s)+$, argument $r$ can reduce argument $s$. The reduction expression for $s$ is $s - k r$, where $k = x \cap y$. For $r$, there is no reduction expression.

Proof: All tuples of relational table $r$ with $x$ attributes flow to the common operation for one edge and some tuples of $s$, in addition to tuples of other relational table(s), all with $y$ attributes, flow to the common operation for the other input edge to the operation node.

The proof is the same as that for labels $A_x(r)$ and $E_y(s)$. 
Lemma 8.3.5 In an expression when a common operation between two arguments $r$ and $s$ is union and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)+$ and $A_y(s)+$, either $r$ can reduce $s$ or $s$ can reduce $r$. The reduction expression is either $r - s$ for $r$ or $s - r$ for $s$.

Proof: All tuples of relational table $r$, in addition to tuples of other relational table(s), all with $x$ attributes, flow to the common operation for one edge and all tuples of $s$, in addition to tuples of other relational table(s), all with $y$ attributes, flow to the common operation for the other input edge to the operation node.

Output of the common operation consists of all tuples of $r$, all tuples of $s$ and tuples of other relational table(s). Duplicate tuples between tuples of $r$ or $s$ can be removed. Thus, either $r$ can be reduced to $r - s$ or $s$ can be reduced to $s - r$.

Lemma 8.3.6 In an expression when a common operation between two arguments $r$ and $s$ is union and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)-$ and $A_y(s)-$, argument $r$ can reduce argument $s$. The reduction expression for $s$ is $s - r$. For $r$, there is no reduction expression.

Proof: All tuples of $r$, in addition to tuples of other relational table(s), all with $x$ attributes, flow to the common operation for one edge and tuples of other relational table(s) except tuples of $s$ flow to the common operation for the other input edge to the operation node.

Let $T_{Ny+s}$ be the tuples of other relational table(s) other than $s$ that flow to the common operation for the edge labelled by $N_y(s)+$.

- Relation $r$ cannot be reduced by $s$. If $r$ is reduced in a reduction operation by $s$, then there may be a tuple in $r$ that is removed by the reduction operation and that tuple is not in $T_{Ny+s}$. Thus, $r$ cannot be reduced.

- Relation $s$ can be reduced to $s - r$. Let $t_i \in s$. If tuple $t_i$ is removed from $s$ in the reduction operation, the impact may be such that tuple $\pi_y(t_i)$ appears in $T_{Ny+s}$. Since tuple $t_i$ is also in $r$, the appearance of it in $T_{Ny+s}$ will not affect the output of the common operation because in the operation the tuple will be removed. Thus, $s$ can be reduced to $s - r$.

Lemma 8.3.7 In an expression when a common operation between two arguments $r$ and $s$ is union and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)+$ and $E_y(s)$, argument $r$ can reduce argument
s. The reduction expression is for s is $s \bowtie_k r$, where $k = x \cap y$. For r, there is no reduction expression.

**Proof:** All tuples of r, in addition to tuples of other relational table(s), all with x attributes, flow to the common operation for one edge and some tuples of s with y attributes flow to the common operation for the other input edge to the operation node.

Let $s'$ be those tuples of s that flow to the operation for the edge labelled by $E_y(s)$. Thus, $s' \subseteq s$.

- Relation r cannot be reduced. Let $t_i \in r$. If tuple $t_i$ is removed by s in a reduction operation, then $t_i$ is joinable with a tuple such as $t_j$ in s. However, $t_j$ may not be in $s'$. The reduction operation is correct if, after reduction, $t_i$ or $t_j$ be input to the common operation. If $t_i$ is removed from r by $t_j$ in s and $t_j$ is not in $s'$, the evaluation result of the common operation will not be correct. Thus, r cannot be reduced.

- Relation s can be reduced to $s \bowtie_k r$. If the tuples in s are removed by the reduction operation, then s, after the reduction, consists of the joinable tuples with r. Thus, s can be reduced to $s \bowtie_k r$.

