Master Thesis

Tools and Techniques for Exploring Execution Model Relationships across Heterogeneous Stream Processing Engines

Author(s):
Kalay, Burak

Publication Date:
2011

Permanent Link:
https://doi.org/10.3929/ethz-a-006487708

Rights / License:
In Copyright - Non-Commercial Use Permitted
Master’s Thesis Nr. 13
Systems Group, Department of Computer Science, ETH Zurich

Tools and Techniques for Exploring Execution Model Relationships across Heterogeneous Stream Processing Engines

by
Omer Burak Kalay

Supervised by
Prof. Dr. Nesime Tatbul
Nihal Dindar

December 2010–June 2011
**Abstract**

Today, there is a diverse range of stream processing engines available for use. However, due to lack of standardization, they differ greatly in semantics, syntax and execution model which may lead differences in query results. SECRET model [1] is proposed to explain such behavioral differences. Yet, exploring relationships between heterogeneous stream processing engines remains as an important task.

This thesis investigates how SECRET can be used to explore execution model relationships between heterogeneous Stream Processing Engines. We define a methodology and propose a technique to predict relationships between any given engine configurations with high efficiency. We further show the validity of our technique through extensive experiments. We present design and architecture of a simulation and analysis software to serve as a multipurpose auxiliary tool in exploration of relationships. We also provide a prototype implementation of the proposed technique as part of this software.
Acknowledgements

I would like to thank Prof. Dr. Nesime Tatbul for her support and guidance during the course of this thesis. I would also like to thank Nihal Dindar for her patience, invaluable assistance and useful advices.

Finally, I would like to express my sincere thanks to my family for everything they have done.
# Contents

Abstract i

Acknowledgements ii

List of Figures vi

List of Tables vii

Abbreviations viii

1 Introduction 1

1.1 Background and Motivation 1

1.2 Problem Statement 2

1.3 Contribution 2

1.4 Overview 3

2 SECRET Model 4

2.1 Overview 4

2.2 Key Concepts 5

2.3 The Model 6

2.3.1 Scope 7

2.3.2 Content 8

2.3.3 Report 8

2.3.4 Tick 9

2.4 Relevance 12

3 Secret Simulation and Analysis Tool 14

3.1 Overview 14

3.2 Architecture 15

3.2.1 Scenario Generator 16

3.2.2 Input Generator 19

3.2.3 Input Analyzer 19

3.2.4 SECRET Engine Simulator 20

3.2.5 Result Analyzer 21

3.2.6 Table Maker 21
Contents

3.2.7 Table Reader ................................................. 22
3.2.8 Prediction Calculator ........................................ 22
3.2.9 Relation Analyzer .......................................... 22
3.2.10 Mismatch Analyzer ......................................... 22
3.2.11 Statistic Generator ......................................... 24

4 Exploration of Base Rules .......................................... 25
4.1 The Motive ...................................................... 25
4.2 What is a Base Rule Set? ....................................... 25
4.3 Notation ......................................................... 26
4.4 Setup and Methodology ....................................... 26
4.4.1 Comparison Methodology ................................ 26
4.4.2 Setup ....................................................... 27
4.5 Definitions ...................................................... 29
4.5.1 Set Relations ............................................... 29
4.5.2 General Concepts ......................................... 30
4.6 Base Rules ...................................................... 31

5 Relation Prediction Technique ...................................... 33
5.1 Notation ......................................................... 33
5.2 Overview ....................................................... 34
5.3 The Algorithm ................................................... 35
5.3.1 Simplification Phase ...................................... 36
5.3.2 Prediction Table ........................................... 40
5.3.3 Recursive Phase .......................................... 43
5.4 Limitations ...................................................... 45
5.5 RPT in Action .................................................... 47

6 Validation of Relation Prediction Technique ......................... 50
6.1 Setup and Methodology ....................................... 50
6.1.1 Setup ....................................................... 51
6.1.2 Expectations .............................................. 53
6.2 Results and Observations .................................... 54
6.2.1 Report Parameter Analysis ............................... 54
6.2.2 Cross Parameter Analysis ................................. 55
6.2.3 Observations .............................................. 55
6.3 Further Discussion ............................................. 57

7 Case Study .......................................................... 58
7.1 Overview ....................................................... 58
7.2 Setup .......................................................... 58
7.3 Results and Discussion ...................................... 59

8 Conclusion and Future Work .......................................... 64
8.1 Conclusion ...................................................... 64
8.2 Future Work .................................................... 65
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>SECRET of a query plan</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>Tick models</td>
<td>10</td>
</tr>
<tr>
<td>3.1</td>
<td>SSAT architecture and component interaction</td>
<td>15</td>
</tr>
<tr>
<td>5.1</td>
<td>Principle behind RPT</td>
<td>35</td>
</tr>
<tr>
<td>5.2</td>
<td>Low Precision Prediction case 1.</td>
<td>46</td>
</tr>
<tr>
<td>5.3</td>
<td>Low Precision Prediction case 2.</td>
<td>46</td>
</tr>
<tr>
<td>5.4</td>
<td>Ambiguous Prediction</td>
<td>47</td>
</tr>
<tr>
<td>5.5</td>
<td>Simplification phase of RPT</td>
<td>47</td>
</tr>
<tr>
<td>5.6</td>
<td>Recursive evaluation phase of RPT</td>
<td>48</td>
</tr>
<tr>
<td>7.1</td>
<td>Case Study. Prediction Tree</td>
<td>63</td>
</tr>
</tbody>
</table>
# List of Tables

5.1 Simplification Table .................................................. 38  
7.1 Case Study. Calculated Relation Set .............................. 60  
7.2 Case Study. Predicted Relation Set .............................. 60  
A.1 Common constructs used in base rule formulation ............... 66  
B.1 Prediction Table for RPT algorithm .............................. 83  
B.2 Prediction Table Legend ............................................ 83
Abbreviations

SPE  Stream Processing Engine
SSAT  Secret Simulation and Analysis Tool
RPT  Relation Prediction Technique
tud  Tuple_Driven
td  Time_Driven
bd  Batch_Driven
Chapter 1

Introduction

1.1 Background and Motivation

As continuous data sources are diversified with advancing technology, the importance of stream computing increases. Today, there are already many stream processing engines (SPEs) commercially available [2–4] and they are constantly improving. However, due to lack of standardization in stream processing, each SPE has its own semantics, syntax and even execution model which makes life harder both for users and developers.

Understanding this diversity in SPEs is of crucial importance in many aspects. For example, developers should have a clear understanding of the SPE which their applications work on or purchasers should be able to choose the SPE which cater for their requirements. SECRET model [1] is proposed to fulfil that need. It aims to identify, analyze and explain such behavioral differences on a broad range of different SPEs. It is a direct result of extensive analysis of carefully chosen set of real systems.

Finding how different stream processing engines relate to each other is an equally tempting challenge. Because once these relations are explored, they can be used to devise query rewriting rules. In a federation layer over different SPEs, these rules can provide foundation for query optimization [5]. Likewise, they can be used to devise query transformation rules so that queries can be semantically translated across SPEs which enables application portability and provides better query distribution. Additionally, query transformation and rewriting rules can be used in combination with a set of equivalences between SPEs as a foundation for decision mechanism which can choose between engines automatically in a federation layer.
1.2 Problem Statement

This thesis pursues the possibility of using SECRET model beyond its main purposes to discover relationships between different engines. Therefore, this work is seeking to answer following questions:

- Can SECRET model be used for exploring execution model relationships between engines? Is it capable of revealing such relations expressively?

- What kind of relationships should be discovered? How should they be defined? How are they related to each other?

- What methodology should be pursued? Which configurations and border cases should be taken into consideration? Which properties, patterns etc. should the input data have?

- Once discovered, how can relationships be expressed formally? Which parameter vectors should be included in formal expressions?

- Is there are any alternative method for exploration of mutual relationships to increase efficiency, speed etc.?

- Can patterns be found between specific engine configurations? If so, what is their use?

1.3 Contribution

This thesis describes a methodology for exploring relationships between heterogenous stream processing engines. It proposes a method, Relation Prediction Technique (RPT), to predict relationships between any SPEs based on known set of relationships between certain engine configurations. RPT reduces the need for simulation and analysis required in the process, hence achieves a high increase in efficiency. This thesis also presents the design and the architecture of Secret Simulation and Analysis Tool, an auxiliary software to compare SPEs and explore mutual relationships through simulation.

The tool is described and features of each software component are explained. The proposed technique and the development process is discussed in great detail. The derivation of rule sets, which RPT builds upon, is explained and formal base structures to be used in rule construction are defined. In addition, several rule sets are discovered and presented. This thesis also proves the validation of RPT through extensive testing over
complete available parameter space. A prototype implementation is also provided as part of the tool.

1.4 Overview

The rest of the thesis is organized as follows:

- Chapter 2 gives an overview of the SECRET model as background information.
- Chapter 3 presents architecture and features of the Secret Simulation and Analysis Tool, an auxiliary software developed for analysis and testing.
- Chapter 4 explains base rule sets and presents the detailed methodology of exploration.
- Chapter 5 explains the Relation Prediction Technique, a method we propose for exploring relationship between different engine configurations.
- Chapter 6 describes the methodology of validation of the Relation Prediction Technique and discusses results.
- Chapter 7 presents a case study where relationships between two known SPE configurations are explored.
- Chapter 8 covers conclusion and future work.
Chapter 2

SECRET Model

In this chapter we will briefly describe SECRET, a model for analysis of the execution semantics of stream processing systems. An overview is provided in Section 2.1, fundamental concepts are explained in Section 2.2, parameters of the model are explained in detail in Section 2.3. Finally, the relevance of the SECRET model to our work is given in Section 2.4.

This chapter summarizes the work presented in [1].

2.1 Overview

Today, there is a diverse range of stream processing engines (SPEs) available for use. However, they differ in execution model, semantics and syntax owing to lack of standardization in stream processing. Such variation may lead hard to analyze differences in query results and issues with application development and portability. SECRET is a descriptive model that is proposed to analyze and explain such behavioral differences for window-based queries on a broad range of heterogeneous SPEs.

The heterogeneity in question reveals itself in three categories:

**Syntax Heterogeneity**  This type of heterogeneity refers to the differences in language structures such as keywords used by the SPE for defining common constructs.

**Capability Heterogeneity**  This type of heterogeneity refers to the differences in types of queries supported by different SPEs. It also intervenes with syntax heterogeneity since some engines may provide functionality through language clauses that are not offered by others.
Execution Model Heterogeneity  This type of heterogeneity refers to the differences in the underlaying query execution model of SPEs. It is regarded as the most subtle type of heterogeneity since it is closed to outside effect. For this reason, SECRET focuses on analyzing execution semantics.

SECRET focuses on execution model heterogeneity for time-based windows and single-input query plans. It captures query execution semantics in four orthogonal but complementary dimensions after which it is also named: \textbf{ScopE}, \textbf{Content}, \textbf{REport} and \textbf{Tick}. Each of them is designed to reflect a certain aspect of window-based execution so that an overall end-to-end view over execution semantics can be provided.

2.2 Key Concepts

Several fundamental concepts of SECRET are defined as follows:

\textbf{Time Domain}  The time domain $T$ is a discrete, linearly ordered, countably infinite set of time instants $t \in T$. It is assumed to be bounded in the past, but not necessarily in the future.

\textbf{Stream}  A stream $S$ is a countably infinite set of elements $s \in S$. Each stream element $s : (v, t^{app}, t^{sys}, bid)$ consists of a relational tuple $v$ conforming to a schema $S$, with an application time value $t^{app} \in T$, a system time value $t^{sys} \in T$, and a batch-id value $bid \in \mathbb{N}$.

The two different time notations mentioned above serve different needs. \textit{System time}, denoted by $t^{sys}$, refers to the time information associated with the occurrence of a certain system event, such as arrival of a stream element. It is unique and used as the basis for decisions about tuple arrival events and corresponding system reactions. On the other hand, \textit{application time}, denoted by $t^{app}$, refers to the time information associated with occurrence of an application event represented by the stream element. Multiple elements in a stream can share the same $t^{app}$ value. $t^{app}$ values are used as a basis for query execution over the stream.

\textbf{Batch}  A batch $B$ of stream elements for a given stream $S$ is a finite subset of $S$, where all $b \in B$ have an identical $t^{app}$. Each such batch is given a unique batch-id $bid \in \mathbb{N}$ such that, for all $b \in B$, $b.bid = bid$, indicating that $b$ belongs to the batch that is uniquely identified by $bid$. For tuples $t_1$ and $t_2$ where $t_1.t^{sys} < t_2.t^{sys}$, then $t_1.bid \leq t_2.bid$.

Streams are totally ordered by $t^{sys}$ values of their elements and partially ordered by their $t^{app}$ values. Batches provide further ordering among simultaneous tuples [6]. Notice that,
while all tuples in a given batch share the same $t^{app}$ value, it does not mean that all tuples with the same $t^{app}$ value are in the same batch.

**Window**  A window $W$ over a stream $S$ is a finite subset of $S$.

Window definition can be specialized in various ways. However, SECRET particularly concentrates on *time-based windows* which are based on stream elements whose $t^{app}$ values fall into a certain interval. Formal definition is as follows:

**Time-based Window**  A *time-based window* $W = (o, c]$ over a stream $S$ is a finite subset of $S$ containing all data elements $s \in S$ where $o < s.t^{app} \leq c$.

Windows usually have specific relationships with each other defined by the following parameters;

**Window Size and Slide**  The set $\mathcal{W}$ of all time-based windows defined over a stream $S$ must satisfy the following two constraints:

1. Size($\omega$): All windows must be the same size, that is, $\forall W = (o, c] \in \mathcal{W}, c - o = \omega$.

2. Slide($\beta$): The distance between consecutive windows must be the same. For two windows $W_1 = (o_1, c_1]$ and $W_2 = (o_2, c_2]$, we require that $o_1 \neq o_2$. Furthermore, we say $W_1$ and $W_2$ are consecutive if $o_1 < o_2$ and there is no window $W' = (o', c')$ such that $o_1 < o' < o_2$. For all consecutive windows $W_1$ and $W_2$ in $\mathcal{W}$, we require that $o_2 - o_1 = \beta$.

At $t^{app} = t$, we say a window $W = (o, c]$ is *open*, if $o < t \leq c$. A window is *closed*, if $c < t$.

### 2.3 The Model

SECRET consists of four different parameters: *Scope* provides information about window intervals, *Content* maps specified window intervals to actual window contents, *Report* states the conditions under which window content becomes visible for processing and lastly *Tick* models under which conditions the SPE will react to given input stream.

Tick is the entry point which triggers a chain reaction by invoking Report, which in turn invokes Content, which builds up on Scope. SECRET is compositional in terms of its parameters in a similar way that a query plan is compositional in terms of sequence of operators.

Figure 2.1 illustrates the usage of SECRET in explaining the semantics of a query plan. We will now briefly explain each parameter.
2.3.1 Scope

Scope maps an application time value \( t \) to an interval over which query \( q \) should be evaluated. It is defined as follows:

\[
Scope(m) = \begin{cases} 
\emptyset & \text{if } t < t_0 \\
(o_n, t] & \text{otherwise}
\end{cases}
\]

In the above formulation, \( t_0 \in \mathbb{T} \) denotes the application time instant when the initial window \( (W_0) \) starts. This value is system-specific such that different systems are likely to have different start times. If the initial window \( (W_0) \) starts at \( t_0 \), then the very next window \( (W_1) \) starts at \( t_0 + \beta \). Hence, \( n^{th} \) window \( W_n \) starts at \( o_n = t_0 + n\beta \).

Index of the earliest open window \( n \), is calculated as follows:

\[
n = \max(0, \lceil \frac{t - t_0 - \omega}{\beta} \rceil)
\]

Several important points to note about Scope are as follows:

- The scope of window for given application time point depends on \( t_0 \) and window parameters of the query (size and slide) only. Scope does not depend on the input stream or any system specific behavior in any way.

- The Scope formulation focuses on the time interval for the earliest open window, although scope can be defined in various other ways as well.

- Windows can be constructed in two different ways depending on the interpretation of window slide value. If every new slide signals the end of a window which started \( \omega \) time units ago, then the construction is said to be in the backward direction [7, 8]. If every new slide signals the beginning of a new window, then the construction is said to be in the forward direction [3, 9]. The window scopes produced for these two different approaches differ only by a fixed amount \( \delta \). Therefore they can be
Chapter 2. SECRET Model

switched by calibrating $t_0$ by the amount of $\delta$. The Scope formulation uses the forward interpretation of slide, nevertheless both behavior can be produced by adjusting $t_0$ as explained.

2.3.2 Content

*Content* maps the application time interval representation of a window, which is defined by *Scope*, to a set of data elements. Thus, it depends on actual contents of the input stream which can only be known at real-time.

Given the application time instant $t$ and system time instant $\tau$, *Content* is defined as follows:

$$Content(t, \tau) = \{ s \in S : s.t_{app} \in Scope(t) \land s.t_{sys} < \tau \}$$

2.3.3 Report

*Report* defines the conditions under which window contents become visible for processing. Different reporting strategies exists. SECRET uses four of them:

1. **Content change** ($R_{cc}$): reporting is done for $t$ only if the content has changed since $t - 1$.

2. **Window close** ($R_{wc}$): reporting is done for $t$ only when the active window closes (i.e., $|Scope(t)| = \omega$).

3. **Non-empty content** ($R_{ne}$): reporting is done for $t$ only if the content at $t$ is not empty.

4. **Periodic** ($R_{pr}$): reporting is done for $t$ only if it is a multiple of $\lambda$, where $\lambda$ denotes the reporting frequency.

Different strategies can be combined. Thus, they are represented as four boolean variables ($R_{cc}, R_{wc}, R_{ne}, R_{pr}$) by SECRET. Note that, even if all variables set to false, $Content(t, \tau)$ will still be returned by the *Report* as the default behavior.
Formal definition of Report is as follows:

\[
Report(t, \tau) = \begin{cases} 
\text{Content}(t, \tau) & \text{if } (\neg R_{cc} \lor \text{Content}(t, \tau) \neq \\
\text{Content}(t - 1, \tau)) \\
\land (\neg R_{wc} \lor (|\text{Scope}(t)| = \omega \land \\
t < \max\{s.t_{app} | s \in S \land s.t_{sys} \leq \tau\}) \\
\land (\neg R_{ne} \lor \text{Content}(t, \tau) \neq \emptyset) \\
\land (\neg R_{pr} \lor \text{mod}(t, \lambda) = 0) \\
\emptyset & \text{otherwise}
\end{cases}
\]

### 2.3.4 Tick

Tick defines the condition that triggers a SPE to take action on its input. SECRET identifies three different tick strategies:

1. **Tuple driven**: Arrival of each tuple triggers a system reaction.
2. **Time driven**: Progress of \(t_{app}\) triggers a system reaction.
3. **Batch driven**: Arrival of a new batch or the progress of \(t_{app}\) triggers a system reaction.

Figure 2.2 depicts different tick models. Tuple arrivals are shown on the time line for \(t_{sys}\), and window scopes are shown underneath, on the time line for \(t_{app}\). Circles around the tuples show the units of tuples that the system will react to at one time, whereas the arrows show to which application time instant those units belong.

Note that all three events (new tuple arrival, application time progress and new batch arrival) that the Tick relies on are actually based on the detection of arrival of a new tuple, since all information needed for resolving those events is carried in tuples. Tuple arrivals can only be detected through system time instants.

At every tick, SECRET checks the reporting condition. However, while Report is modeled on application time units, Tick relies on system time units as mentioned above. Thus a mapping between two time notions is required. Furthermore, input stream might contain irregularities, such as simultaneous tuples (multiple tuples with the same \(t_{app}\)) or gaps (absence of tuples at certain \(t_{app}\)). Both cases require accurate detection. Simultaneous tuples can be detected by comparing current \(t_{app}\) with the previous tick.
time, while gaps can be detected by invoking \textit{Report} on all application time instants between current $t_{app}$ and previous tick time.

In lights of the above, SECRET makes following definitions:

$S(\tau)$ denotes the set of tuples in stream $S$ that has arrived through time instant $\tau$.

$$S(\tau) = \{s \in S | s.t^{sys} \leq \tau\}$$

$S_I(\tau)$ denotes the set of tuples in stream $S$ that has arrived at time instant $\tau$. There can be at most one such tuple.

$$S_I(\tau) = \{s \in S | s.t^{sys} = \tau\}$$

\textit{app}(\tau): Given a system time instant $\tau$, returns the application time value of the tuple that has arrived at $\tau$.

$$\text{app}(\tau) = \{s.t^{app} | s \in S_I(\tau) \land S_I(\tau) \neq \emptyset\}$$

\textit{prev.app}(\tau): Given a system time instant $\tau$, returns the application time value of the most recent tuple that has arrived before $\tau$. If no such tuple exists, it returns $t_0$.

$$\text{prev.app}(\tau) = \max(\max\{t_0, s.t^{app} | s \in S(\tau - 1)\})$$

\textit{batch}(\tau): Given a system time instant $\tau$, returns the batch-id value of the tuple that has arrived at $\tau$. 
Chapter 2. SECRET Model

\[
\text{batch}(\tau) = \max\{s.\text{bid}|s \in S_I(\tau)\}
\]

prev\_batch(\tau): Given a system time instant \(\tau\), returns the batch-id value of the most recent tuple that has arrived before \(\tau\). If no such tuple exists, it returns 0.

\[
\text{prev\_batch}(\tau) = \max(0, \max\{s.\text{bid}|s \in S(\tau - 1)\})
\]

prev\_tick(\tau): Given a system time instant \(\tau\), returns the application time value of the most recent tuple that has arrived before \(\tau\) for which the result of the tick was non-empty. If no such tuple exists, it returns \(t_0\).

\[
\text{prev\_tick}(\tau) = \{\max(t_0, \max(x| x < \tau \land \text{Tick}(x) \neq \emptyset))\}
\]

Based on the above, \(\text{Tick}\) is defined as follows in various systems:

In a tuple-driven system, \(\text{Tick}\) is triggered (a) if a tuple arrives whose \(t_{\text{app}}\) is the same as the previous tick time so that simultaneous tuples can be reacted or (b) if a tuple arrives whose \(t_{\text{app}}\) is greater than the previous tick time so that system can react accurately even though there are gaps in between tuples. Formal definition is as follows:

\[
\text{Tick}(\tau) = \begin{cases} 
\{\text{Report}(\text{app}(\tau), \tau)\} & \text{if } S_I(\tau) \neq \emptyset \land \\
\bigcup_{x<\text{app}(\tau)} \text{Report}(x, \tau) & \text{if } S_I(\tau) \neq \emptyset \land \\
\text{app}(\tau) > \text{prev\_tick}(\tau) & \text{otherwise}
\end{cases}
\]

In a time-driven system, first condition becomes obsolete since simultaneous tuples do not need to be reacted separately. Formal definition is as follows:

\[
\text{Tick}(\tau) = \begin{cases} 
\bigcup_{x<\text{app}(\tau)} \text{Report}(x, \tau) & \text{if } S_I(\tau) \neq \emptyset \land \\
\text{app}(\tau) > \text{prev\_app}(\tau) & \text{otherwise}
\end{cases}
\]

A batch-driven system acts like a modified tuple-driven system. Therefore, simultaneous tuples as well as tuples with new \(t_{\text{app}}\) should be checked. Additionally, initiations of new
batches should be checked through batch-id’s of incoming tuples. The formal definition is as follows:

\[
\text{Tick}(\tau) = \begin{cases} 
\{\text{Report}(\text{app}(\tau), \tau)\} & \text{if } S_I(\tau) \neq \emptyset \wedge \text{prev_tick}(\tau) = \text{app}(\tau) \wedge \text{batch}(\tau) > \text{prev_batch}(\tau) \\
\bigcup_{x=\text{prev_tick}(\tau)} x < \text{app}(\tau) \text{ Report}(x, \tau) & \text{if } S_I(\tau) \neq \emptyset \wedge \text{batch}(\tau) > \text{prev_batch}(\tau) \\
\emptyset & \text{otherwise}
\end{cases}
\]

2.4 Relevance

In order to explain differences between execution semantics of SPEs, the SECRET model decouples concerns in three different levels and treats them separately; data-level issues, system-level issues and query-level issues. Data-level issues are handled by Content parameter which is characterized by input properties such as simultaneity in tuple distribution or gap values between tuples of the stream. System level issues are handled by Report and Tick parameters which capture operational effects of query processing. Finally, query-level issues are handled by Scope which is characterized by query’s window parameters as well as \( t_0 \) of the system.

In this work, we aim to explore execution model relationships between heterogeneous SPEs using the SECRET model. Therefore we are interested in the system parameters (Report, Tick and \( t_0 \)) since they explain underlying execution model of a SPE. In this sense, we are specifically interested in how different Report and Tick parameter values across opposing configurations affect the relationships. This draws the boundaries of the parameter space we are interested in. We keep everything else, namely \( t_0 \), query’s window parameters and input parameters, the same across opposing engine configurations.

For each Report and Tick combination match-up, we explore the relationships for a set of query parameters. Likewise, for every association of system parameter match-up and query parameters, we explore the relationships for a set of input streams so that relationships between regarding engine configurations could be revealed expressively.

In this work, we will use the terms \textit{query’s window parameters} and \textit{window parameters} interchangeably to refer query’s window parameters. When we say \textit{engine configurations},
we mean different points in our parameter space which we described above. An engine configuration consists of system parameters and query parameters.
Chapter 3

Secret Simulation and Analysis Tool

In this chapter we will explain Secret Simulation and Analysis Tool (SSAT), a reporting and analysis software built upon SECRET model, which we developed and used extensively in our work. An overview is provided in Section 3.1, an overall system architecture along with component functionalities is provided in Section 3.2.

3.1 Overview

Secret Simulation and Analysis Tool is a simulation and automatic testing software which provides extensive analysis capabilities. It is developed in a modular fashion upon SECRET core which contains an implementation of the model we briefly presented in Chapter 2. One of the main capabilities of the software is to simulate a SPE using the SECRET model. The simulation is able to produce output in response to given query and input stream. Moreover, SSAT provides means of further analysis to compare results of simulations so that relationships between corresponding engine configurations can be explored. Several important features of SSAT are; generation of execution scenarios, automatic data generation, input analysis, result analysis, result prediction, statistics and mismatch analysis between real and predicted results.

SSAT served for different needs in our work. First, we used it for exploring base relationships across heterogeneous SPEs upon which we developed our prediction technique (Chapter 5). Then we implemented a prototype for our technique and used the tool for validation over all available search space.
3.2 Architecture

We used a modular design scheme in Secret Simulation and Analysis Tool such that it consists of several components. Each component is independent from the others and specializes on a certain task. Moreover, they can be composed to implement more complex features. This design scheme provided us a highly flexible and extensible structure.

Figure 3.1 shows components of SSAT and depicts interaction between them.

![SSAT architecture and component interaction](image)

Figure 3.1: SSAT architecture and component interaction

Execution starts with generation of simulation scenarios by Scenario Generator. Each scenario associates set of query parameters and input streams to system parameter match-ups. Each [system parameter match-up, window parameters, input file] association defines a comparison to be evaluated which reveals relationship between opposing
sides of system parameter match-up. Therefore, simulation scenarios are collections of related comparisons. At this stage input streams are defined by tuple distribution patterns. Input Generator generates actual contents according to these patterns. Once real contents are generated, they are analyzed by Input Analyzer to reveal hidden patterns in tuple distributions. SECRET Engine Simulator is responsible for simulating engine configurations to be compared, which are given by execution scenarios. It produces results for opposing sides. These execution results are analyzed by Result Analyzer and the relation between them is calculated.

Alternatively, evaluation results of comparisons can be predicted without simulation by using Prediction Calculator. It takes execution scenarios as its input and calculates a prediction for every comparison defined in them through Relation Analyzer. Relation Analyzer is the prototype implementation of the prediction technique we describe in Chapter 5. Prediction Calculator produces a set of predictions for each simulation scenario. Then, Mismatch Analyzer compares this predicted relation set with the relation set calculated by Result Analyzer to detect mismatches between them. Results of the analysis are evaluated by Statistics Calculator and a set of statistics are generated.

Next, we will describe each component in detail.

### 3.2.1 Scenario Generator

**Function** The Scenario Generator is responsible for generating execution scenarios. Execution Scenarios are sets of mappings from engine’s system parameters (Report, Tick and $t_0$) to set of window parameters (size and slide) and input files. They describe a complete simulation of corresponding configurations and produce a set of relations.

At this stage generated execution scenarios do not contain input data files but empty skeletons specifying how regarding data stream should be generated. Content is created by Input Generator through these skeletons later. Most of the time, input data stream skeletons follow a predefined pattern that defines how tuples in the stream will be distributed over application time line so that a specific case can be simulated and tested.

**Features** Several important features of the Scenario Generator are listed below:

- It is possible to control tuple distribution of input data streams over application time line. For instance, it is possible to generate a skeleton for an input data stream and say all tuples of the stream should arrive at window closes and nowhere else or no tuple should arrive at time instants where periodic reporting occurs.
• It is possible to adjust periodic reporting points to special application time instants. For instance, it is possible adjust the application time line so that periodic reporting happens only on window closes or on every second tuple expiration point or in between every expiration point as well as the points themselves. We call this feature as **Density control**. Following variations are possible: (i)tuples arrive in between alignment points as well as at alignment points, (ii)tuples arrive at only alignment points and nowhere else (iii)tuples arrive at only every \( n^{th} \) alignment point and nowhere else.

• It is possible to control whether generated input data stream should contain simultaneous tuples (multiple tuples with the same \( t^{\text{app}} \)) or simultaneous batches (multiple batches with the same \( t^{\text{app}} \) but different \( bid \)) through the skeleton of the corresponding input data stream.

• The Scenario Generator generates five different window configurations with following patterns:

1. \( \beta = 1 \) AND \( \omega = 1 \)
2. \( \beta = 1 \) AND \( \omega \neq 1 \). \( \omega \) is randomly generated from the interval \([1, \text{maxwindowsize}]\)
3. \( \beta = \omega \). \( \omega \) is randomly generated from the interval \([1, \text{maxwindowsize}]\)
4. \( \beta < \omega \) AND \( \omega \% \beta = 0 \)
5. \( \beta < \omega \) AND \( \omega \% \beta \neq 0 \)

• The Scenario Generator enables to control tuple distribution over application time line through predefined patterns applied to input data stream skeletons. It decides which patterns to use based on activated reporting strategies, user choices (e.g simultaneous tuples) and randomly generated window parameter values(i.e they affect intervals from which random values are chosen). Some of the patterns stated can be customized further via input alignment parameter.

Available patterns are listed below. Note that, each item in the list has a dual which contains simultaneous tuples (and simultaneous batches when enabled). For the sake of simplicity, we omit to list simultaneous counterparts.

– **regular**: Input with one and only one tuple at every application time instant, in other words tuples are 1 application time unit apart.

– **gap = \( \beta \)**: Input with tuples that are \( \beta \) application time unit apart. Input data streams following this pattern can further be customized by input alignment parameter. Tuples of the stream can be aligned to window closes \((t_0 + n\beta + \omega)\) or tuple expiration points \((t_0 + n\beta + \omega + 1)\) or can have no alignment.
− gap = \omega: Input with tuples that are \omega application time unit apart.

− gap = \lambda: Input with tuples that are \lambda application time unit apart. Input data streams following this pattern can further be customized by input alignment parameter. Tuples of the stream can be aligned to window closes or tuple expiration points or can have no alignment. Density control is available. This pattern is used only when periodic reporting is set.

− gap > \lambda: Input with tuples that are more than \lambda application time unit apart. The default gap value currently in use is 2 \times \lambda. Input data streams following this pattern can further be customized by input alignment parameter. Tuples of the stream can be aligned to window closes or tuple expiration points or can have no alignment. Density control is available. This pattern is used only when periodic reporting is set.

− gap \%\lambda = 0: Input with tuples that are less than \lambda application time unit apart. Input data streams following this pattern can further be customized by input alignment parameter. Tuples of the stream can be aligned to window closes or tuple expiration points or can have no alignment. Density control is available. This pattern is used only when periodic reporting is set.

− gap = \textbf{random}: At most 3 input scenarios with varying gap between tuples are generated. For each input data stream, gap values are chosen randomly from the following intervals respectively: [2, \beta], (\beta, \omega], (\omega,3 \times \omega](Assuming that \beta < \omega)

\textbf{Input}  The Scenario Generator does not take any input. However, it is controlled by several parameters:

- **maxwindowsize**: It sets the maximum possible size of a window that Scenario Generator can generate.

- **slide**: It either directly sets the value of the slide(\beta) as a window parameter or enables Scenario Generator to generate a random value.

- **simultaneous batches**: If enabled, Scenario Generator produces skeletons for input data streams which will contain simultaneous batches on arbitrary time instants.

- **simultaneous tuples**: If enabled, Scenario Generator produces skeletons for input data streams which will contain simultaneous tuples on arbitrary time instants.

- **no_tuple_on_period**: If enabled, Scenario Generator produces skeletons for input data streams which will contain no tuples at time instants where periodic reporting occurs.
• **input\_align**: It specifies application time instants only on which a tuple can arrive. This parameter is of vital importance for testing specific cases.

• **period\_align**: It specifies how the application time line should be adjusted so that periodic reporting happens at specific points such as tuple expiration points or window closes.

**Output**  
The Scenario Generator generates execution scenarios as output.

### 3.2.2 Input Generator

**Function**  
The Input Generator generates input data streams with the patterns specified by skeletons carried in corresponding execution scenario.

**Input**  
The Input Generator takes execution scenarios produced by Scenario Generator as input.

The module can further be controlled with the following parameters:

• **max\_batch\_size**: It defines the maximum number of tuples that a batch can contain. When the option *simultaneous\_batches* is enabled, all batches are guaranteed to contain at most given number of tuples.

• **maxinputsize**: It defines maximum number of tuples that generated data streams can contain. Actual size is randomly chosen from the interval [lower limit, maxinputsize].

**Output**  
The Input Generator generates a data stream for each input data skeleton contained in corresponding scenario as output. Generated streams are defined on tuple basis. Each tuple is expressed in the following form:

\[< t^{sys}, tuple - id, t^{app}, b_{id} > \]

### 3.2.3 Input Analyzer

**Function**  
The Input Analyzer analyzes the given input file against predefined set of evaluation vectors for each window configuration. It aims to expose hidden patterns in data streams especially for the ones which do not follow any tuple distribution pattern.

Following statistics are generated as a result of the analysis:
• The ratio of the tuples that arrive at window closes \((t_0 + n\beta + \omega)\) to all tuples

• The ratio of the window closes that contain a tuple to all window closes

• The ratio of the tuples that arrive at tuple expiration points \((t_0 + n\beta + \omega + 1)\) to all tuples

• The ratio of the tuple expiration points that contain a tuple to all tuple expiration points

• For configurations where periodic reporting is activated:
  – The ratio of the tuples that arrive at periods \((n\lambda)\) where periodic reporting happens to all tuples
  – The ratio of periods that contain a tuple to all periods
  – The ratio of tuples that arrive at \([\text{window-open, period}]\) \(\left[t_0 + n\beta, n\lambda\right]\) intervals to all tuples
  – The ratio of \([\text{window-open, period}]\) intervals that contain a tuple to all such intervals.
  – The ratio of tuples that arrive at \([\text{period, window-close}]\) \([n\lambda, t_0 + n\beta + \omega]\) intervals to all tuples
  – The ratio of \([\text{period, window-close}]\) intervals that contain a tuple to all such intervals.

**Input**  The Input Analyzer takes data files produced by Input Generator as input.

**Output**  The Input Analyzer generates a file containing analysis results as output.

### 3.2.4 SECRET Engine Simulator

Please note that SECRET Engine Simulator was not developed as part of the work we presented. Instead, upon several modifications it is used as a ready made module.

**Function**  SECRET Engine Simulator simulates the given SPE configuration through SECRET model. The simulation executes the given query on corresponding input stream and produces results.

The SECRET Engine Simulator is the core of SSAT and forms the "simulating" branch. All analyzing functionality builds up on this module. It contains an implementation of the SECRET model (Chapter 2). It is possible to simulate any engine configuration as well as several predefined SPEs.
Input  The SECRET Engine Simulator takes system parameters and query parameters of the engine configuration to be simulated and a data stream as inputs.

Output  The SECRET Engine Simulator produces execution results of the given query on corresponding input data stream as output.

3.2.5 Result Analyzer

Function  The Result Analyzer is mainly responsible for exploring relationships between outputs of simulated engine configurations. It analyzes execution results and determines the set relation between them. The Result Analyzer is capable of making two different types of comparisons:

- **Content and Scope Based Comparison** takes reporting content and corresponding scope as a defining entity. Two result elements are considered equal only if their contents are the same and they are reported at the same time, therefore have the same scope at the time of reporting.