**Lemma 8.3.8** In an expression when a common operation between two arguments r and s is union and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)^+$ and $E_y(s)^+$, argument r can reduce argument s. The reduction expression for s is $s \bowtie_k r$, where $k = x \cap y$. For r, there is no reduction expression.

**Proof:** All tuples of relational table r, in addition to tuples of other relational table(s), all with x attributes, flow to the common operation for one edge and some tuples of s, in addition to tuples of other relational table(s), all with y attributes, flow to the common operation for the other input edge to the operation node.

The proof is the same as that for labels $A_x(r)^+$ and $E_y(s)$

**Lemma 8.3.9** In an expression when a common operation between two arguments r and s is union and the labels of the input edges for the common operation node in the corresponding expression tree are $N_x(r)^+$ and $N_y(s)^+$, neither r nor s can reduce each other.
Proof: None of the tuples of relational tables \( r \) and \( s \) flow on the common operation node for the input edges to the operation. However, for the both edges, tuples of other relational tables flow to the operation.

Neither \( r \) nor \( s \) can be reduced because none of their tuples flow to the operation. Furthermore, with the information that is expressed through the labels, no reduction operations are possible.

Lemma 8.3.10 In an expression when a common operation between two arguments \( r \) and \( s \) is union and the labels of the input edges for the common operation node in the corresponding expression tree are \( N_x(r)-\) and \( E_y(s) \), neither \( r \) nor \( s \) can reduce each other.

Proof: None of the tuples of relational tables \( r \) flow on the common operation for one edge but, for this edge, tuples of other relational table(s) with \( x \) attributes flow on the operation. For the other input edge to the operation, some tuples of \( s \) with \( y \) attributes flow to the operation.

Neither \( r \) nor \( s \) can be reduced because not only does none of the tuples in \( r \) flow to the common operation but a subset of tuples in \( s \) flow to the operation.

Lemma 8.3.11 In an expression when a common operation between two arguments \( r \) and \( s \) is union and the labels of the input edges for the common operation node in the corresponding expression tree are \( N_x(r)-\) and \( E_y(s)+ \), neither \( r \) nor \( s \) can reduce each other.

Proof: None of the tuples of relational table \( r \) except tuples of other relational table(s) with \( x \) attributes flow on the common operation for the edge labelled by \( N_x(r)-\). For the other input edge to the operation, some tuples of \( s \), in addition to tuples of other relational table(s), all with \( y \) attributes, flow on the operation.

The proof is the same as that for labels \( N_x(r)+ \) and \( E_y(s) \).

Lemma 8.3.12 In an expression when a common operation between two arguments \( r \) and \( s \) is union and the labels of the input edges for the common operation node in the corresponding expression tree are \( E_x(r) \) and \( E_y(s) \), neither \( r \) nor \( s \) can reduce each other.

Proof: Some tuples of relational table \( r \) with \( x \) attributes flow to the common operation for one edge and some tuples of \( s \) with \( y \) attributes flow to the common operation for the other input edge to the operation node.
Neither \( r \) nor \( s \) can be reduced. Let \( r' \) and \( s' \) be the tuples of \( r \) and \( s \) that flow to the common operation. Thus, \( r' \subseteq r \) and \( s' \subseteq s \). If \( r \) is reduced by \( s \), then there may be a tuple in \( s \) that is not in \( s' \) and it causes a tuple be removed from \( r \). If \( s \) is reduced by \( r \), then there may be a tuple in \( r \) that is not in \( r' \) and it removes a tuple from \( s \).

Thus, neither \( r \) nor \( s \) can be reduced.

**Lemma 8.3.13** In an expression when a common operation between two arguments \( r \) and \( s \) is union and the labels of the input edges for the common operation node in the corresponding expression tree are \( E_x(r) \) and \( E_y(s)+ \), neither \( r \) nor \( s \) can reduce each other.

**Proof:** Some tuples of relational table \( r \) with \( x \) attributes flow to the common operation for one edge and some tuples of \( s \), in addition to tuples of other relational table(s), all with \( y \) attributes flow to the common operation for the other input edge to the operation.

Neither \( r \) nor \( s \) can be reduced. The proof is the same as that for labels \( E_x(r) \) and \( E_y(s) \).

**Lemma 8.3.14** In an expression when a common operation between two arguments \( r \) and \( s \) is union and the labels of the input edges for the common operation node in the corresponding expression tree are \( E_x(r)+ \) and \( E_y(s)+ \), neither \( r \) nor \( s \) can reduce each other.

**Proof:** Some tuples of relational table \( r \), in addition to tuples of other relational table(s), all with \( x \) attributes, flow to the common operation for one edge and some tuples of \( s \), in addition to tuples of other relational table(s), all with \( y \) attributes, flow to the common operation for the other input edge to the operation node.

Neither \( r \) nor \( s \) can be reduced. The proof is the same as that for labels \( E_x(r) \) and \( E_y(s) \).

### 8.4 Intersection Operation

Proof of reduction expression for labels \( A_x(r) \) and \( A_y(s)+ \) of the input edges to an intersection as the common operation has already been given. We will continue to present the proofs of reduction expressions for other labels.

**Lemma 8.4.1** In an expression when a common operation between two arguments \( r \) and \( s \) is intersection and the labels of the input edges for the common operation node in
8.4. Intersection Operation

Figure 8.4: An expression tree in which the common operation for two arguments \( r \) and \( s \) is intersection and the labels of the input edges to the operation are \( A_x(r) \) and \( N_y(s) + \).

the corresponding expression tree are \( A_x(r) \) and \( A_y(s) \), arguments \( r \) and \( s \) can reduce each other. The reduction expression for \( r \) is \( r \times s \), and for \( s \), the reduction expression is \( s \times r \).

Proof: All tuples of \( r \) with \( x \) attributes, and all tuples of \( s \) with \( y \) attributes flow to the operation. Tuples of relation \( r \) can be reduced to the tuples that can be joined with \( s \) and tuples of relation \( s \) can be reduced to the tuples that can be joined with tuples of \( r \).

Lemma 8.4.2 In an expression when a common operation between two arguments \( r \) and \( s \) is intersection and the labels of the input edges for the common operation node in the corresponding expression tree are \( A_x(r) \) and \( N_y(s) + \), argument \( r \) can reduce argument \( s \). The reduction expression for \( s \) is \( s \times r \). For \( r \), there is no reduction expression.

Proof: None of the tuples of \( s \) flow to the common operation because the input edge to the operation is labelled by \( N_y(s) + \). Tuples that flow to the operation are all tuples of \( r \) with \( x \) attributes for the edge labelled by \( A_x(r) \) and tuples of other relational table(s) other than \( r \) with \( y \) attributes for the edge labelled by \( N_y(s) + \).

Let \( T_{Nys+} \) be the tuples that flow to the common operation for the edge labelled by \( N_y(s) + \). Also, let \( t_i \in s \). Then, \( \pi_y(t_i) \notin T_{Nys+} \). Assume \( \beta \) is an operation whose output edge is labelled by \( N_y(s) + \) (Figure 8.4) and which is not an intersection operation.

- Relation \( s \) can be reduced to \( s \times r \). Assume tuple \( \pi_y(t_i) \) cannot be joined with any tuples of \( r \). If removing \( t_i \) from \( s \) causes \( \pi_y(t_i) \) to appear in \( T_{Nys+} \) in the
output of $\beta$ operation, then, since tuple $\pi_y(t_i)$ cannot be joined with any tuples in $r$, it does not affect the output of the common operation.

If removing $t_i$ from $s$ has no impact on the output of $\beta$, then its removal has also no effect on the output of the common operation. Thus, $s$ can be reduced to $s \bowtie r$.

- Tuples of $r$ cannot be reduced by $s$ because if tuples of $r$ are reduced in a reduction operation, then there may be a tuple in $r$ that can be joined with $T_{NVR}$ and it is removed by the reduction operation from $r$. Thus, $r$ cannot be reduced.

**Lemma 8.4.3** In an expression when a common operation between two arguments $r$ and $s$ is intersection and the labels of the input edges for the common operation node in the corresponding expression tree are $Ax(r)$ and $Ey(s)$, arguments $r$ and $s$ can reduce each other. The reduction expression for $r$ is $r \bowtie \pi_k(s)$, where $k = x \cap y$. For $s$, the reduction expression is $s \bowtie r$.