- **Content Based Comparison** takes reporting content as a defining entity. Two result elements are considered equal if their contents are the same. The time of reporting does not matter.

The work presented in this thesis relies on the former.

Input  The Result Analyzer takes execution results of two engine configurations simulated by SECRET Engine Simulator.

Output  The Result Analyzer produces a detailed analysis document which contains both execution and comparison details. It is the most extensive output document that SSAT produces.

3.2.6 Table Maker

Function  The Table Maker categorizes given data and organizes it in a table format.

Input  The Table Maker takes raw analysis data as input.

Output  The Table Maker produces a table containing given data in an organized fashion.
3.2.7 Table Reader

Function The Table Reader reads in printout of a table produced by Table Maker and creates an in-memory representation of it.

Input The Table Reader takes the text file which contains the table as input.

Output The Table Reader generates in-memory representation of corresponding table as output.

3.2.8 Prediction Calculator

Function The Prediction Calculator generates sets of predictions for the engine comparisons contained in given execution scenarios. Each comparison is evaluated through Relation Analyzer to calculate a prediction.

Input The Prediction Calculator takes a set of execution scenarios as input. It is controlled through following parameter:

- enable_prediction_calculation: When set, prediction calculation feature is enabled.

Output The Prediction Calculator generates a set of predicted relations for each execution scenario as output.

3.2.9 Relation Analyzer

Function The Relation Analyzer is the prototype implementation of the Relation Prediction Technique (Chapter 5). It functions as the sub module of the Prediction Calculator and predicts evaluation result of given statements (Section 4.3).

Input The Relation Analyzer takes a statement to evaluate as input.

Output The Relation Analyzer generates a prediction for given statement as output.

3.2.10 Mismatch Analyzer

Function The Mismatch Analyzer compares predicted evaluation results produced by the Prediction Calculator with the real evaluation results produced by the Result Analyzer for mismatches. It also calculates prediction efficiency and misprediction distribution amongst different configurations.
Features The Mismatch Analyzer analyzes for the following evaluation vectors for every configuration match-up that contains at least one mismatch:

- **Overall Valid Prediction Ratio**:

- **Overall Match Rate**: Ratio of prediction-real value matches to all comparisons
  - **Equal Ratio**: Ratio of matches with set relation *Equal* to all matches
  - **Subset Ratio**: Ratio of matches with set relation *Subset* to all matches
  - **Superset Ratio**: Ratio of matches with set relation *Superset* to all matches
  - **Disjoint Ratio**: Ratio of matches with set relation *Disjoint* to all matches
  - **Intersecting Ratio**: Ratio of matches with set relation *Intersecting* to all matches

- **Overall MisMatch Rate**: Ratio of prediction-real value mismatches to all comparisons
  - **Equal Ratio**: Ratio of mismatches whose real values are *Equal* to all mismatches
  - **Subset Ratio**: Ratio of mismatches whose real values are *Subset* to all mismatches
  - **Superset Ratio**: Ratio of mismatches whose real values are *Superset* to all mismatches
  - **Disjoint Ratio**: Ratio of mismatches whose real values are *Disjoint* to all mismatches
  - **Intersecting Ratio**: Ratio of mismatches whose real values are *Intersecting* to all mismatches

- **Overall Low Precision Mismatch Rate**: Ratio of mismatches due to low precision predictions (Section 5.4) to all comparisons

- **Overall Ambiguous Prediction Mismatch Rate**: Ratio of mismatches due to ambiguous predictions (Section 5.4) to all comparisons

- **Low Precision Mismatch Ratio** Ratio of mismatches due to low precision predictions to all mismatches

- **Ambiguous Prediction Mismatch Ratio** Ratio of mismatches due to ambiguous predictions to all mismatches

Input The Mismatch Analyzer takes predictions produced by the Prediction Calculator and the real results produced by the Secret Simulation Module as input.
Output  The Mismatch Analyzer produces an output file containing analysis results along with overall and statistics as output.

3.2.11 Statistic Generator

Function  The Statistic Generator is a highly configurable module that is responsible for generating statistics out of collected data.

Input  It takes the raw data and a road map of calculation for desired statistics as inputs.

Output  It generates corresponding statistics as output.
Chapter 4

Exploration of Base Rules

Base Rules are sets of conditions which express relationships among different engine configurations. This chapter discusses how base rule sets can be derived. Section 4.1 explains our motivation, Section 4.2 gives a brief introduction. Section 4.3 describes our notation. Section 4.4 presents our methodology. Section 4.5 defines fundamental concepts we used in formulation of base rules. Finally, Section 4.6 explains group of example rules in detail.

4.1 The Motive

As discussed in Section 1.1, given the input stream SECRET model can further be used to explore relationships (equality, subset, superset etc.) among heterogenous SPEs. Once explored, those relationships can be used for variety of purposes. However, given the large number of available different engine configurations, apart from being impractical it is both time and resource consuming to explore all such relationships through output analysis. Hence, a more efficient and subtle technique is required.

We propose a technique to derive relationships between any given two engine configurations by using already known set of relations which belongs to a certain configuration subspace. We call this subspace as base set.

4.2 What is a Base Rule Set?

Base Rule Set refers to set of parameterized rules which defines relations between different configurations in the base set, based on window configurations and input stream properties. In other words, each base rule corresponds to the condition which has to
be fulfilled by the input stream so that certain type of relation (i.e equality) between certain pair of engine configurations can exist.

Base rules express conditions for the simplest configuration match-ups. More complex cases can be expressed in terms of these simple cases. Therefore, in order to discover base rule sets, one has to analyze for the simplest configuration match-ups.

Discovery of base rules is the first step to develop our method since it functions upon them.

### 4.3 Notation

This section gives an introduction to the notation we will use in our work. We will detail it as we need in following chapters.

We use the term "vs." (versus) as the comparison operator in between compared configurations. The comparison is always in terms of produced output. For example, given that we have two engines \( E_1 \) and \( E_2 \), when we say \( E_1 \text{ vs. } E_2 \) we refer to a comparison between outputs of regarding engines.

In a comparison, when we state engine configurations explicitly by parameter value, we use the term **statement** to refer to it. For example, \( R_{ne} \text{ vs. } R_{pr} \) is a statement expressing a comparison between a \( R_{ne} \) activated engine configuration on one side and a \( R_{pr} \) activated engine configuration on the other. Note that all available variables except the ones that are explicitly stated in the statement should be assumed to be same on both configurations unless otherwise stated. For example, for the previous example statement, one can assume both engine configurations have the same tick parameter.

### 4.4 Setup and Methodology

We used SSAT’s scenario generation capability to generate test cases with the configurations we discussed below. Then we experimented with each test case on SSAT and produced relation tables. Finally we derived base rules by those relation tables. An example relation table can be found at Table 7.1.

#### 4.4.1 Comparison Methodology

We consider elements of produced output as a unique entity with both its content and reporting time. In general reporting time can be regarded as the time information which
corresponds to creation of regarding output element. In our system, reporting time refers to scope of the system at that particular time instant when reporting occurs. For example, if two output elements have the same content but different scopes of reporting, then we consider them as different.

SSAT’s Result Analyzer module is responsible for this operation. For further information about the module and comparison methods please refer to section 3.2.5.

4.4.2 Setup

In general, when two engine configurations are being compared, following setup of the system parameters expresses the simplest configurations:

- **Tick Parameter**: Tick strategies can not be combined, thus a setup with a single tick strategy is the most basic configuration by default (Section 2.3.4).

- **Report Parameter**: A setup with single active reporting strategy per engine is the simplest possible combination where only one out of four possible reporting strategies is being used (Section 2.3.3). Different reporting strategies can be combined, hence more complex report parameters can be expressed as combination of the basic strategies.

**Engine Configuration** We derived base rule sets only for the setups listed below. Note that, any base rule set can be derived with a similar methodology.

- **Tick Parameter**: We assigned the same tick parameter to both engine configurations. Therefore, we have three different sets of base rules, one for each tick parameter combination (tuple-driven vs. tuple-driven, time-driven vs. time-driven, batch-driven vs. batch-driven).

- **Report Parameter**: We assigned a single report parameter to both engine configurations.

- **t0**: We assumed both engine configurations are executed with the same $t_0$.

- **Window Size ($\omega$)**: We assigned the same $\omega$ to both engine configurations.

- **Slide ($\beta$)**: We assigned the same $\beta$ to both engine configurations.

An example engine configuration match-up, given by the statement $E_1 vs E_2$ is as follows:
Note that $E_2 \text{ vs } E_1$ also refers to the same comparison. The total number of engine configuration match-ups to be tested is given by the number of unique Report parameter combinations, since it is the only varying parameter on opposing sides. Given that there are 4 unique report parameters (Section 2.3.3), we can omit 4 combinations which have the same reporting strategy on both sides since they always produce the same results. This gives $\binom{4}{1} \times \binom{4}{1} - 4 = 12$ combinations. Since $E_2 \text{ vs } E_1 = E_2 \text{ vs } E_1$, half of those 12 combinations can also be omitted. This results in 6 unique engine configuration match-ups to be tested.

In exploration of base rules, one of our main aims was to show that SECRET model could actually be used to explore relationships between different engine configurations. Therefore, we wanted to have single varying parameter on opposing sides to keep configuration match-ups simple. If these sets could be derived, it would mean similar methodology can be applied to derive any base rule set. Our choice of base rule sets to derive is founded on this reasoning.

**Window Configuration** For each system parameter configuration, we will explore the window configurations that follow patterns stated below:

1. $\beta = 1$ AND $\omega = 1$
2. $\beta = 1$ AND $\omega \neq 1$
3. $\beta = \omega$
4. $\beta < \omega$ AND $\omega \% \beta = 0$
5. $\beta < \omega$ AND $\omega \% \beta \neq 0$

We aim to sample window configuration space at important points effectively, so that we gather accurate results. Our choice of patterns is based on both the observations made during derivation of the SECRET model [1] and actual configurations of several available SPEs [2, 3, 10]. Note that we omit any pattern with $\beta > \omega$ since, there are no known real world application of the setting.

**Input Data** For each of the above window configurations, we generate input data streams with the below characteristics. Be aware that for simplicity reasons, simultaneous counterparts of the patterns are not listed. Except for the last category, a
corresponding input data stream with simultaneous tuples\batches is also generated. For further details about the characteristics, please refer to Section 3.2.1.

- **regular**: 1 input file.
- **gap = \( \beta \)**: 2 input files, one aligned to window closes \((t_0 + n\beta + \omega)\), one aligned to expiration points \((t_0 + n\beta + \omega + 1)\).
- **gap = \( \omega \)**: 1 input file
- **gap = \( \lambda \)**: 1 input file aligned to periods \((n \cdot \lambda)\) when \(R_{pr}\) is activated.
- **gap > \( \lambda \)**: 1 input file aligned to periods when \(R_{pr}\) is activated.
- **gap \%\( \lambda \) = 0**: 1 input file aligned to periods when \(R_{pr}\) is activated.
- **gap = random**: At most 3 input files depending on the actual values of window parameters.

For each window configuration, we aim to test all possible border cases with the input data streams. In addition, we generate 3 random input streams, one for all available intervals, so that we can catch any interesting behavior that might be missed by border cases.

### 4.5 Definitions

In this section, we will explain the concepts that we defined and used through our analysis.

#### 4.5.1 Set Relations

We slightly redefined some of the basic set theory relation concepts to express relationships between engine configurations. The complete list of relations we used in our work is as follows:

Let \( A \) and \( B \) are sets and \( x = A \setminus B \), \( z = A \cap B \) and \( y = B \setminus A \);
EQUA L : if \( x = \emptyset \land y = \emptyset \land z \neq \emptyset \), then we say sets are equal.

SUBSET : if \( x = \emptyset \land y \neq \emptyset \land z \neq \emptyset \), then we say A is subset of B.

SUPERSET : if \( x \neq \emptyset \land y = \emptyset \land z \neq \emptyset \), then we say A is superset of B.

DISJOINT : if \( z = \emptyset \), then we say sets A and B are disjoint. Additionally, if one of the sets is an empty set (\( x = \emptyset \lor y = \emptyset \)), then we also say sets are disjoint. Thus, empty sets always lead to disjoint relation.

INTERSECTING : if \( x \neq \emptyset \land y \neq \emptyset \land z \neq \emptyset \), then we say set A is intersecting with B.

4.5.2 General Concepts

In this section we will explain constructs that we use in our formulation.

\( S(t) \) denotes the set of tuples in stream \( S \) that has arrived through application time instant \( t \).

\[
S(t) = \{ s \in S \mid s.\text{app} \leq t \}
\]

\( S_i(t) \) denotes the set of tuples in stream \( S \) that has arrived at application time instant \( t \). They are called as simultaneous tuples.

\[
S_i(t) = \{ s \in S \mid s.\text{app} = t \}
\]

\( S_{bid}(t) \) denotes the set of bid’s of tuples in stream \( S \) that has arrived at application time instant \( t \).

\[
S_{bid}(t) = \{ s.\text{bid} \mid s \in S_i(t) \}
\]

\( B_i(bid, t) \) denotes the set of tuples in stream \( S \) that belongs to batch defined by bid which arrived at application time instant \( t \).

\[
B_i(bid, t) = \{ s \in S_i(t) \mid s.\text{bid} = bid \}
\]

\( \text{maxBatchSize}(t) \): It returns the size of the largest batch that arrived at application time instant \( t \).

\[
\text{maxBatchSize}(t) = \arg \max_{bid \in S_{bid}(t)} | B_i(bid, t) |
\]
4.6 Base Rules

Example base rule sets which we derived by the setup we explained can be found in Appendix A.

We will now briefly explain a small subset of one of the example base rule sets. The subset expresses relations for the following statement:

$$(\text{tick: tuple\_driven report: } R_{cc}) \text{ vs. (tick: tuple\_driven report: } R_{wc})$$

$R_{cc} = R_{wc}$

$$\forall n S_i(t_0 + n\beta + \omega) \neq \emptyset \land \beta = 1 \land \forall t \mid S_i(t) \mid \leq 1$$

Given engine configurations produce the same results if and only if all content changes occur at window closes and there are no window closes without a content change. This can be accomplished only when $\beta = 1$ since otherwise expired tuples would cause a content change at a non window close point($t_0 + n\beta + \omega + 1$), which in turn causes $R_{cc}$ to report at that time instant, thus invalidating equality. Lastly, there can not be any simultaneous tuples since they are reported individually by $R_{cc}$, while $R_{wc}$ reports them all at once by definition (every window closes only once [1]).

$R_{cc} \supset R_{wc}$

$$\forall n S_i(t_0 + n\beta + \omega) \neq \emptyset$$

If there is a tuple at every window close, then both engines are guaranteed to report at that same time instants. Therefore, if no other tuple arrives, the output of the engines will be equal. If there is a tuple arriving at any point which is not a window close, then only $R_{cc}$ will report. Thus results will have the superset relation. Note that, a window close without a content change is not allowed since it would turn the relation between regarding engine configurations to intersecting.

$R_{cc} \subset R_{wc}$

$$\exists n S_i(t_0 + n\beta + \omega) \neq \emptyset \land \beta = 1 \land \forall t \mid S_i(t) \mid \leq 1$$

This case is almost similar with the equality case but it is less strict. Reporting due to window close without existence of a content change is allowed (but a content change can
not occur at a non window close point, since this would invalidate subset relationship). This is expressed with a change of qualifier in the condition.

\[ R_{cc} \cap R_{wc} = \emptyset \]

\[ \forall n \ S_i ( t_0 + n\beta + \omega ) = \emptyset \land \beta \neq 1 \]

If no tuple arrives at any window close, then \( R_{cc} \) and \( R_{wc} \) can not report at the same time instants. Therefore results will be disjoint. When \( \beta = 1 \), there can not be any tuple at all since a window closes at every single application time instant, hence no result is produced. Empty results are considered as disjoint by default and second part of the condition just makes this distinction.
Chapter 5

Relation Prediction Technique

Chapter 4 presents our methodology for exploring base rule sets. In this chapter, having rule sets as basis, we will present the Relation Prediction Technique (RPT), a technique we propose for exploring relationships between different SPE configurations. Section 5.1 details our notation, an overview is given in Section 5.2. The algorithm is discussed in detail in Section 5.3. Limitations of the technique are discussed in Section 5.4. Finally, execution of RPT is illustrated on an example in Section 5.5.

5.1 Notation

We will extend the notation described here in Section 4.3.

As presented, Statement refers to a comparison between two engine configurations. A statement consists of two Statement Terms, Left Term and Right Term respectively, each corresponding to an opposing configuration. Each statement term consists of elements referring to values of single parameters in the configuration, therefore each element expresses a unique condition which will affect the output.

Assume we have the following engine configurations:

\[ E_1 \leftarrow \text{Tick = tuple\_driven, Report = } R_{ne}R_{pr} \]
\[ E_2 \leftarrow \text{Tick = tuple\_driven, Report = } R_{cc} \]

The statement \( R_{ne}R_{pr} \text{ vs. } R_{cc} \) expresses the comparison between above configurations. \( R_{ne}R_{pr} \) is the Left Statement Term consisting of two elements \( R_{ne} \) and \( R_{pr} \), while \( R_{cc} \) is the Right Statement Term consisting of a single element \( R_{cc} \).
Often, we will categorize statements by number of elements which are explicitly stated. For instance, the above statement is categorized as 2vs1 since left term contains 2 elements and right term contains 1 element. Any such statement following the same pattern belongs to the same category.

5.2 Overview

The method we present aims to predict relational connection between the foreseen outputs of two engine configurations. It uses known set of relationships between regarding simplest configurations as a basis of prediction so that no output analysis is required. By output analysis, we mean simulating given engine configurations or testing with real SPEs and comparing results to expose the relationship.

Each element in a Statement Term corresponds to a condition in configuration which can produce output of its own when it is alone. When multiple elements come together, the common condition they refer is the conjunction of individual conditions. In the same manner, if each element in a Statement Term produces a result set of its own, the Statement Term itself produces the result set which is expressed by intersection of those individual sets. The evaluation of a Statement then refers to discovery of set relation between final result sets produced by each Statement Term.

The configuration with a single condition is always the simplest one, since no conjunction is required in determination of the condition and respectively no intersection is required in generation of corresponding result set. Therefore, 1vs1 statements always express the simplest configuration match-up’s. We refer them as Simple Statements. Any other statement which does not belong to this group is a Complex Statement.

RPT suggests a method to evaluate complex statements by their simpler components. It minimizes the processing burden by propagating it to simplest components. The principle behind the algorithm is as follows: given three sets A, B and C, once all 1 to 1 relations between sets are known (A vs B, A vs C and A vs B), the relation between any combination of sets, such as $A \cap B$ vs C can be deduced as well. Assume $A \supset C$, $A \supset B$, $B$ and $C$ are INTERSECTING as depicted in Figure 5.1. This information is sufficient to know relative positions of sets to each other. Therefore, it can be deduced that $A \cap B$ and $C$ would be INTERSECTING. Here, statements expressing 1 to 1 relations are the simplest components while the statement expressing relation between combination of sets is a complex statement. The same principle can be applied recursively to any statement with any level of complexity, thus producing a tree like parsing structure which we refer as Processing Tree.
If a building block can be found in terms of which all complex statements can be evaluated, then a data table specifying which relation combinations leads which final relation for that building block can also be created. It is this table which makes it possible to apply the principle to any statement with any level of complexity, since all can be evaluated in the same manner through the table. We will refer this table as Prediction Table. It will be discussed in detail in Section 5.3.2. The number of configurations the table should contain to catch all cases is determined by the chosen building block.

RPT uses 2vs1 statements as building blocks. It requires evaluation of three 1vs1 statements to predict regarding 2vs1 statement. Any statement more complex than that is basically treated as a 2vs1 statement and parsed down to its components. If components are not simple enough, they are parsed as well. The process goes on until all subcomponents come down to 1vs1 form. We call this process as parsing phase. Once parsing phase is completed, base rules step in to evaluate 1vs1 statements\(^1\). From then on, prediction table can be used repeatedly to predict the evaluation result of the ancestor component until root component is reached. We call this process as prediction phase.

In the next section, we present detailed description of our technique.

### 5.3 The Algorithm

The main algorithm of RPT is given in Algorithm 1.

The algorithm starts with simplification phase. It is an optional preprocessing phase applied on both terms of the input statement to reduce the number of elements where possible, so that less cases are dealt during both parsing and prediction phases. As a result of simplification phase, input statement might even be reduced to 1vs1 simplest

\(^1\)1vs1 statements are placed at leaves in the processing tree. The "base" in naming of base rules originates from this property.
Algorithm 1: Main Algorithm

Data: Statement statement to be evaluated
Result: SetRelation relation as an evaluation outcome

1 begin
2 Simplify(statement.leftTerm)
3 Simplify(statement.rightTerm)
4 // if any two elements are disjoint
5 if statement.ContainsEmptyTerm then
6 | return DISJOINT
7 // if both terms of the statement contain only one element
8 else if statement.isOneVsOne then
9 | return evaluateByFormula(statement)
10 else
11 | return evaluate(statement)

form which can be evaluated through base rules such that actual RPT does not need to be executed. evaluateByFormula() is responsible for calculation through base rules.

Disjoint relation is a shortcut case for RPT. Whenever it occurs at any point of the prediction tree, the final result is guaranteed to be disjoint. In the same manner, any discovery of disjoint relation during simplification phase results in removal of all elements in corresponding statement to mark the event. In algorithm, the expression statement.ContainsEmptyTerm tests for this case.

Finally, if none of the above cases occur, the recursive phase begins and actual prediction algorithm is executed. Once again, each step up to this point is optional and aims improvement in processing speed as well as a reduction in operational complexity of the recursive phase.

Next we will discuss simplification phase, prediction table and recursive phase in detail.

5.3.1 Simplification Phase

Simplification phase is an optional phase which refers to removal of unnecessary elements from the corresponding Statement Term. As discussed in previous section, a Statement Term refers to a result set which can be expressed by intersection of individual sets that are referred by its elements. Because of this characteristic, the common result set of any two such individual sets can be represented by the more specific of the two in terms of intersection where possible. Let there be statement term $abc$. Assume the relation
between result sets that \( a \) and \( b \) represent is \( R_a \subset R_b \). Since the result set expressed by statement term is given by \( R_{abc} = R_a \cap R_b \cap R_c \), the statement can be simplified to \( ac \). There is no need to have the element \( b \) anymore since intersection always favors the contained set. Removal of all such unnecessary elements means less nodes in processing tree at the recursive prediction step.

The choice of simplification strategy is up to the implementer since there is no preset method assumed for RPT. Yet, in general we classify possible strategies in two categories which we call as naive simplification and smart simplification.

**Naive Simplification** refers to a strategy where elements are evaluated case by case in a dumb way without taking the general picture into account. Whenever a suitable simplification is found, it is applied. Algorithm 2 is an example of this kind.

**Smart Simplification** refers to a strategy where a heuristic approach is used to determine the best possible combination of simplifications such that the reduction of the statement term can be maximized. Smart simplification strategies are expected to perform better reduction, though they are more complex and more expensive to implement.

**Algorithm 2: Simplify**

```plaintext
Data: StatementTerm statementTerm
Result: simplified statementTerm
begin
ptrA ← statementTerm.getElement(0)  // first element in the term
ptrB ← statementTerm.getElement(1)  // second element in the term
while ptrA < statementTerm.lastElement && ptrB ≤ statementTerm.lastElement && ptrB > ptrA do
    element_a ← statementTerm.getElement(ptrA)
    element_b ← statementTerm.getElement(ptrB)
    subStatement ← new statement(element_a, element_b)
    /* check history to avoid multiple evaluation of the same statement */
    relation ← checkEvaluationHistory(subStatement)
    if relation.exist = false then
        relation ← evaluateByFormula(subStatement)
        updateEvaluationHistory(subStatement, relation)
    action ← SimplificationTable.check(relation)
    if action.exists then
        applySimplification(statementTerm, ptrA, ptrB, action)
updatePointers(ptrA, ptrB, action)
```
Our strategy which is given by Algorithm 2, maintains two pointers on the elements of the input Statement Term. At every iteration, it generates a 1vs1 temporary statement with pointed elements and evaluates it by base rules. Next, a data structure, which we refer as Simplification Table containing all mappings from set relation to action of simplification is checked to determine suitable action for evaluation result. Once an action is found, it is applied. As a final step, pointers are updated. Algorithm terminates when all possible combinations of remaining elements are checked for simplification.

The Simplification Table as we present in Table 5.1 specifies the action describing which element of the compared pair is unnecessary, hence should be removed. Note that we define Simplification Table in accordance with our needs, thus it does not pose a design requirement. One can redefine the structure and data contained as he wishes. In our table representation, a relation in the table refers to relation between sets A and B respectively.

<table>
<thead>
<tr>
<th>Set Relation (A vs. B)</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQUAL</td>
<td>remove-second-element</td>
</tr>
<tr>
<td>SUBSET</td>
<td>remove-second-element</td>
</tr>
<tr>
<td>SUPERSET</td>
<td>remove-first-element</td>
</tr>
<tr>
<td>DISJOINT</td>
<td>remove-all</td>
</tr>
<tr>
<td>INTERSECTING</td>
<td>no-action</td>
</tr>
</tbody>
</table>

Table 5.1: Simplification Table

INTERSECTING is a special case where element reduction is not possible since there are no contained or containing sets. Hence, the efficiency of a simplification strategy can be measured by how efficient any non-intersecting relation can be caught and simplified.

In Algorithm 2, we introduce a new concept: Evaluation History. It is an optional, global data structure which acts as a cache. It holds "statement:relation" pairs as well as "reversed statement:relation" pairs for any statement processed so that multiple evaluation of the same statement can be prevented. It makes use of the fact that if evaluation result of a statement is known, evaluation result of the reversed statement can be deduced. For example, if A vs B evaluates to subset, then B vs A evaluates to superset. Just as any cache, Evaluation History increases overall space complexity, but it can be helpful in reducing execution times especially when statements with re-occurring elements on both terms are processed. Evaluation History is managed by checkEvaluationHistory() and updateEvaluationHistory().
**Analysis**  In the worst case, all $n$ elements contained in the statement term are compared in pairs. First element is compared with next $n-1$ element, second element is compared with next $n-2$ element and so on which is $(n-1)+(n-2)+...+2+1 = n(n-1)/2$ comparisons in total. Thus worst case complexity is given by $O(n^2)$. In the best case, a disjoint relation is encountered in the first comparison, thus simplification ends immediately. This is given by $O(1)$. `applySimplification()` and `updatePointers()` do not affect complexity.

Algorithm 3 describes how element pointers are updated at the end of each iteration of `Simplify()`.

---

**Algorithm 3: updatePointers**

```
Data: StatementTermPointer firstPtr, secondPtr; StatementTerm statementTerm

begin

if action not exists OR action = no-action then

    secondPtr ← secondPtr.next
    // if secondPtr has reached end

    if secondPtr > statementTerm.lastElement then

        firstPtr ← firstPtr.next
        secondPtr ← firstPtr.next

else if action = remove-first-element then

    firstPtr ← firstPtr.next
    secondPtr ← firstPtr.next

else if action = remove-second-element then

    secondPtr ← secondPtr.next
    if secondPtr > statementTerm.lastElement then

        firstPtr ← firstPtr.next
        secondPtr ← firstPtr.next
```

Algorithm 4 describes how the action acquired from Simplification Table is applied to the corresponding Statement Term. First, element to be removed is decided upon given action. Then corresponding element is removed from the statement.
Algorithm 4: ApplySimplification

Data: StatementTerm statementTerm, Action action, StTermPointer firstElPtr, secondElPtr
Result: Simplified statementTerm

begin
  // in case intersecting
  if statementTerm.elementCount = 0 OR action = no-action then
    return false
  else
    // in case of disjoint
    if action = remove-all then
      statementTerm.removeAllElements
    else
      // in case of superset
      if action = remove-first-element then
        elementToDelete ← statementTerm.getElement(firstElPtr)
      // in case of subset or equal
      else if action = remove-second-element then
        elementToDelete ← statementTerm.getElement(secondElPtr)
      statementTerm.removeElement(elementToDelete)
    end
  return true

5.3.2 Prediction Table

Prediction Table is the data structure used by RPT algorithm to evaluate a building block. It describes which combination of relations acquired by evaluation of sub components leads which final prediction.

As described earlier in Section 5.2, RPT uses components in 2vs1 form as building blocks which contains 3 elements in total. Evaluation of 2vs1 statement requires three 1 to 1 relations among elements of the block.

The complete prediction table is given in Appendix B as well as validations by set theory of each combination contained in the table.

Prediction table has following properties:

- Given result sets of $R_1$, $R_2$ and $R_3$, we predict result of the target statement $R_1R_2$ vs $R_3$ from results of the statements:
Chapter 5. Relation Prediction Technique

1. $R_1$ vs $R_3$
2. $R_2$ vs $R_3$
3. $R_1$ vs $R_2$

Since multiple conditions in a Statement Term means intersection between individual result sets of regarding elements, target statement can also be written as $R_1 \cap R_2$ vs $R_3$ in terms of set operations. While the last column of the table corresponds to the prediction, first three columns correspond to evaluation results of sub components in the exact order stated.

- $R_1$, $R_2$ and $R_3$ can be freely chosen to represent result set of any element or element combination in the given statement. However, the prediction always corresponds to evaluation result of the statement $R_1 R_2$ vs $R_3$.

- Prediction table does not contain all possible combinations of relations for given building block. Instead, it contains a representative set of combinations to keep the size small. The combination count as well as the combinations themselves are determined by the statements we assigned to columns.

For example, for the statement $R_1 R_2$ vs $R_3$ we assign sub statements $R_1$ vs $R_3$, $R_2$ vs $R_3$ and $R_1$ vs $R_2$ to columns and hold corresponding relation combinations in the table. However, although logically evaluation results of $R_2$ vs $R_1$, $R_2$ vs $R_3$ and $R_2$ vs $R_1$ lead prediction of the same statement, we can not process the data since we do not contain it. Therefore we convert it to the first form algorithmically.

- The columns of the table are not commutative. Hence the order of application of the relations is important and effects the prediction.

Consider the following example: given elements A,B,C assume we have the following evaluated statements;

A vs C $\rightarrow$ SUPERSET
B vs C $\rightarrow$ SUPERSET
A vs B $\rightarrow$ EQUAL

We choose $R_1$ to denote result set of A, $R_2$ to denote result set of B and $R_3$ to denote result set of C and we would like to deduce the evaluation result for the statement $AB$ vs $C$.

Now we apply relations to the Table B.1 as follows:

first column$\leftarrow$ result of $A$ vs $C$
second column$\leftarrow$ result of $B$ vs $C$
and third column$\leftarrow$ result of $A$ vs $B$
which will give us the prediction SUPERSET for \( AB \ vs \ C \). But if we apply relations as follows:

- first column ← result of \( A \ vs \ C \)
- second column ← result of \( A \ vs \ B \)
- and third column ← result of \( B \ vs \ C \)

we get prediction of EQUAL, which is wrong.

- Since for sets \( S_1 \) and \( S_2 \), \( S_1 \cap S_2 = S_2 \cap S_1 \) by commutative law, we can say \( AB \ vs \ C = BA \ vs \ C \). Thus, both of the following assignments are valid for evaluating the statement \( AB \ vs \ C \):
  - \( R_1 \) to denote result set of \( A \), \( R_2 \) to denote result set of \( B \) and \( R_3 \) to denote result set of \( C \)
  - \( R_1 \) to denote result set of \( B \), \( R_2 \) to denote result set of \( A \) and \( R_3 \) to denote result set of \( C \)

- Several relation combinations can lead to multiple predictions. In such cases, we choose the most representative prediction by our relation definitions (Section 4.5.1). We call such predictions as **Low Precision Predictions** since precision is lost due to generalization. If they cannot be commonly represented by a member, then we cannot make any logical prediction. Instead, we favor INTERSECTING without any base. We call our prediction as **Ambiguous Predictions** since it is random and untrustworthy. Further details can be found in Section 5.4.
5.3.3 Recursive Phase

Recursive phase refers to the execution of recursive prediction algorithm described in Algorithm 5.

**Algorithm 5: evaluate**

```plaintext
Data: Statement statement
Result: SetRelation relation
begin
  relation ← checkEvaluationHistory(statement)
  if relation.exists then
    return relation;
  else if statement.isOneVsOne then
    return evaluateByFormula(statement)
  else
    subStatements ← parseAsTwoVsOne(statement)
    foreach statement in subStatements do
      // recursive call
      relation ← evaluate(statement)
      // Terminate. Final relation will be disjoint
      if relation = DISJOINT then
        return relation
    stRelPair ← pairUp(relation, statement)
    statementRelPairs.add(stRelPair)
    updateEvaluationHistory(statement, relation)
  return evaluateByTable(statementRelPairs)
```

`evaluate()` first checks whether given statement was evaluated before. If it was, evaluation result is retrieved from Evaluation History directly. Then, it checks whether input statement is in 1vs1 form. Simple statements constitute the base case for recursion since they cannot be reduced further. They are evaluated through base rules. If given statement is a complex statement, then it has to be parsed down to its components first. Parsing is done by Algorithm 6. Each component is evaluated individually in the
same manner. Once all components of a statement are evaluated, a prediction is made through Prediction Table for the statement.

Note that right after recursive call returns, each statement is paired up with its evaluation result. This is required since during parsing, statements are associated with a column in Prediction Table so that their evaluation results are applied to associated columns.

The Algorithm 6 describes how we parse given statements.

**Algorithm 6: parseAsTwoVsOne**

```java
Data: Statement statement
Result: Statement parsed sub-statements

begin

if statement contains empty term OR statement is one vs. one then
  return

if statement.leftTerm.elementCount ≠ 1 then
  elementSet1 ←− statement.leftTerm.getElementsFromTo(0, 0)
  elementSet2 ←− statement.leftTerm.getElementsFromTo(1, lastElement)
  elementSet3 ←− statement.rightTerm.getElementsFromTo(0, lastElement)
else
  elementSet1 ←− statement.rightTerm.getElementsFromTo(0, 0)
  elementSet2 ←− statement.rightTerm.getElementsFromTo(1, lastElement)
  elementSet3 ←− statement.lefttTerm.getElementsFromTo(0, lastElement)
mark as reverse

// mark with appropriate keys for table look up
parsedStatements.add("AC", new statement (elementSet1, elementSet3) )
parsedStatements.add("BC", new statement (elementSet2, elementSet3) )
parsedStatements.add("AB", new statement (elementSet1,elementSet2) )
```

As described in Section 5.3.2, the 2vs1 statement $AB \text{ vs } C$ is parsed into three sub statements $A \text{ vs } C$, $B \text{ vs } C$ and $A \text{ vs } B$. Elements that are represented by A, B and C are chosen as follows:

1. A: first element
2. B: $2^{nd}$ element $t$ $n^{th}$ element
3. C: all elements of the unused term
Each sub statements is marked to indicate on which column of Prediction Table its result should be applied to.

As discussed, RPT is specialized to evaluate its building block only. Hence any other statement form except 2vs1 is first converted and then evaluated. The conversion is done by `parseAsTwoVsOne()` and marked with the flag "reverse", so that after evaluation corresponding result is converted back to fit the form of original statement.

Analysis At each step of recursion in Algorithm 5, input statement is parsed into its components by Algorithm 6. Parsing always produces 3 sub components. Each of these components then evaluated recursively. When a component is in 1vs1 form, recursion stops. Therefore, algorithm creates an implicit ternary parsing tree, leaf nodes of which are constituted by 1vs1 statements. An example prediction tree can be seen at Figure 7.1.

In a tree with $n$ nodes, time complexity is given by $O(n)$ since each subtree of a node has to be evaluated first in order to evaluate the node itself.

Once a node in the tree is evaluated, its 3 subtrees are no longer held in the memory, hence the tree is implicit. Therefore, space complexity is given by $\theta(h)$ where $h$ denotes height of the tree. Since in a ternary tree height is given by $\log_3(n)$, space complexity becomes $\theta(\log_3(n))$.

At each level of the tree, depending on the direction of parsing (left to right vs right to left) number of elements of the statement term in that direction is decreased 1. In the last level of the tree both sides of the corresponding statement contain 1 element. Therefore height of the tree can also be expressed by

$$(\text{rightStatementTerm.elementCount} + \text{leftStatementTerm.elementCount} - 2)$$

5.4 Limitations

As described in Section 5.3.2, RPT has two limitations. Some relation combinations lead to multiple predictions among which one has to be chosen. Precise selection can only be made by analyzing outputs of regarding engine configurations, however this is what RPT has been developed to avoid, therefore it is not possible. Given the circumstances, we follow below procedures in such cases:

Low Precision Prediction One way to resolve the issue is to choose the most representative relation of all. The trade off is loss of precision due to generalization, however
the prediction will still be correct and trustable. We refer such predictions as Low Precision Predictions.