**Proof:** All tuples of relational table $r$ with $x$ attributes flow to the common operation for one edge, and some tuples of $s$ with $y$ attributes flow to the common operation for the other input edge to the operation node.

Let $s'$ be those tuples of $s$ that flow to the operation. Thus, $s' \subset s$.

- Relation $r$ can be reduced to $r \bowtie \pi_k(s)$. If $r$ is reduced in the reduction operation by $s$, the reduced $r$ consists of those tuples that can be joined with tuples in $\pi_x(s)$. Since any tuples in $s'$ is also in $s$, then the reduced $r$ also contains those tuples that can be joined with tuples in $s'$.

- Relation $s$ can be reduced to $s \bowtie r$. If relation $s$ is reduced in the reduction operation, those tuples of $s$ that cannot be joined with tuples of $r$ are removed from $s$. Since $s' \subset s$, the tuples that are removed from $s$ will not be in $s'$.

**Lemma 8.4.4** In an expression when a common operation between two arguments $r$ and $s$ is intersection and the labels of the input edges for the common operation node in the corresponding expression tree are $Ax(r)$ and $Ey(s)^+$, argument $r$ can reduce argument $s$. The reduction expression for $s$ is $s \bowtie r$. For $r$, there is no reduction expression.
Proof: All tuples of relational table $r$ with $x$ attributes flow to the common operation for one edge and some tuples of $s$, in addition to tuples of other relational table(s), all with $y$ attributes, flow to the common operation for the other input edge to the operation node.

Let $T_{sy+}$ be tuples of other relational table(s) other than $s$ that flow to the operation. Also, let $s'$ be the tuples of $s$ that flow to the common operation. Thus, $s' \subseteq s$.

- Relation $r$ cannot be reduced by $s$. If $r$ is reduced by $s$ in a reduction operation, then there may be a tuple in $T_{sy+}$ such that it can be joined with the tuple that is removed from $r$ in the reduction operation. Thus, $r$ cannot be reduced.

- Relation $s$ can be reduced to $s \bowtie r$. In the reduction operation, those tuples of $s$ that cannot be joined with $r$ are removed. If $s$ is reduced by $r$ in the reduction operation, then the tuples that are removed from $s$ are not in $s'$ as well. Thus, $s$ can be reduced to $s \bowtie r$.

Lemma 8.4.5 In an expression when a common operation between two arguments $r$ and $s$ is intersection and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)+$ and $A_y(s)+$, neither $r$ nor $s$ can reduce each other.

Proof: All tuples of relational table $r$, in addition to tuples of other relational table(s), all with $x$ attributes, flow to the common operation for one edge, and all tuples of $s$, in addition to tuples of other relational table(s), all with $y$ attributes, flow to the common operation for the other input edge to the operation node.

Let $T_{rx+}$ and $T_{sy+}$, in order, be tuples of other relational table(s) except $r$ and $s$ that flow to the operation.

- Relation $r$ cannot be reduced by $s$. If $r$ is reduced by $s$, then there may be a tuple in $r$ that is already removed from $r$ because it cannot be joined with tuples of $s$. However, it can be joined with a tuple in $T_{sy+}$. Thus, $r$ cannot be reduced.

- Using the same approach as above, relation $s$ cannot be reduced by $r$.

Lemma 8.4.6 In an expression when a common operation between two arguments $r$ and $s$ is intersection and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)+$ and $N_y(s)+$, neither $r$ nor $s$ can reduce each other.
8.4. Intersection Operation

Proof: All tuples of relational table \( r \), in addition to tuples of other relational table(s), all with \( x \) attributes, flow to the common operation for one edge and tuples of other relational table(s) except tuples of \( s \) flow to the common operation for the other input edge to the operation node.

Let \( T_{rx+} \) be tuples of other relational table(s) other than \( r \) that flow on the common operation for the edge labelled by \( A_x(r) \). Let \( T_{Ny+} \) be tuples of other relational table(s) other than \( s \) that flow to the common operation for the other edge.