In Figure 5.2, relation combination of $A \text{ vs } C$, $B \text{ vs } C$ and $B \text{ vs } C$ produces two possible predictions for $A \cap B \text{ vs } C$. Among them (I) is chosen since intersecting is more representative than subset (Section 4.5.1).

![Figure 5.2: Low Precision Prediction case 1.](image)

In Figure 5.3, again given relation combination produces two possible predictions for $A \cap B \text{ vs } C$. (II) is chosen since subset is more representative than equal.

![Figure 5.3: Low Precision Prediction case 2.](image)

**Ambiguous Prediction** If no relation can be selected to represent all possible relations, we made a random choice. This makes our prediction ambiguous, therefore untrustworthy and it means we simply can not make any accurate prediction. We call this kind of predictions as Ambiguous Predictions. Note that since multiple predictions are in question we consider ambiguous predictions as low precision predictions as well. Therefore, they are included in corresponding statistical calculations.

There is only 1 relation combination that leads ambiguous predictions. In Figure 5.4, relation combination of $A \text{ vs } C$, $B \text{ vs } C$ and $B \text{ vs } C$ produces three possible predictions for $A \cap B \text{ vs } C$ and none of them can express all the others, hence no generalization is possible. In this particular case, we favor (I), the intersecting case, without any basis to our decision.
5.5 RPT in Action

In this section, we will show how RPT works with a simple example.

We would like to predict evaluation result of the statement *abc vs cd*. Assume the relations between elements of the statement are as listed below.

- *a vs b*: INTERSECTING
- *a vs c*: SUPERSET
- *b vs c*: SUBSET
- *b vs d*: INTERSECTING
- *c vs d*: INTERSECTING

RPT starts with simplification phase where each term of the statement is processed separately by Algorithm 2. Figure 5.5(a) shows how statement term *abc* is simplified, while Figure 5.5(b) depicts the process for Statement Term *cd*.

In Figure 5.5(a), algorithm starts with checking the relation between *a* and *b* by evaluating the statement *a vs b* through base rules. INTERSECTING is acquired as a result. Then, algorithm checks Table 5.1 with it for a simplification action. Since corresponding entry in the table is *no_action*, no simplification can be made. Pointers are updated and next element pair, *a* and *c* are compared in the same manner. SUPERSET is acquired as a result. This time, Table 5.1 returns *remove_first_element* as simplification action. Hence, element *a* is removed. As a final step, evaluation of statement *b vs c*
yields to SUBSET whose corresponding simplification action is \textit{remove\_second\_element}. Therefore element $c$ is removed. Since there are no other elements to be compared, simplification ends.

In Figure 5.5(a), elements $c$ and $d$ are compared through statement $c \ vs \ d$ which evaluates to INTERSECTING. As a result no simplification occurs.

Now, Algorithm 5 evaluates simplified statement $b \ vs \ cd$. Figure 5.6 depicts the evaluation process on resulting \textit{prediction tree}. Note that the original statement $abc \ vs \ cd$ would require 19 nodes in prediction tree to be evaluated, while after simplification it requires only 4 nodes. Therefore, for this case operational complexity is 78\% reduced by simplification phase.

\begin{figure}[h]
\centering
\includegraphics[width=0.7\textwidth]{recursive_evaluation.png}
\caption{Recursive evaluation of statement $b \ vs \ cd$}
\end{figure}

The statement is parsed into its components by Algorithm 6. Then, resulting leaf nodes are evaluated through base rules. Evaluation results can be seen on Figure 5.6. Once evaluation of all child nodes are completed, Prediction Table B.1 is used to calculate the final prediction. In our case, line 17 corresponds to our relation combination which produces SUBSET as the prediction. Notice the order in which evaluation results are applied to the Prediction Table. At this point, execution of the algorithm is not over yet.

Circle with R indicates, the algorithm will reverse the evaluation result of corresponding node while passing it to its ancestor. As we explained in discussion of Algorithm 6, under the hood, RPT can make predictions only about statements in form $2 \ vs \ 1$. Any other statement is first converted to this form and then evaluated. Following evaluation, the
result is converted to fit the original form of the statement. If the child nodes of the root are inspected closely, it can be seen that statement $b \text{ vs } cd$ is parsed as if it is $cd \text{ vs } b$. Therefore, calculated result SUBSET belongs to the statement $cd \text{ vs } b$. Then, relation SUPERSET expresses the evaluation result of the statement $b \text{ vs } cd$. 

Chapter 6

Validation of Relation Prediction Technique

In this chapter, we present experiments we run to test correctness of RPT. The system setup and experiment methodology is explained in Section 6.1. Results and observations are discussed in Section 6.2. Finally, Section 6.3 discusses how RPT can be used to derive complex rules from base rules over whole target search space.

6.1 Setup and Methodology

We used SSAT (Chapter 3) to conduct experiments and validate RPT. Overview of our methodology is as follows:

- First, we implemented RPT as a module in SSAT (Section 3.2.9)

- Then, we generated a set of test scenarios with the setup we discussed in Section 6.1.1. Each test scenario groups a system parameter configuration match-up with a set of window configurations and input streams so that regarding system configurations can be tested for different cases.

- We executed all scenarios on SSAT. In detail, for every window configuration we repeatedly simulated given system parameter configurations with assigned input streams. After each simulation pair of opposing configurations, we compared the outputs to infer set relation between them. At the end, we acquired a set of values revealing relationships between system parameter configurations for different window configuration and input stream pairs. We call this relation set as Real
Chapter 6. Validation of Relation Prediction Technique

Result Set, since it is generated by simulation. An example can be found at Table 7.1.

- We evaluated all scenarios through RPT which is embedded into SSAT. In detail, for every window configuration we evaluated corresponding statement, which express comparison between given system parameter configurations, with Relation Prediction Technique and calculated a prediction. At the end, we acquired a set of values revealing predicted relationships between system parameter configurations for different window configuration and input stream pairs. We call this result set as Predicted Result Set. An example can be found at Table 7.2.

- We analyzed real result set and predicted result set for mismatches to measure the accuracy of prediction algorithm.

6.1.1 Setup

Input Data  To generate input streams, we used the same patterns as we did for exploration of base rules in Section 4.4.2. Note that we omitted to list simultaneous counterparts.

- **regular**: 1 input file.
- **gap = \( \beta \)**: 2 input files, one aligned to window closes \((t_0 + n \beta + \omega)\), one aligned to expiration points \((t_0 + n \beta + \omega + 1)\).
- **gap = \( \omega \)**: 1 input file
- **gap = \( \lambda \)**: 1 input file aligned to periods \((n \ast \lambda)\) when \(R_{pr}\) is activated.
- **gap > \( \lambda \)**: 1 input file aligned to periods when \(R_{pr}\) is activated.
- **gap \( \% \lambda = 0 \)**: 1 input file aligned to periods when \(R_{pr}\) is activated.
- **gap = random**: At most 3 input files depending on the actual values of window parameters. The input streams in this category assume no pattern in tuple distribution. Therefore, they are more representative of real time scenarios.

Window Configuration  We used same window configuration patterns as we did for exploration of base rules in Section 4.4.2. Our motive is still the same.

- \( \beta = 1 \) AND \( \omega = 1 \)
- \( \beta = 1 \) AND \( \omega \neq 1 \)
\[ \beta = \omega \]
\[ \beta < \omega \text{ AND } \omega \% \beta = 0 \]
\[ \beta < \omega \text{ AND } \omega \% \beta \neq 0 \]

**Engine Configuration**  In order to validate RPT thoroughly, whole available configuration space should be covered. We divided configuration space into 2 sub spaces according to tick parameter combinations. For each subspace, we ran distinct experiments which we named as, **Report Parameter Analysis** and **Cross Parameter Analysis** respectively.

The parameters below are common in setup of both analysis's:

- **Report Parameter**: We tested for all possible report parameter combinations which are grouped into statement types 1vs1, 2vs1, 2vs2 ... 4vs4. This gives 120 report parameter combinations in total.
- **t_0**: We assumed both engine configurations have the same t_0.
- **Window Size (\( \omega \))**: We executed both engine configurations with the same \( \omega \).
- **Slide (\( \beta \))**: We executed both engine configurations with the same \( \beta \).

We divided the search space into 2 sub spaces as follows:

1. **Report Parameter Analysis** We assigned the same tick parameter to both engine configurations and analyzed for all Report parameter combinations. Hence, we have three different tick parameter combinations;

   (a) tick driven vs tick driven
   (b) time driven vs time driven
   (c) batch driven vs batch driven

   Report parameter analysis couples 3 different tick parameter combinations with 120 different report parameter combinations, producing 360 configurations in total.

2. **Cross Parameter Analysis** We analyzed for all possible tick parameter, report parameter combinations except the ones we have already tested in Report Parameter Analysis. This gives us 6 different tick parameter combinations excluding the ones with the same value on both sides.

   (a) tick driven vs time driven
(b) tick-driven vs batch-driven
(c) time-driven vs batch-driven
(d) reversed counterparts of the above

Cross parameter analysis couples 6 different tick parameter combinations with 120 different report parameter combinations, producing 720 configurations in total.

6.1.2 Expectations

In this section we will describe our expectations from the experiments we conduct.

If all mismatches between real results and predicted results are due to Low Precision Predictions, then we say RPT works accurately. Low precision predictions are trustable but not precise. Because of the way they are generated (Section 5.4), they are often more general than their real counterparts and they are likely to cause mismatches. Low Precision Mismatch Ratio is the most important indication of accuracy for RPT. We expect it to be 100% for every case as an indication of absence of mismatches caused by wrong predictions.

Overall Ambiguous Prediction Mismatch Rate specifies the rate of mismatches caused by ambiguous, therefore inaccurate, predictions to all predictions. As low precision predictions, ambiguous predictions are also caused by the limitations of RPT. Hence, they are assessed differently than wrong predictions although both group of predictions can not be trusted. Overall Ambiguous Prediction Mismatch Rate can also be interpreted as Overall Inaccuracy Rate to indicate the natural expected fault rate of the algorithm. We expect it to be at negligible rates when total number of predictions are considered.

If Prediction Table B.1 is examined closely, it can be seen that multiple predictions are clustered as follows:

- EQUAL/SUPERSET
- SUBSET/INTERSECTING

In the clusters listed above, SUPERSET and INTERSECTING are more general than their respective counterparts, thus they are chosen as predictions by RPT. This means in foreseen mispredictions, corresponding real values should be either EQUAL or SUBSET. Additionally, during the prediction phase RPT produces intermediary evaluation results. They might be reversed in the course of evaluation to match with the form of the statement they correspond to (e.g 3vs1, 1vs3) (Section 5.3.3). Therefore, not only EQUAL
and SUBSET but also SUPERSET could be a real result in case of a misprediction. On the contrary, INTERSECTING and DISJOINT could not since they are already favored when there are multiple interpretations of a prediction. For this reason, we expect Overall Mismatch Rate $\rightarrow$ Disjoint Ratio and Overall Mismatch Rate $\rightarrow$ Intersecting Ratio to be 0% as complementary statistics to Low Precision Mismatch Ratio.

### 6.2 Results and Observations

Detailed explanations of evaluation vectors can be found at Section 3.2.10.

#### 6.2.1 Report Parameter Analysis

Report Parameter Analysis aims to validate RPT on configurations with the same tick parameter on opposing sides. At the same time, it also proves validity of example basic rules derived in Chapter 4.

Overall results of report parameter analysis is as follows:

<table>
<thead>
<tr>
<th>Overall Valid Prediction Ratio</th>
<th>%100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Match Rate</td>
<td>%91</td>
</tr>
<tr>
<td>Equal Ratio</td>
<td>%52</td>
</tr>
<tr>
<td>Subset Ratio</td>
<td>%28</td>
</tr>
<tr>
<td>Superset Ratio</td>
<td>%5</td>
</tr>
<tr>
<td>Disjoint Ratio</td>
<td>%8</td>
</tr>
<tr>
<td>Intersecting Ratio</td>
<td>%4</td>
</tr>
<tr>
<td>Overall Mismatch Rate</td>
<td>%8</td>
</tr>
<tr>
<td>Equal Ratio</td>
<td>%66</td>
</tr>
<tr>
<td>Subset Ratio</td>
<td>%26</td>
</tr>
<tr>
<td>Superset Ratio</td>
<td>%6</td>
</tr>
<tr>
<td>Disjoint Ratio</td>
<td>%0</td>
</tr>
<tr>
<td>Intersecting Ratio</td>
<td>%0</td>
</tr>
<tr>
<td>Overall Low Precision Mismatch Rate</td>
<td>%8</td>
</tr>
<tr>
<td>Overall Ambiguous Prediction Mismatch Rate</td>
<td>%2</td>
</tr>
<tr>
<td>Low Precision Mismatch Ratio</td>
<td>%100</td>
</tr>
<tr>
<td>Ambiguous Prediction Mismatch Ratio</td>
<td>%34</td>
</tr>
</tbody>
</table>

Listing 6.1: Results of Report Parameter Analysis

Low Precision Mismatch Ratio is 100% as expected. It means there are no mismatches caused by completely wrong and unrelated predictions. Of all predictions made, mismatches constitutes 8% and ambiguous predictions constitutes only 2%, which is very low, hence negligible given the large number of total predictions made. Lastly, both Overall Mismatch Rate $\rightarrow$ Disjoint Ratio and Overall Mismatch Rate $\rightarrow$ Intersecting Ratio are 0% as expected, complementing what Overall Mismatch Rate suggests.
6.2.2 Cross Parameter Analysis

Cross Parameter Analysis aims to catch any possible pattern caused by certain tick parameter, report parameter configurations as well as proving validity of RPT.

Overall results of cross parameter analysis is as follows:

<table>
<thead>
<tr>
<th>Overall Valid Prediction Ratio</th>
<th>%100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Match Rate</td>
<td>%90</td>
</tr>
<tr>
<td>Equal Ratio</td>
<td>%45</td>
</tr>
<tr>
<td>Subset Ratio</td>
<td>%30</td>
</tr>
<tr>
<td>Superset Ratio</td>
<td>%8</td>
</tr>
<tr>
<td>Disjoint Ratio</td>
<td>%9</td>
</tr>
<tr>
<td>Intersecting Ratio</td>
<td>%5</td>
</tr>
<tr>
<td>Overall Mismatch Rate</td>
<td>%9</td>
</tr>
<tr>
<td>Equal Ratio</td>
<td>%61</td>
</tr>
<tr>
<td>Subset Ratio</td>
<td>%31</td>
</tr>
<tr>
<td>Superset Ratio</td>
<td>%6</td>
</tr>
<tr>
<td>Disjoint Ratio</td>
<td>%0</td>
</tr>
<tr>
<td>Intersecting Ratio</td>
<td>%0</td>
</tr>
<tr>
<td>Overall Low Precision Mismatch Ratio</td>
<td>%9</td>
</tr>
<tr>
<td>Overall Ambiguous Prediction Mismatch Rate</td>
<td>%3</td>
</tr>
<tr>
<td>Low Precision Mismatch Ratio</td>
<td>%100</td>
</tr>
<tr>
<td>Ambiguous Prediction Mismatch Ratio</td>
<td>%38</td>
</tr>
</tbody>
</table>

(Listing 6.2: Results of Cross Parameter Analysis)

Low Precision Mismatch Ratio is 100% as expected. Of all predictions made, mismatches constitutes 9% and ambiguous predictions constitutes only 3% which is again negligible given the number of total predictions made. Both Overall Mismatch Rate $\rightarrow$ Disjoint Ratio and Overall Mismatch Rate $\rightarrow$ Intersection Ratio are 0% again as expected.

6.2.3 Observations

For both analysis, we observed that input files in categories random, simultaneous random, gap = W, simultaneous gap = W have higher rates of mismatches than other categories. We believe this is caused by the fact that these categories assumes no tuple distribution patterns or tuple alignment and solely represent random input streams with constant and varying gaps between tuples. Hence, they are more representative of real world scenarios in predicting performance of RPT. Similarly, window configuration $\beta < \omega \land \omega % \beta \neq 0$ has higher rates of mismatches than others.

We observed that if an engine configuration contains $R_{wc}$ in its report parameters, no matter which tick parameter it has it behaves like a time driven engine. SECRET model assumes windows close only once, therefore $R_{wc}$ reports only once at each window close
This behavior is similar with time driven engines, since they react advancements of application time instead of tuple arrivals or batch arrivals, therefore report once at every advancement of application time provided that reporting condition is fulfilled. This observations can be used to reduce the size of search space in our analysis as follows:

- If opposing engine configurations contain $R_{wc}$ in their report parameters, they both are expected to behave as time driven engines. Therefore, there is no need to test for other tick parameter configurations. Given that there are 6 different tick parameter match-ups, the number of such cases that require testing is reduced to $1/6$ of its value.

- Given a statement, if one of the Statement Terms contains $R_{wc}$ and the other Statement Term has time driven as its tick parameter, both behave as time driven. Assume Left Statement Term contains $R_{wc}$. Then 2 tick parameter combinations, tuple driven vs time driven and batch driven vs time driven contains time driven on their right side. Hence, one of these sets can be omitted which constitutes $1/6$ of all combinations. Same argument applies for the opposing scenario. Therefore, 33% reduction in number of test cases is achieved.

Consider the following example. In 3vs2 form there are $C(4, 3) \times C(4, 2) = 24$ report parameter combinations. $C(3, 2) \times C(3, 1) = 9$ of them contains $R_{wc}$ on both sides and $C(3, 2) \times C(3, 2) + C(3, 3) \times C(3, 1) - 9 = 3$ of them only on one side. Combined with 6 different tick parameter combinations, there are total of 144 test cases including 54 cases with $R_{wc}$ on both sides and 18 cases with only one. The former set of combinations will behave the same for all tick parameter configurations, so it is enough to test them with a single tick parameter combination. Hence, 54 cases can be reduced to 9. For the latter, when repetitive tick parameter combinations are omitted as explained, 18 cases can be reduced to $18 \times 2/3 = 12$. This calculation leaves 93 test cases in total, meaning that search space for this group of statements is now 35% smaller.

For configuration match-ups which have the same report parameters but different tick parameters on opposing sides, we derived the rule set expressing relationships between tick parameters. It can be found in Appendix A, Section A.4.

Last but not least, there are 1080 configuration match-ups in total that we experimented with in both Report Parameter Analysis and Cross Parameter Analysis. Of all, 90 of them corresponds to simplest configuration match-ups, therefore associated with a base rule set. This means, once these 90 cases are processed through simulation and regarding base rule sets are derived, we can predict the evaluation result of remaining 990 cases without simulation or output analysis. Therefore, RPT reduces number of cases that require simulation for derivation of relations by 92% which verifies efficiency claim.
6.3 Further Discussion

Section 2.4 explains boundaries of the target search space we are interested in. In this chapter we showed the validity of RPT by testing it through whole target search space. Overview of our exploration methodology is as follows: First we varied Report parameter across engine configurations where we kept other system parameters (Tick and \( t_0 \)) the same to reveal relationships among different Report parameter values of the opposing sides. For each Report parameter variation, we experimented with a set of query and input parameters so that relationships at important points are revealed expressively. Next, we followed the same track for Tick parameter. Finally, in order to reveal cross interaction between Report and Tick parameters, we varied both across opposing engine configurations while keeping \( t_0 \) the same. We experimented for all possible combinations except the ones we have already tested in previous steps with a set of query and input parameters.

Similar methodology can be used to cover whole target search space to derive complex rules from the base rules using RPT. As a pre-requisite, base rules have to cover all the simplest configurations in the search space. To do that, one should compare each possible combination of single report parameter, tick parameter configurations with each other for a set of different query parameters and input parameters. There are 90 such configuration match-ups. Since relationships are defined based on query parameters and input characteristics, this set should reflect expressive points. The accuracy of our predictions suggest that our choice of query parameters and input tuple distribution patterns are expressive enough to reveal such relationships. Once base rule sets are derived, remaining 990 report parameter, tick parameter configurations can be evaluated through RPT. In our analysis, we do not vary system parameter \( t_0 \) across opposing sides.
Chapter 7

Case Study

7.1 Overview

This chapter presents a case study in which we compared Coral8 [2] and StreamBase [3] through simulation in order to reveal relationships between them.

Case study is constructed as follows: First, we simulated engines with several different window configurations on SSAT (Chapter 3) to explore relationships between them. Then, we generated a prediction set through RPT (Chapter 5) for the same window configurations. Finally, we analyze both relation sets for mismatches to prove accuracy of our predictions.

7.2 Setup

Engine Configurations We will use SECRET parameter configurations of Coral8 and StreamBase in simulation. Our choice is based on the work presented in [1, table 2]. Engine configurations are as follows:

Coral8 :

- Report Parameter: $R_{cc} \wedge R_{ne}$
- Tick Parameter: batch_driven

StreamBase :

- Report Parameter: $R_{wc} \wedge R_{ne}$
- Tick Parameter: tuple_driven
**Window Configuration** We used following patterns to generate random window parameters. Same window parameters were used in simulation of both engine configurations.

- $\beta = 1$ AND $\omega = 1$
- $\beta = 1$ AND $\omega \neq 1$
- $\beta = \omega$
- $\beta < \omega$ AND $\omega \% \beta = 0$
- $\beta < \omega$ AND $\omega \% \beta \neq 0$

**Input Data** For each window parameter configuration, we generated a set of input streams whose tuple distribution follow patterns listed below. Note that we omitted to list simultaneous counterparts of the patterns which contain simultaneous tuples and batches.

- **regular**: 1 input file.
- **gap $=\beta$**: 2 input files, one aligned to window closes ($t_0 + n\beta + \omega$), one aligned to expiration points ($t_0 + n\beta + \omega + 1$).
- **gap $=\omega$**: 1 input file
- **gap $=\text{random}$**: At most 3 input files depending on values of window parameters. Gap values between tuples vary within the following intervals respectively: $[2, \beta], (\beta, \omega], (\omega, 3\omega]$

As presented in [1, table 2], both Coral8 and StreamBase uses the same formulation for calculating $t_0$. Given that we execute them with the same input files, hence the same tuple start time, we expect them to have the same $t_0$. Therefore, we included aligned input sets which would not be possible otherwise.

In total 60 input data streams were generated by SSAT to be used in the case study. Actual parameter values can be found in the next section.

### 7.3 Results and Discussion

Table 7.1 lists the result set acquired through simulation, which reveals the mutual relationships between engine configurations of Coral8 and StreamBase. In the table, row headers specify categories of input files while column headers specify window configurations. Each cell in the table shows the relationship between Coral8 and StreamBase for the corresponding window configuration and input stream.
Likewise, Table 7.2 shows the predicted result set acquired through execution of RPT.

<table>
<thead>
<tr>
<th>Rec.Ruc(bd),Ruc.Ruc(tud)</th>
<th>$\beta = 1 \land \omega = 1$</th>
<th>$\beta = 1 \land \omega &gt; \beta$</th>
<th>$\beta = \omega$</th>
<th>$\beta &lt; \omega \land \omega / \beta = 0$</th>
<th>$\beta &lt; \omega \land \omega / \beta \neq 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>simultaneous.regular</td>
<td>SUPERSET</td>
<td>SUPERSET</td>
<td>SUPERSET</td>
<td>SUPERSET</td>
<td>SUPERSET</td>
</tr>
<tr>
<td>simultaneous_gap = B.expr</td>
<td>SUPERSET</td>
<td>SUPERSET</td>
<td>SUPERSET</td>
<td>SUPERSET</td>
<td>SUPERSET</td>
</tr>
<tr>
<td>simultaneous_gap = B.wc</td>
<td>SUPERSET</td>
<td>SUPERSET</td>
<td>SUPERSET</td>
<td>SUPERSET</td>
<td>SUPERSET</td>
</tr>
<tr>
<td>simultaneous_gap = W</td>
<td>SUPERSET</td>
<td>INTERSECTING</td>
<td>DISJOINT</td>
<td>DISJOINT</td>
<td>DISJOINT</td>
</tr>
<tr>
<td>simultaneous.random</td>
<td>SUPERSET</td>
<td>INTERSECTING</td>
<td>INTERSECTING</td>
<td>INTERSECTING</td>
<td>INTERSECTING</td>
</tr>
<tr>
<td>regular</td>
<td>EQUAL</td>
<td>EQUAL</td>
<td>DISJOINT</td>
<td>DISJOINT</td>
<td>DISJOINT</td>
</tr>
<tr>
<td>gap = B.expr</td>
<td>EQUAL</td>
<td>EQUAL</td>
<td>DISJOINT</td>
<td>DISJOINT</td>
<td>DISJOINT</td>
</tr>
<tr>
<td>gap = B.wc</td>
<td>EQUAL</td>
<td>EQUAL</td>
<td>SUPERSET</td>
<td>SUPERSET</td>
<td>SUPERSET</td>
</tr>
<tr>
<td>gap = W</td>
<td>EQUAL</td>
<td>SUBSET</td>
<td>DISJOINT</td>
<td>DISJOINT</td>
<td>DISJOINT</td>
</tr>
<tr>
<td>random</td>
<td>EQUAL</td>
<td>SUBSET</td>
<td>INTERSECTING</td>
<td>INTERSECTING</td>
<td>INTERSECTING</td>
</tr>
</tbody>
</table>

Table 7.1: Case Study. Calculated Relation Set

We will now list under which conditions which type of relation is expected in calculated relation set presented in Table 7.1.

Conditions should be assessed with the following points in mind:

1. We consider elements of output as a unique entity both with its contents and its scope at the time of reporting. Therefore equality of contents is not enough to label two output elements as the same (Section 4.4.1)

2. We slightly redefined set relations (Section 4.5.1).

. The conditions are as follows:

- For given engine configurations to report at the same time, there has to be a content change at non-empty window closes. Hence, if there are no such content change at any window close, the relation between the result sets of Coral8 and StreamBase will be DISJOINT. For example, tuples of input streams in categories simultaneous_gap = $\beta$.expr and gap = $\beta$.expr, arrive only at expiration points where they also expire. Thus, except cases with $\beta = 1$, where expiration points coincide with window closes, given engine configurations do not report at the same time. As a result, corresponding cells in Table 7.1 is DISJOINT. The case is the same for categories simultaneous_gap = $\omega$ and gap = $\omega$ with one difference,
now the issue occurs by luck since those categories refer random input files with constant gaps.

- If there is at least one non-empty window close where no content change occurs or simultaneous batches exist and if there is at least one content change at a non-empty, non-window close point, then the relation will be INTERSECTING.

- When there are no simultaneous tuples or batches, both Coral8 and StreamBase behave like time-driven systems. So;

  - If there is a content change at all window closes and there are no other content changes at any other time instants, both engines always report at the same time. Therefore, corresponding elements in outputs contain the same time information as well as the same contents. As a result, relation between result sets of Coral8 and StreamBase will be EQUAL. Note that, content changes due to tuple arrivals always fulfill the non-empty condition.

  - If the condition above is relaxed so that content changes can occur also at other time instants in addition to all window close’s, Coral8 will react them given that regarding time instant is non-empty, while StreamBase will not since it is not a window close. Therefore Coral8’s result set will be SUPERSET of StreamBase’s result set.

  - If all content changes occur at window closes, but not at all window closes there is a content change, then Coral8’s result set will be SUBSET of StreamBase’s result set. That is because, as long as they are non-empty \( R_{wc}R_{ne} \) reports at all window closes even if no content change occurs on them, while \( R_{cc}R_{ne} \) does not unless there is a content change.

- When there are simultaneous batches, \( R_{cc}R_{ne} \) reacts to them individually as they arrive, while \( R_{wc}R_{ne} \) reacts to them all together at once, since it always behave like a time-driven engine (remember our observation in Section 6.2.3). This fact in mind, above rules are modified as follows:

  - In addition to the equality rule of non-simultaneous case, there should not be any simultaneous batches at window closes for the relation between results of Coral8 and StreamBase to be EQUAL. That is because, Coral8 reports them individually as they arrive, while StreamBase does not.

  - The SUPERSET rule of non-simultaneous case applies when there are simultaneous batches in the input stream as well.

  - In addition to the SUBSET rule for non-simultaneous case, there should not be any simultaneous batches in the input stream for the relation between results of Coral8 and StreamBase to be SUBSET. Note that in existence
of any simultaneous batches, only Coral8 would report them individually. StreamBase on the other hand, reports them altogether. In addition, it already reports at time instants where Coral8 does not (window closes without a content change). Therefore, the relation would turn into INTERSECTING.

Listing 7.1 lists the statistics about comparison between real and predicted result sets.

All mismatches are due to low precision predictions as indicated by \textit{Low Precision Mismatch Ratio}. Hence, it can be said that prediction algorithm worked nicely. There are no ambiguous predictions as indicated by \textit{Overall Ambiguous Prediction Mismatch Rate} as well as no case where a prediction can not be made as indicated by \textit{Overall Valid Prediction Ratio}. Our results also meet our expectations which we discussed in Section 6.1.2.

Details of mismatches are listed in Listing 7.2.

Finally, Figure 7.1 illustrates the prediction tree generated by RPT during recursive phase. Note that, \textit{tud} is an acronym for tuple driven and \textit{bd} is an acronym for batch driven and they refer to tick parameter value of the corresponding configuration.

Circles with \textit{R} indicates, the algorithm will reverse the evaluation result of corresponding node while passing it to its ancestor. For example, if the node corresponding to statement $R_{cc}(bd) vs R_{wc}R_{ne}(tud)$ is examined closely, it can be seen that it is parsed as if it is $R_{wc}R_{ne}(tud) vs R_{cc}$. Therefore, acquired relation expresses evaluation result of $R_{wc}R_{ne}(tud) vs R_{cc}$ and it has to be reversed to express evaluation result of the original statement (Section 5.5).
Listing 7.2: Case study. Mismatches between real and predicted result sets

The yellow hexagon indicates that RPT does not process regarding sub tree, since it is already evaluated (Section 5.3.1).
Chapter 8

Conclusion and Future Work

8.1 Conclusion

Beyond explaining differences between SPEs, the SECRET model can also be useful for exploring execution model relationships between heterogeneous SPEs. Once discovered, these relationships can be used for variety of purposes such as developing rewrite based query optimizers [5] or devising query transformation rules. Therefore, they are valuable assets. This thesis pursues this research direction and describes a methodology for exploring execution model relationships across different engine configurations by using SECRET model. It also proposes a technique for efficient derivation of such relationships. Additionally, it presents design and architecture of an auxiliary support tool to be used in relationship exploration for analysis, testing and simulation purposes. Furthermore, a prototype implementation of the technique is provided as part of the tool.

Secret Simulation and Analysis Tool is a software built up on a simulator core which is capable of simulating SPEs through SECRET model. It enables comparing different engine configurations via simulation so that mutual relationships can be exploited.

The Relation Prediction Technique proposes a fast and efficient method for exploring execution model relationships between SPEs. It bases its predictions on a known set of relationships among certain subset of available configurations. RPT is able to deduce relationships among the remaining cases on the fly without simulation by using this relationship set. The method reduces processing burden that comes with simulation and analysis phases, therefore offers high efficiency. Within the scope of this work, the technique is thoroughly tested and validated for the complete available configuration space.
8.2 Future Work

While Secret Simulation and Analysis Tool provides an efficient and fast way to derive set of relations between engine configurations, it is still fairly cumbersome process to convert those relations to formal expressions. Therefore, the best possible solution seems to be automating the formulation process. In that regard, a Boolean Logic Module could be added to SSAT, which can derive conditional formal expressions out of produced relation tables automatically.

The Relation Prediction Technique requires parameterized implementation of base rules to function. Current implementation of the technique does not contain base rule implementations, rather it regenerates base relation sets at each invocation. Therefore implementation of base rules is of particular importance at this point of the research. Hence, once a boolean logic module is implemented, a complementary Code Generator Module, which is capable of generating code blocks for the predefined formal structures produced by the logic module, could be developed.

Last but not least, the base rule inventory of RPT should be enriched with missing sets of base rules so that the technique can work with full capacity.
Appendix A

Base Rules

This appendix lists the three base rule sets we derived. Further details can be found in Chapter 4.

Base rule sets are categorized by their tick parameter configurations, namely: *tuple-driven vs. tuple-driven, time-driven vs. time-driven* and *batch-driven vs. batch-driven.* Each section is divided into subsections by corresponding report parameter settings. Rules in subsections are listed by the relation they correspond to. For example, the very first rule in section A.1, subsection A.1.1 gives the condition for equality between engines $E_1 = \text{tick: tuple-driven, report: } R_{cc}$ and $E_2 = \text{tick: tuple-driven, report: } R_{wc}.$

For ease of use common constructs used in formulas are as listed below:

<table>
<thead>
<tr>
<th>Construct</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window open</td>
<td>$t_0 + n\beta$</td>
</tr>
<tr>
<td>Window close</td>
<td>$t_0 + n\beta + \omega$</td>
</tr>
<tr>
<td>Expiration Points</td>
<td>$t_0 + n\beta + \omega + 1$</td>
</tr>
<tr>
<td>Periodic Reporting Points</td>
<td>$k\lambda$</td>
</tr>
<tr>
<td>No simultaneous tuples</td>
<td>$\forall t \mid</td>
</tr>
<tr>
<td>No simultaneous batches</td>
<td>$\forall t \mid</td>
</tr>
<tr>
<td>No tuples in between points P</td>
<td>$\forall t \neq P \land S_i(t) = \emptyset$</td>
</tr>
<tr>
<td>Simultaneous batches with size 1</td>
<td>$</td>
</tr>
</tbody>
</table>

Table A.1: Common constructs used in base rule formulation
Appendix A. Base Rules

A.1 Tick Parameter: tuple_driven

A.1.1 \( R_{cc} \) vs. \( R_{wc} \)

\[
R_{cc} = R_{wc}
\]

\[
\forall n \ S_i(t_0 + n\beta + \omega) \neq \emptyset \ \land \ \beta = 1 \ \land \ \forall t \ \mid S_i(t) \mid \leq 1 \quad (A.1)
\]

\[
R_{cc} \supset R_{wc}
\]

\[
\forall n \ S_i(t_0 + n\beta + \omega) \neq \emptyset \quad (A.2)
\]

\[
R_{cc} \subset R_{wc}
\]

\[
\exists n \ S_i(t_0 + n\beta + \omega) \neq \emptyset \ \land \ \beta = 1 \ \land \ \forall t \ \mid S_i(t) \mid \leq 1 \quad (A.3)
\]

\[
R_{cc} \cap R_{wc} = \emptyset
\]

\[
\forall n \ S_i(t_0 + n\beta + \omega) = \emptyset \ \land \ \beta \neq 1 \quad (A.4)
\]

A.1.2 \( R_{cc} \) vs. \( R_{ne} \)

\[
R_{cc} = R_{ne}
\]

\[
\forall t \ S_i(t) \neq \emptyset \quad (A.5)
\]

\[
R_{cc} \supset R_{ne}
\]

\[
\forall t \ S_i(t) \neq \emptyset
\]

\[
\lor
\]

\[
[ \beta = \omega \ \land \ \exists n \ S_i(t_0 + n\beta + \omega) \neq \emptyset
\]

\[
\land
\]

\[
( \forall t \ t \neq t_0 + n\beta + \omega \ \land \ S_i(t) = \emptyset ) \quad (A.6)
\]
$R_{cc} \subset R_{ne}$

$$\forall n \ S( t_0 + (n + 1)\beta ) - S( t_0 + n\beta ) \neq \emptyset$$

$$\wedge$$

$$S( t_0 + n\beta + w ) - S( t_0 + (n + 1)\beta ) = \emptyset$$

$$\wedge$$

$$S_i( t_0 + n\beta + w + 1 ) \neq \emptyset$$

(A.7)

$R_{cc} \cap R_{ne} = \emptyset$

No rule

(A.8)

A.1.3 $R_{cc}$ vs. $R_{pr}$

$R_{cc} = R_{pr}$

$$\forall n \exists k \ k\lambda = t_0 + n\beta + \omega + 1 \ \wedge \ ( \forall t \neq k\lambda \wedge S_i(t) = \emptyset )$$

$$\wedge$$

$$[ \ \forall k\exists n \ ( k\lambda = t_0 + n\beta + w + 1 \ \wedge \ S( t_0 + (n + 1)\beta ) - S( t_0 + n\beta ) = \emptyset$$

$$\wedge \ S_i( k\lambda ) \neq \emptyset ) \ \vee \ ( k\lambda \neq t_0 + n\beta + w + 1 \ \wedge \ S_i( k\lambda ) \neq \emptyset ) ]$$