- Relation \( r \) cannot be reduced by \( s \) because if \( r \) is reduced in a reduction operation by \( s \), then there may be a tuple in \( r \) that can be joined with \( T_{Ny+} \). However, the tuple is removed by the reduction operation from \( r \). Thus, \( r \) cannot be reduced.
- Relation \( s \) cannot be reduced by \( r \) because a tuple \( t_i \) in \( s \) may be removed in a reduction operation from \( s \). The impact of removing \( t_i \) may be such that a new tuple appears in \( T_{Ny+} \) that can be joined with a tuple in \( T_{rx+} \). An incorrect output for the common operation will be produced. Thus, \( s \) cannot be reduced.

Lemma 8.4.7 In an expression when a common operation between two arguments \( r \) and \( s \) is intersection and the labels of the input edges for the common operation node in the corresponding expression tree are \( A_x(r) \) and \( E_y(s) \), argument \( s \) can reduce argument \( r \). The reduction expression for \( r \) is \( r \times \pi_k(s) \) where \( k = x \cap y \). For \( s \), there is no reduction expression.

Proof: All tuples of relational table \( r \), in addition to tuples of other relational table(s), all with \( x \) attributes, flow to the common operation for one edge and some tuples of \( s \) with \( y \) attributes flow to the common operation for the other input edge to the operation node.

Let \( s' \) be those tuples of \( s \) that flow to the operation. Thus, \( s' \subset s \). Also, let \( T_{rx+} \) be tuples of other relational table(s) other than \( r \) that flow to the operation for the edge labelled by \( A_x(r) \).

- Relation \( r \) can be reduced to \( r \times \pi_k(s) \). If tuple \( t_i \) in \( r \) cannot be joined with tuples in \( \pi_y(s) \), then tuple \( t_i \) cannot also be joined with tuples in \( \pi_y(s') \). Thus, \( r \) can be reduced to \( r \times \pi_k(s) \).
- Relation \( s \) cannot be reduced. If a tuple \( t_i \) in \( s \) cannot be joined with tuples of \( r \), then it cannot be removed from \( s \). The tuple \( t_i \) may be a tuple that can be joined with tuples of \( T_{rx+} \). Thus, \( s \) cannot be reduced.
Lemma 8.4.8 In an expression when a common operation between two arguments $r$ and $s$ is intersection and the labels of the input edges for the common operation node in the corresponding expression tree are $A_x(r)+$ and $E_y(s)+$, neither $r$ nor $s$ can reduce each other.

Proof: All tuples of relational table $r$, in addition to tuples of other relational table(s), all with $x$ attributes, flow to the common operation for one edge and some tuples of $s$, in addition to tuples of other relational table(s), all with $y$ attributes, flow to the common operation for the other input edge to the operation.

Neither $r$ nor $s$ can be reduced. Let $T_{rx+}$ and $T_{sy+}$, in order, be tuples of other relational table(s) other than $r$ and $s$ that flow to the operation for labels $A_x(r)-$ and $E_y(s)+$, respectively.

- Relation $r$ cannot be reduced by $s$. If tuples of $r$ are reduced by $s$, then there may be a tuple in $r$ that is already removed from $r$ because it cannot be joined with tuples of $s$. However, the tuple can be joined with a tuple in $T_{sy+}$. Thus, $r$ cannot be reduced.

- Using the same approach as above, relation $s$ cannot be reduced by $r$.

Lemma 8.4.9 In an expression when a common operation between two arguments $r$ and $s$ is intersection and the labels of the input edges for the common operation node in the corresponding expression tree are $N_x(r)+$ and $N_y(s)+$, neither $r$ nor $s$ can reduce each other.

Proof: None of the tuples of relational tables $r$ and $s$ flow on the common operation node for the input edges to the operation. However, for the both edges, tuples of other relational tables flow to the operation.

Neither $r$ nor $s$ can be reduced because none of their tuples flow to the operation. Furthermore, with the information that is expressed through the labels, no reduction operations are possible.