(A.9)

$R_{cc} \supset R_{pr}$

$$\forall n \exists k \ k\lambda = t_0 + n\beta + \omega + 1$$

$$[ \ \forall k\exists n \ ( k\lambda = t_0 + n\beta + w + 1 \ \wedge \ S( t_0 + (n + 1)\beta ) - S( t_0 + n\beta ) = \emptyset$$

$$\wedge \ S_i( k\lambda ) \neq \emptyset ) \ \vee \ ( k\lambda \neq t_0 + n\beta + w + 1 \ \wedge \ S_i( k\lambda ) \neq \emptyset ) ]$$

(A.10)

$R_{cc} \subset R_{pr}$

$$\forall n \exists k \ k\lambda = t_0 + n\beta + \omega + 1 \ \wedge \ ( \forall t \neq k\lambda \wedge S_i(t) = \emptyset )$$

(A.11)

$R_{cc} \cap R_{pr} = \emptyset$

$$\forall n \exists k \ k\lambda \neq t_0 + n\beta + \omega + 1 \ \wedge \ ( \forall t = k\lambda \wedge S_i(t) = \emptyset )$$

(A.12)
A.1.4  \( R_{wc} \) vs. \( R_{ne} \)

\( R_{wc} = R_{ne} \)

\[
\forall n \ S( t_0 + n\beta + \omega ) - S( t_0 + n\beta ) \neq \emptyset \quad \land \quad \forall t \quad | S_i( t ) | \leq 1
\quad \land
\quad ( \forall t \neq t_0 + n\beta + \omega \quad \land \quad S_i( t ) = \emptyset )
\quad \land
\quad ( \beta = \omega \lor \beta = 1 ) \quad (A.13)
\]

\( R_{wc} \supset R_{ne} \)

\[
\exists n \ S_i( t_0 + n\beta + \omega ) \neq \emptyset \quad \land \quad \forall t \quad | S_i( t ) | \leq 1
\quad \land
\quad ( \forall t \neq t_0 + n\beta + \omega \quad \land \quad S_i( t ) = \emptyset )
\quad \land
\quad ( \beta = \omega \lor \beta = 1 ) \quad (A.14)
\]

\( R_{wc} \subset R_{ne} \)

\[
\forall n \ S( t_0 + n\beta + \omega ) - S( t_0 + n\beta ) \neq \emptyset \quad (A.15)
\]

\( R_{wc} \cap R_{ne} = \emptyset \)

No rule \quad (A.16)

A.1.5  \( R_{wc} \) vs. \( R_{pr} \)

\( R_{wc} = R_{pr} \)

\[
\forall n \ \exists k \quad t_0 + n\beta + \omega = k\lambda \quad \land \quad \exists k \ k\lambda \neq t_0 + n\beta + \omega \quad \land \quad \forall k \quad | S_i( k\lambda ) | \leq 1
\quad (A.17)
\]

\( R_{wc} \supset R_{pr} \)

\[
\forall k \ \exists n \quad t_0 + n\beta + \omega = k\lambda \quad \land \quad \forall k \quad | S_i( k\lambda ) | \leq 1 \quad (A.18)
\]
Appendix A. Base Rules

\[ R_{wc} \subset R_{pr} \]
\[ \forall n \exists k \ t_0 + n\beta + \omega = k\lambda \quad (A.19) \]

\[ R_{wc} \cap R_{pr} = \emptyset \]
\[ \forall n \ \nexists k \ t_0 + n\beta + \omega = k\lambda \quad (A.20) \]

A.1.6 \( R_{ne} \) vs. \( R_{pr} \)

\( R_{ne} = R_{pr} \)
\[ \forall k \exists n \ S(k\lambda) - S(t_0 + n\beta) \neq \emptyset \]
\[ \land \]
\[ ( \forall t \neq k\lambda \land S_i(t) = \emptyset ) \land \]
\[ [ (\beta = \omega \land k\lambda = t_0 + n\beta + \omega) \lor \lambda = 1 ] \quad (A.21) \]

\( R_{ne} \supset R_{pr} \)
\[ \forall k \exists n \ S(k\lambda) - S(t_0 + n\beta) \neq \emptyset \quad (A.22) \]

\( R_{ne} \subset R_{pr} \)
\[ \lambda = 1 \]
\[ \lor \]
\[ [ \beta = \omega \land \forall k \exists n \ t_0 + n\beta + \omega = k\lambda \land ( \forall t \neq k\lambda \land S_i(t) = \emptyset ) ] \quad (A.23) \]

\( R_{ne} \cap R_{pr} = \emptyset \)
\[ \forall n \exists k \ S(k\lambda) - S(t_0 + n\beta) = \emptyset \land \lambda \neq 1 \quad (A.24) \]
Appendix A. Base Rules

A.2 Tick Parameter: time\_driven

A.2.1 $R_{cc}$ vs. $R_{wc}$

$R_{cc} = R_{wc}$

$$\forall n S_i(t_0 + n\beta + \omega) \neq \emptyset \land \beta = 1$$ (A.25)

$R_{cc} \supset R_{wc}$

$$\forall n S_i(t_0 + n\beta + \omega) \neq \emptyset$$ (A.26)

$R_{cc} \subset R_{wc}$

$$\exists n S_i(t_0 + n\beta + \omega) \neq \emptyset \land \beta = 1$$ (A.27)

$R_{cc} \cap R_{wc} = \emptyset$

$$\forall n (S_i(t_0 + n\beta + \omega) = \emptyset \land \beta \neq 1)$$ (A.28)

A.2.2 $R_{cc}$ vs. $R_{ne}$

$R_{cc} = R_{ne}$

$$\forall t S_i(t) \neq \emptyset$$ (A.29)

$R_{cc} \supset R_{ne}$

$$\forall t S_i(t) \neq \emptyset$$

$$\lor$$

$$[ \beta = \omega \land \exists n S_i(t_0 + n\beta + \omega) \neq \emptyset$$

$$\land$$

$$[ \forall t t \neq t_0 + n\beta + \omega \land S_i(t) = \emptyset ]$$ (A.30)
Appendix A. Base Rules

\[ R_{cc} \subset R_{ne} \]

\[
\forall n \ S( t_0 + (n+1)\beta ) - S( t_0 + n\beta ) \neq \emptyset \\
\land \\
S( t_0 + n\beta + w ) - S( t_0 + (n+1)\beta ) = \emptyset \\
\land \\
S_i( t_0 + n\beta + w + 1 ) \neq \emptyset \tag{A.31}
\]

\[ R_{cc} \cap R_{ne} = \emptyset \]

No rule \tag{A.32}

A.2.3 \quad R_{cc} \text{ vs. } R_{pr}

\[ R_{cc} = R_{pr} \]

\[
\forall n \exists k \ k\lambda = t_0 + n\beta + \omega + 1 \land ( \forall t \neq k\lambda \land S_i(t) = \emptyset ) \\
\land \\
[ \forall k\in\mathbb{N} \ ( k\lambda = t_0 + n\beta + w + 1 \land S( t_0 + (n+1)\beta ) - S( t_0 + n\beta ) = \emptyset \\
\land \ S_i( k\lambda ) \neq \emptyset ) \lor ( k\lambda \neq t_0 + n\beta + w + 1 \land S_i( k\lambda ) \neq \emptyset ) ] \tag{A.33}
\]

\[ R_{cc} \supset R_{pr} \]

\[
\forall n \exists k \ k\lambda = t_0 + n\beta + \omega + 1 \\
[ \forall k\in\mathbb{N} \ ( k\lambda = t_0 + n\beta + w + 1 \land S( t_0 + (n+1)\beta ) - S( t_0 + n\beta ) = \emptyset \\
\land \ S_i( k\lambda ) \neq \emptyset ) \lor ( k\lambda \neq t_0 + n\beta + w + 1 \land S_i( k\lambda ) \neq \emptyset ) ] \tag{A.34}
\]

\[ R_{cc} \subset R_{pr} \]

\[
\forall n \exists k \ k\lambda = t_0 + n\beta + \omega + 1 \land ( \forall t \neq k\lambda \land S_i(t) = \emptyset ) \tag{A.35}
\]

\[ R_{cc} \cap R_{pr} = \emptyset \]

\[
\forall n \exists k \ k\lambda \neq t_0 + n\beta + \omega + 1 \land ( \forall t = k\lambda \land S_i(t) = \emptyset ) \tag{A.36}
\]
A.2.4 $R_{wc}$ vs. $R_{ne}$

$R_{wc} = R_{ne}$

\[
\forall n \ S( t_0 + n\beta + \omega ) - S( t_0 + n\beta ) \neq \emptyset \\
\land \\
( \forall t \neq t_0 + n\beta + \omega \land S_i(t) = \emptyset ) \\
\land \\
( \beta = \omega \lor \beta = 1 ) \tag{A.37}
\]

$R_{wc} \supset R_{ne}$

\[
\exists n \ S_i(t_0 + n\beta + \omega) \neq \emptyset \\
\land \\
( \forall t \neq t_0 + n\beta + \omega \land S_i(t) = \emptyset ) \\
\land \\
( \beta = \omega \lor \beta = 1 ) \tag{A.38}
\]

$R_{wc} \subset R_{ne}$

\[
\forall n \ S( t_0 + n\beta + \omega ) - S( t_0 + n\beta ) \neq \emptyset \tag{A.39}
\]

$R_{wc} \cap R_{ne} = \emptyset$

No rule \tag{A.40}

A.2.5 $R_{wc}$ vs. $R_{pr}$

$R_{wc} = R_{pr}$

\[
\forall n \ \exists k \ t_0 + n\beta + \omega = k\lambda \land \not\exists k \ k\lambda \neq t_0 + n\beta + \omega \tag{A.41}
\]

$R_{wc} \supset R_{pr}$

\[
\forall k \ \exists n \ t_0 + n\beta + \omega = k\lambda \tag{A.42}
\]

$R_{wc} \subset R_{pr}$

\[
\forall n \ \exists k \ t_0 + n\beta + \omega = k\lambda \tag{A.43}
\]
Appendix A. Base Rules

\( R_{wc} \cap R_{pr} = \emptyset \)

\[ \forall n \not\exists k \ t_0 + n\beta + \omega = k\lambda \]  \hspace{1cm} (A.44)

A.2.6 \( R_{ne} \) vs. \( R_{pr} \)

\( R_{ne} = R_{pr} \)

\[ \forall k \exists n \ S(k\lambda) - S(t_0 + n\beta) \neq \emptyset \]

\[ \land \]

\[ ( \forall t \neq k\lambda \land S_i(t) = \emptyset ) \]

\[ \land \]

\[ [ (\beta = \omega \land k\lambda = t_0 + n\beta + \omega) \lor \lambda = 1 ] \]  \hspace{1cm} (A.45)

\( R_{ne} \supset R_{pr} \)

\[ \forall k \exists n \ S(k\lambda) - S(t_0 + n\beta) \neq \emptyset \]  \hspace{1cm} (A.46)

\( R_{ne} \subset R_{pr} \)

\[ \lambda = 1 \]

\[ \lor \]

\[ [ \beta = \omega \land \forall k \exists n \ t_0 + n\beta + \omega = k\lambda \land ( \forall t \neq k\lambda \land S_i(t) = \emptyset ) ] \]

\hspace{1cm} (A.47)

\( R_{ne} \cap R_{pr} = \emptyset \)

\[ \forall n \exists k \ S(k\lambda) - S(t_0 + n\beta) = \emptyset \land \lambda \neq 1 \]  \hspace{1cm} (A.48)
A.3 Tick Parameter: batch\_driven

A.3.1 $R_{cc}$ vs. $R_{wc}$

$R_{cc} = R_{wc}$

\[
\forall n \ S_i(t_0 + n\beta + \omega) \neq \emptyset \land \beta = 1 \land \forall t \ | S_{bid}(t) | = 1 \quad (A.49)
\]

$R_{cc} \supset R_{wc}$

\[
\forall n \ S_i(t_0 + n\beta + \omega) \neq \emptyset \quad (A.50)
\]

$R_{cc} \subset R_{wc}$

\[
\exists n \ S_i(t_0 + n\beta + \omega) \neq \emptyset \land \beta = 1 \land \forall t \ | S_{bid}(t) | = 1 \quad (A.51)
\]

$R_{cc} \cap R_{wc} = \emptyset$

\[
\forall n \ S_i(t_0 + n\beta + \omega) = 0 \land \beta \neq 1 \quad (A.52)
\]

A.3.2 $R_{cc}$ vs. $R_{ne}$

$R_{cc} = R_{ne}$

\[
\forall t \ S_i(t) \neq \emptyset \quad (A.53)
\]

$R_{cc} \supset R_{ne}$

\[
\forall t \ ( S_i(t) \neq \emptyset ) \\
\lor \\
[ \beta = \omega \land \exists n \ S_i(t_0 + n\beta + \omega) \neq \emptyset \\
\land \\
( \forall t \ t \neq t_0 + n\beta + \omega \land S_i(t) = \emptyset ) ] \quad (A.54)
\]
**Appendix A. Base Rules**

\[ R_{cc} \subset R_{ne} \]

\[ \forall n \ S( t_0 + (n + 1)\beta ) - S( t_0 + n\beta ) \neq \emptyset \]
\[ \land \]
\[ S( t_0 + n\beta + w ) - S( t_0 + (n + 1)\beta ) = \emptyset \]
\[ \land \]
\[ S_i( t_0 + n\beta + w + 1 ) \neq \emptyset \] (A.55)

\[ R_{cc} \cap R_{ne} = \emptyset \]

No rule (A.56)

**A.3.3 \quad R_{cc} vs. \quad R_{pr}**

\[ R_{cc} = R_{pr} \]

\[ \forall n \exists k \ k\lambda = t_0 + n\beta + \omega + 1 \land ( \forall t \neq k\lambda \land S_i(t) = \emptyset ) \]
\[ \land \]
\[ [ \forall k\exists n \ ( k\lambda = t_0 + n\beta + w + 1 \land S( t_0 + (n + 1)\beta ) - S( t_0 + n\beta ) = \emptyset \]
\[ \land \ S_i( k\lambda ) \neq \emptyset ) \lor ( k\lambda \neq t_0 + n\beta + w + 1 \land S_i( k\lambda ) \neq \emptyset ) ] \] (A.57)

\[ R_{cc} \supset R_{pr} \]

\[ \forall n \exists k \ k\lambda = t_0 + n\beta + \omega + 1 \]
\[ [ \forall k\exists n \ ( k\lambda = t_0 + n\beta + w + 1 \land S( t_0 + (n + 1)\beta ) - S( t_0 + n\beta ) = \emptyset \]
\[ \land \ S_i( k\lambda ) \neq \emptyset ) \lor ( k\lambda \neq t_0 + n\beta + w + 1 \land S_i( k\lambda ) \neq \emptyset ) ] \] (A.58)

\[ R_{cc} \subset R_{pr} \]

\[ \forall n \exists k \ k\lambda = t_0 + n\beta + \omega + 1 \land ( \forall t \neq k\lambda \land S_i(t) = \emptyset ) \] (A.59)

\[ R_{cc} \cap R_{pr} = \emptyset \]

\[ \forall n \exists k \ k\lambda \neq t_0 + n\beta + \omega + 1 \land ( \forall t = k\lambda \land S_i(t) = \emptyset ) \] (A.60)
A.3.4  \(R_{we}\) vs. \(R_{ne}\)

\[R_{we} = R_{ne}\]

\[\forall n \ S( t_0 + n\beta + \omega ) - S( t_0 + n\beta ) \neq \emptyset \quad \land \quad \forall t \mid S_{bid}( t ) \mid = 1 \quad \land \quad \left( \forall t \neq t_0 + n\beta + \omega \land S_i(t) = \emptyset \right) \quad \land \quad \left( \beta = \omega \lor \beta = 1 \right)\]

(A.61)

\[R_{we} \supset R_{ne}\]

\[\exists n \ S_i(t_0 + n\beta + \omega) \neq \emptyset \quad \land \quad \forall t \mid S_{bid}( t ) \mid = 1 \quad \land \quad \left( \forall t \neq t_0 + n\beta + \omega \land S_i(t) = \emptyset \right) \quad \land \quad \left( \beta = \omega \lor \beta = 1 \right)\]

(A.62)

\[R_{we} \subset R_{ne}\]

\[\forall n \ S( t_0 + n\beta + \omega ) - S( t_0 + n\beta ) \neq \emptyset \]

(A.63)

\[R_{we} \cap R_{ne} = \emptyset\]

No rule

(A.64)

A.3.5  \(R_{we}\) vs. \(R_{pr}\)

\[R_{we} = R_{pr}\]

\[\forall n \ \exists k \ t_0 + n\beta + \omega = k\lambda \quad \land \quad \nexists k \ k\lambda \neq t_0 + n\beta + \omega \quad \land \quad \forall t \mid S_{bid}( t ) \mid = 1\]

(A.65)

\[R_{we} \supset R_{pr}\]

\[\forall k \ \exists n \ t_0 + n\beta + \omega = k\lambda \quad \land \quad \forall t \mid S_{bid}( t ) \mid = 1\]

(A.66)

\[R_{we} \subset R_{pr}\]

\[\forall n \ \exists k \ t_0 + n\beta + \omega = k\lambda\]

(A.67)

\[R_{we} \cap R_{pr} = \emptyset\]

\[\forall n \ \nexists k \ t_0 + n\beta + \omega = k\lambda\]

(A.68)
A.3.6  $R_{ne}$ vs. $R_{pr}$

$R_{ne} = R_{pr}$

$$\forall k \exists n \ S(k\lambda) - S(t_0 + n\beta) \neq \emptyset \wedge \ (\forall t \neq k\lambda \wedge S_i(t) = \emptyset)$$ \wedge [ (\beta = \omega \wedge k\lambda = t_0 + n\beta + \omega ) \vee \lambda = 1 ] \quad \text{(A.69)}$$

$R_{ne} \supset R_{pr}$

$$\forall k \exists n \ S(k\lambda) - S(t_0 + n\beta) \neq \emptyset$$ \quad \text{(A.70)}

$R_{ne} \subset R_{pr}$

$$\lambda = 1 \vee [ \beta = \omega \wedge \forall k \exists n \ t_0 + n\beta + \omega = k\lambda \wedge (\forall t \neq k\lambda \wedge S_i(t) = \emptyset) ]$$ \quad \text{(A.71)}