Lemma 8.4.10 In an expression when a common operation between two arguments $r$ and $s$ is intersection and the labels of the input edges for the common operation node in the corresponding expression tree are $N_x(r)+$ and $E_y(s)$, argument $s$ can reduce argument $r$. The reduction expression for $r$ is $r \times \pi_k(s)$, where $k = x \cap y$. For $s$, there is no reduction expression.
8.4. Intersection Operation

Figure 8.5: An expression tree in which the common operation for two arguments $r$ and $s$ is intersection and the labels of the input edges to the operation are $N_x(r)+$ and $E_y(s)$.

**Proof:** None of the tuples of relational table $r$ flow on the common operation. For one edge but, for this edge, tuples of other relational table(s) with $x$ attributes flow on the operation. For the other input edge to the operation, some tuples of $s$ with $y$ attributes flow to the operation.

Let $T_{N+x+}$ be all tuples that flow to the common operation for the edge labelled by $N_x(r)+$. Also, let $t_i \in r$. Then, $\pi_x(t_i) \notin T_{N+x+}$. Assume $\beta$ is an operation whose output edge is labelled by $N_x(r)+$ (Figure 8.5).

- Relation $r$ can be reduced to $r \bowtie \pi_k(s)$. Assume tuple $t_i$ in $r$ cannot be joined with any tuples of $s$. If removing tuple $t_i$ from $r$ causes that in the output of $\beta$ (i.e. in $T_{N+x+}$), a tuple $\pi_x(t_i)$ to appear, then, since $t_i$ cannot be joined with any tuples in $s$, its appearance has no impact on the output of the common operation. Thus, $r$ can be reduced to $r \bowtie \pi_k(s)$.

- Tuples of $s$ cannot be reduced with respect to tuples of $r$ because if tuples of $s$ are reduced to those tuples that can be joined with tuples of $r$, then there exists a tuple $t_j$ in $s$ that cannot be joined with any tuples of $r$. However, tuple $t_j$ can be joined with tuples in $T_{N+x+}$. Thus, for $s$, there is no reduction expression.

**Lemma 8.4.11** In an expression when a common operation between two arguments $r$ and $s$ is intersection and the labels of the input edges for the common operation node in the corresponding expression tree are $N_x(r)+$ and $E_y(s)+$, neither $r$ nor $s$ can reduce each other.
8.4. Intersection Operation

**Proof:** None of the tuples of relational table $r$ except tuples of other relational table(s) with $x$ attributes flow on the common operation for the edge labelled by $N_x(r)+$. For the other input edge to the operation, some tuples of $s$, in addition to tuples of other relational table(s), all with $y$ attributes, flow to the operation.

Since, there are tuples of other relational tables other than $r$ and also other than $s$ that flow to the operation for both edges, neither $r$ nor $s$ can be reduced. Thus, there are no reduction expressions for $r$ and $s$.

**Lemma 8.4.12** In an expression when a common operation between two arguments $r$ and $s$ is intersection and the labels of the input edges for the common operation node in the corresponding expression tree are $E_x(r)$ and $E_y(s)$, arguments $r$ and $s$ can reduce each other. The reduction expression for $r$ is $r \times \pi_k(s)$ and for $s$ the reduction expression is $s \times \pi_k(r)$, where $k = x \cap y$.

**Proof:** Some tuples of relational table $r$ with $x$ attributes flow to the common operation for one edge and some tuples of $s$ with $y$ attributes flow to the common operation for the other input edge to the operation.

Let $r'$ be those tuples of $r$ which flow to the common operation for the edge labelled by $E_x(r)$. Also, let $s'$ be those tuples of $s$ which flow to the operation for the edge labelled by $E_y(s)$. Thus, $r' \subseteq r$ and $s' \subseteq s$.

- Relation $r$ can be reduced to $r \times \pi_k(s)$. There exists a tuple $t_i$ in $r$, such that $\pi_x(t_i)$ cannot be joined with any tuples in $s$. Since, $s' \subseteq s$, then $\pi_x(t_i)$ cannot also be joined with any tuples in $\pi_y(s')$.

  If $\pi_x(t_i) \in \pi_x(r)$ but $\pi_x(t_i) \not\in \pi_x(r')$, then removing $t_i$ from $r$ does not affect the evaluation result of the intersection operation.

  If $\pi_x(t_i) \in \pi_x(r')$, then $t_i$ must be in $r$. Since $\pi_x(t_i)$ cannot be joined with any tuples in $\pi_y(s)$, then $t_i$ can be removed from $r$ without any impact on the evaluation result of the intersection operation. Thus, $r$ can be reduced to those tuples that can be joined with $s$. Hence, the reduction expression for $r$ is $r \times \pi_k(s)$.

- Using the same approach as above, $s$ can be reduced to $s \times \pi_k(s)$.

**Lemma 8.4.13** In an expression when a common operation between two arguments $r$ and $s$ is intersection and the labels of the input edges for the common operation node in the corresponding expression tree are $E_x(r)$ and $E_y(s)+$, argument $r$ can reduce argument $s$. The reduction expression for $s$ is $s \times \pi_k(r)$, where $k = x \cap y$. For $r$, there is no reduction expression.
Proof: Some tuples of relational table $r$ with $x$ attributes flow to the common operation for one edge and some tuples of $s$, in addition to tuples of other relational table(s), all with $y$ attributes, flow to the common operation for the other input edge to the operation node.

Let $r'$ be those tuples of $r$ which flow to the common operation for the edge labelled by $E_x(r)$. Let $s'$ be those tuples of $s$ that flow to the operation for the edge labelled by $E_y(s)+$. Also, assume that $T_{sv+}$ are tuples of other relations in addition to $\pi_x(s')$ that flow to the operation. Thus, $r' \subset r$ and $s' \subset s$.

- Relation $r$ cannot be reduced with regards to tuples of $s$. If tuples of $r$ are reduced by $s$, then there may be a tuple $t_i$ in $r$ that can be joined with a tuple $t_j$ in $T_{sv+}$. If tuple $t_i$ is removed in a reduction operation from $r$, then tuple $t_j$ cannot be joined with any tuples in $r$. Thus, $r$ cannot be reduced.

- Relation $s$ can be reduced to $s \bowtie \pi_k(r)$. There exists $t_i$ in $s$, such that $t_i$ cannot be joined with any tuples in $\pi_x(r)$. Since $r' \subset r$, then $\pi_y(t_i)$ cannot be joined with any tuples in $\pi_x(r')$. If $t_i \in s$ but $t_i \notin s'$, then removing $t_i$ from $s$ does not affect the evaluation result of the intersection operation.

Since $\pi_y(t_i)$ cannot be joined with any tuples in $\pi_x(r)$, if $\pi_y(t_i)$ is in $\pi_y(s)$, then $t_i$ can be removed from $s$ without any impact on the evaluation result of the intersection operation. Thus, $s$ can be reduced to those tuples that can be joined with $r$. Hence, the reduction expression for $s$ is $s \bowtie \pi_k(r)$.

Lemma 8.4.14 In an expression when a common operation between two arguments $r$ and $s$ is intersection and the labels of the input edges for the common operation node in the corresponding expression tree are $E_x(r)+$ and $E_y(s)+$, neither $r$ nor $s$ can reduce each other.

Proof: Some tuples of relational table $r$, in addition to tuples of other relational table(s), all with $x$ attributes, flow to the common operation for one edge and some tuples of $s$, in addition to tuples of other relational table(s), all with $y$ attributes, flow to the common operation for the other input edge to the operation node.

Let $T_{sv+}$ be tuples of other relational table(s) other than $s$ that flow to the operation for the edge labelled by $E_y(s)+$.

- Relation $r$ cannot be reduced. If $r$ is reduced with respect to tuples of $s$, then there may be a tuple such as $t_i$ in $r$ that, if it is removed, can be joined with
a tuple in $T_{xy+}$. However, tuple $t_i$ is already removed from $r$ in a reduction operation. Thus, $r$ cannot be reduced.

- Using the same approach as above, $s$ cannot be reduced by $r$
Bibliography


BIBLIOGRAPHY


BIBLIOGRAPHY