$R_{ne} \cap R_{pr} = \emptyset$

$$\forall n \exists k \ S(k\lambda) - S(t_0 + n\beta) = \emptyset \wedge \lambda \neq 1$$ \quad \text{(A.72)}

A.4  Tick Parameter Equivalences

Assume we are comparing two engine configurations $E_1, E_2$ with report parameters $R_1, R_2$ and tick parameters $T_1, T_2$ respectively.

$$E_1 \leftarrow R_1, T_1 \hspace{1em} \text{vs} \hspace{1em} E_2 \leftarrow R_2, T_2$$

This section lists formal rules which express evaluation results of the above comparison where opposing engine configurations follow the pattern stated below:

$$R_1 = R_2 \text{ AND } T_1 \neq T_2$$
Appendix A. Base Rules

\( \mathcal{T}_1 = \text{Tuple\_driven vs. } \mathcal{T}_2 = \text{Time\_driven} \)

\( R_1 = R_2 \xleftarrow{\text{R}_{cc}} \)

\[ \exists t \ | S_i(t) | \geq 1 \rightarrow E_1 \supset E_2 \]  \hspace{1cm} (A.73)

\[ \forall t \ | S_i(t) | \leq 1 \rightarrow E_1 = E_2 \]  \hspace{1cm} (A.74)

\( R_1 = R_2 \xleftarrow{\text{R}_{ne}} \)

Same as \( R_1 = R_2 \xleftarrow{\text{R}_{cc}} \)

\( R_1 = R_2 \xleftarrow{\text{R}_{pr}} \)

\[ \exists k \ | S_i(k\lambda) | \geq 1 \rightarrow E_1 \supset E_2 \]  \hspace{1cm} (A.75)

\[ \forall k \ | S_i(k\lambda) | \leq 1 \rightarrow E_1 = E_2 \]  \hspace{1cm} (A.76)

\( R_1 = R_2 \xleftarrow{\text{R}_{wc}} \)

Always \( E_1 = E_2 \)  \hspace{1cm} (A.77)

\( \mathcal{T}_1 = \text{Tuple\_driven vs. } \mathcal{T}_2 = \text{Batch\_driven} \)

\( R_1 = R_2 \xleftarrow{\text{R}_{cc}} \)

\[ \exists t \ \text{maxBatchSize}(t) \geq 1 \rightarrow E_1 \supset E_2 \]  \hspace{1cm} (A.78)

\[ \forall t \ \text{maxBatchSize}(t) \leq 1 \rightarrow E_1 = E_2 \]  \hspace{1cm} (A.79)
Appendix A. Base Rules

\[ R_1 = R_2 \leftarrow R_{ne} \]

Same as \( R_1 = R_2 \leftarrow R_{cc} \)

\[ R_1 = R_2 \leftarrow R_{pr} \]

\[ \exists k \族群\text{maxBatchSize}(k\lambda) \geq 1 \rightarrow E_1 \supset E_2 \quad (A.80) \]

\[ \forall k \族群\text{maxBatchSize}(k\lambda) \leq 1 \rightarrow E_1 = E_2 \quad (A.81) \]

\[ R_1 = R_2 \leftarrow R_{wc} \]

Always \( E_1 = E_2 \) \quad (A.82)

\[ T_1 = \text{Time Driven vs. } T_2 = \text{Batch Driven} \]

\[ R_1 = R_2 \leftarrow R_{cc} \]

\[ \exists t \mid S_{bid}(t) \mid \geq 1 \rightarrow E_1 \subset E_2 \quad (A.83) \]

\[ \forall t \mid S_{bid}(t) \mid \leq 1 \rightarrow E_1 = E_2 \quad (A.84) \]

\[ R_1 = R_2 \leftarrow R_{ne} \]

Same as \( R_1 = R_2 \leftarrow R_{cc} \)

\[ R_1 = R_2 \leftarrow R_{pr} \]

\[ \exists k \mid S_{bid}(k\lambda) \mid \geq 1 \rightarrow E_1 \subset E_2 \quad (A.85) \]
\[
\forall k \ |S_{bid}(k\lambda)| \leq 1 \implies E_1 = E_2 \quad (A.86)
\]

\[
R_1 = R_2 \leftarrow R_{wc}
\]

Always \( E_1 = E_2 \) \quad (A.87)
Appendix B

Prediction Table

This appendix presents the Prediction Table used by RPT algorithm (Chapter 5) in Section B.1. Validation of table data can be found in Section B.2.

B.1 Prediction Table

<table>
<thead>
<tr>
<th>$R_1$ vs. $R_3$</th>
<th>$R_2$ vs. $R_3$</th>
<th>$R_1$ vs. $R_2$</th>
<th>$R_1 \cap R_2$ vs. $R_3$</th>
<th>Type</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQUAL</td>
<td>EQUAL</td>
<td></td>
<td>EQUAL</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>EQUAL</td>
<td>SUPERSET</td>
<td>-</td>
<td>EQUAL</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>EQUAL</td>
<td>SUBSET</td>
<td>-</td>
<td>SUBSET</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>EQUAL</td>
<td>DISJOINT</td>
<td>-</td>
<td>DISJOINT</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>EQUAL</td>
<td>INTERSECTING</td>
<td>-</td>
<td>SUBSET</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>DISJOINT</td>
<td>EQUAL</td>
<td>-</td>
<td>DISJOINT</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>DISJOINT</td>
<td>SUPERSET</td>
<td>-</td>
<td>DISJOINT</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>DISJOINT</td>
<td>SUBSET</td>
<td>-</td>
<td>DISJOINT</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>DISJOINT</td>
<td>DISJOINT</td>
<td>-</td>
<td>DISJOINT</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>DISJOINT</td>
<td>INTERSECTING</td>
<td>-</td>
<td>DISJOINT</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>SUBSET</td>
<td>EQUAL</td>
<td>-</td>
<td>SUBSET</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>SUBSET</td>
<td>SUPERSET</td>
<td>-</td>
<td>SUBSET</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>SUBSET</td>
<td>SUBSET</td>
<td>DISJOINT</td>
<td>DISJOINT</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>+</td>
<td>SUBSET</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>SUBSET</td>
<td>DISJOINT</td>
<td>-</td>
<td>DISJOINT</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>SUBSET</td>
<td>INTERSECTING</td>
<td>DISJOINT</td>
<td>DISJOINT</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>+</td>
<td>SUBSET</td>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>

82
### Table B.1: Prediction Table for RPT algorithm

<table>
<thead>
<tr>
<th>$R_1$ vs. $R_3$</th>
<th>$R_2$ vs. $R_3$</th>
<th>$R_1$ vs. $R_2$</th>
<th>$R_1 \cap R_2$ vs $R_3$</th>
<th>Type</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUPERSET</td>
<td>EQUAL</td>
<td>-</td>
<td>EQUAL</td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>SUPERSET</td>
<td>SUPERSET</td>
<td>INTERSECTING</td>
<td>EQUAL SUPERSET</td>
<td>LP</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+</td>
<td>SUPERSET</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>SUPERSET</td>
<td>SUBSET</td>
<td>-</td>
<td>SUBSET</td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>SUPERSET</td>
<td>DISJOINT</td>
<td>-</td>
<td>DISJOINT</td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>SUPERSET</td>
<td>INTERSECTING</td>
<td>INTERSECTING</td>
<td>SUBSET INTERSECTING</td>
<td>LP</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+</td>
<td>INTERSECTING</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>INTERSECTING</td>
<td>EQUAL</td>
<td>-</td>
<td>SUBSET</td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>INTERSECTING</td>
<td>SUPERSET</td>
<td>INTERSECTING</td>
<td>SUBSET INTERSECTING</td>
<td>LP</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+</td>
<td>INTERSECTING</td>
<td></td>
<td>27</td>
</tr>
<tr>
<td>INTERSECTING</td>
<td>SUBSET</td>
<td>DISJOINT</td>
<td>DISJOINT</td>
<td></td>
<td>28</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+</td>
<td>SUBSET</td>
<td></td>
<td>29</td>
</tr>
<tr>
<td>INTERSECTING</td>
<td>DISJOINT</td>
<td>-</td>
<td>DISJOINT</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>INTERSECTING</td>
<td>INTERSECTING</td>
<td>INTERSECTING</td>
<td>SUBSET INTERSECTING DISJOINT</td>
<td>AP</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+</td>
<td>INTERSECTING</td>
<td></td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>33</td>
</tr>
</tbody>
</table>

### Legend

- : Any Possible Value  
+ : All Others  
LP : Low Precision Prediction  
AP : Ambiguous Prediction  
R : Result Set  

### Table B.2: Prediction Table Legend

#### B.2 Table Validation

This section validates the correctness of relation combinations presented in the Prediction Table B.1 through set theory.
Following proposition is used extensively:

Let $A$, $B$ and $C$ be sets. If $A \subseteq B$ then $A \cup B = B$ and $A \cap B = A$ \hfill (B.1)

Validations are listed by the line number they correspond to as follows:

(+) Line: 1

\[
A = C \quad \land \quad B = C \rightarrow \ A \cap B = C
\]
\[
A = C \quad \land \quad B = C \rightarrow A = B
\] (transitivity) \hfill (1)
\[
A \cap B = C \iff C \cap B = C
\]
\[
\iff B \cap B = C
\] (by $A = C$)
\[
\iff B = C
\] (by $B = C$)
\[
\iff C = C
\] (idempotent law)

(+ Line: 2, 18

\[
A = C \quad \land \quad B \supset C \rightarrow (A \cap B) = C
\]
\[
A = C \quad \land \quad B \supset C \rightarrow B \supset A
\] (1)
\[
A \subseteq B \rightarrow (A \cap B) = A
\] (by B.1) \hfill (2)
\[
(A \cap B) = C \iff A = C
\] (by 2)
\[
\iff C = C
\]

(+ Line: 3, 11

\[
A = C \quad \land \quad B \subset C \rightarrow (A \cap B) \subset C
\]
\[
A = C \quad \land \quad B \subset C \rightarrow B \subset A
\] (1)
\[
A \supset B \rightarrow (A \cap B) = B
\] (by B.1) \hfill (2)
\[
(A \cap B) \subset C \rightarrow (A \cap B) \cap C = A \cap B
\] (by B.1)
\[
(A \cap B) \cap C = A \cap B \iff (B) \cap C = A \cap B
\] (by 2)
\[
\iff B \cap A = A \cap B
\]
Lemma: If any two sets are DISJOINT, then the relation between intersection of any two sets and the remaining set will be DISJOINT.

\[
A \cap B = \emptyset \rightarrow A \cap B \cap C = \emptyset
\]

\[
A \cap B \cap C = (A \cap B) \cap C
\]

\[
= \emptyset \cap C
\]

\[
= \emptyset
\]

(associative law)

(domination law)

\[
B \cap C = \emptyset \rightarrow A \cap B \cap C = \emptyset
\]

\[
A \cap B \cap C = A \cap (B \cap C)
\]

\[
= A \cap \emptyset
\]

\[
= \emptyset
\]

(associative law)

(domination law)

\[
A \cap C = \emptyset \rightarrow A \cap B \cap C = \emptyset
\]

\[
A \cap B \cap C = A \cap C \cap B
\]

\[
= (A \cap C) \cap B
\]

\[
= \emptyset \cap B
\]

\[
= \emptyset
\]

(commutative law)

(associative law)

(domination law)

(+ Line: 5, 25)

(i) Validation for column 3.

\[
A = C \land (B \cap C \neq \emptyset \land B \not\subseteq C \land B \not\supset C) \rightarrow (B \cap A \neq \emptyset \land B \not\subseteq A \land B \not\supset A)
\]  

(ii) Validation for prediction, column 4.

\[
A = C \land (B \cap C \neq \emptyset \land B \not\subseteq C \land B \not\supset C) \rightarrow (A \cap B) \subset C
\]
Appendix B. Prediction Table

Assume \((B \cap C) = \emptyset\);
\[
(A \cap B) \subset C \rightarrow (A \cap B) \cap C = A \cap B \quad \text{(by B.1)}
\]
\[
(A \cap B) \cap C = A \cap B \iff A \cap (B \cap C) = A \cap B \quad \text{(associative law)}
\]
\[
\iff A \cap \emptyset = A \cap B
\]
\[
\iff \emptyset = A \cap B \quad \text{(domination law)}
\]
Contradicts with (1), thus \(B \cap C \neq \emptyset\) and proposition holds.

(+ Line: 12, 21)

\[
A \subset C \land B \supset C \rightarrow (A \cap B) \subset C
\]
\[
A \subset C \land B \supset C \rightarrow B \supset A \quad \text{(transitivity)} \quad (1)
\]
\[
A \subset B \rightarrow A \cap B = A \quad \text{(by B.1)} \quad (2)
\]
\[
(A \cap B) \subset C \iff (A) \subset C \quad \text{(by 2)}
\]
Already given.

(+ Line: 14)

\[
A \subset C \land B \subset C \rightarrow (A \cap B) \subset C
\]
\[
A \subset C \rightarrow A \cap C = A \quad \text{(by B.1)} \quad (1)
\]
\[
B \subset C \rightarrow B \cap C = B \quad \text{(by B.1)} \quad (2)
\]
\[
(A \cap B) \subset C \rightarrow (A \cap B) \cap C = A \cap B \quad \text{(by B.1)}
\]
\[
A \cap B \cap C = A \cap B \iff A \cap B \cap C \cap C = A \cap B \quad \text{(idempotent law)}
\]
\[
\iff (A \cap C) \cap (B \cap C) = A \cap B \quad \text{(associative law)}
\]
\[
\iff A \cap B = A \cap B \quad \text{(by 1 and 2)}
\]

(+ Line: 17, 29)

(i) Validation for column 3 (on line 17 in the table, + corresponds to: Intersecting, Subset).

\[
A \subset C \land (B \cap C \neq \emptyset) \land B \not\subset C \land B \not\supset C \rightarrow A \not\supset B
\]
Assume \(A \supseteq B\);
\[
C \supset A \land A \supseteq B \rightarrow C \supset B \quad \text{(transitivity)}
\]
Contradiction with \(B \not\subset C\). Thus \(A \not\supset B\).
(ii) Validation for prediction, column 4.

\[ A \subset C \land (B \cap C \neq \emptyset \land B \notin C \land B \notin C) \quad \Rightarrow \quad A \cap B \subset C \]

Given \( A \subset C \rightarrow A \cap C = A \) (by B.1) (1)

\( (A \cap B) \subset C \rightarrow (A \cap B) \cap C = A \cap B \) (by B.1)

\[ A \cap B \cap C = A \cap B \iff A \cap C \cap B = A \cap B \] (associative law)

\[ \iff A \cap B = A \cap B \] (by 1)

(+) Line: 19

\[ A \supset C \land B \supset C \land (B \cap C \neq \emptyset \land B \notin C \land B \notin C) \quad \Rightarrow \quad (A \cap B) \supset C \]

\( A \supset C \rightarrow A \cap C = C \) (by B.1) (1)

\( B \supset C \rightarrow B \cap C = C \) (by B.1) (2)

\( (A \cap B) \supset C \rightarrow (A \cap B) \cap C = C \) (by B.1)

\( (A \cap B) \cap C = C \iff A \cap B \cap C \cap C = C \) (idempotent law)

\[ \iff (A \cap C) \cap (B \cap C) = C \] (associative law)

\[ \iff (C) \cap (C) = C \] (by 1 and 2)

\[ \iff C = C \] (idempotent law)

For validation of other possible prediction, superset, please refer to validation for Line: 20.

(+) Line: 20

\[ A \supset C \land B \supset C \quad \Rightarrow \quad (A \cap B) \supset C \]

\( A \supset C \rightarrow A \cap C = C \) (by B.1) (1)

\( B \supset C \rightarrow B \cap C = C \) (by B.1) (2)

\( (A \cap B) \supset C \rightarrow (A \cap B) \cap C = C \) (by B.1)

\( (A \cap B) \cap C = C \iff A \cap B \cap C \cap C = C \) (idempotent law)

\[ \iff (A \cap C) \cap (B \cap C) = C \] (associative law)

\[ \iff (C) \cap (C) = C \] (by 1 and 2)

\[ \iff C = C \] (idempotent law)
Appendix B. Prediction Table

(+ Line: 23, 26)

\[ A \supset C \land (B \cap C \neq \emptyset \land B \not\subseteq C \land B \not\supseteq C) \land
\]
\[ (A \cap B \neq \emptyset \land A \not\in B \land B \not\in A) \rightarrow (A \cap B) \subset C \]

\[ A \supset C \rightarrow (A \cap C) = C \quad \text{(by B.1)} \quad (1) \]
\[ (A \cap B) \subset C \rightarrow A \cap B \cap C = A \cap B \quad \text{(by B.1)} \]
\[ A \cap B \cap C = A \cap B \iff A \cap C \cap B = A \cap B \quad \text{(associative law)} \]
\[ \iff C \cap B = A \cap B \quad \text{(by 1)} \]

Assume \( A \cap B = \emptyset; \)
\[ \iff C \cap B = \emptyset \]
Contradiction with \( B \cap C \neq \emptyset. \) Thus proposition holds.

For validation of other possible prediction, intersecting, please refer to validation for Line: 24, 27-part 2.

(+ Line: 24, 27)

1. Validation for column 3 (on line 24 in the table, + corresponds to: Superset).

\[ A \supset C \land (B \cap C \neq \emptyset \land B \not\subseteq C \land B \not\supseteq C) \rightarrow A \not\subseteq B \land (A \cap B \neq \emptyset) \]

(i) Assume \( A \subseteq B; \)
\[ A \subseteq B \rightarrow (A \cap B) = A \quad \text{(by B.1)} \quad (1) \]
\[ A \supset C \rightarrow (A \cap C) = C \quad \text{(by B.1)} \quad (2) \]
\[ A \supset C \iff (A \cap B) \supset C \quad \text{(by 1)} \]
\[ (A \cap B) \supset C \rightarrow (A \cap B) \cap C = C \quad \text{(by B.1)} \]
\[ (A \cap B) \cap C = C \iff B \cap C = C \quad \text{(by 2)} \]
\[ \iff B \supset C \quad \text{(by B.1)} \]
Contradiction with \( B \not\subseteq C. \) Thus \( A \not\subseteq B \)
(ii) Assume \( A \cap B = \emptyset \);

\[
B \cap C \neq \emptyset \iff B \cap (A \cap C) \neq \emptyset \quad \text{(by 2)}
\]
\[
\iff \emptyset \cap C \neq \emptyset
\]
\[
\iff \emptyset \neq \emptyset \quad \text{(domination law)}
\]
Contradiction. Thus \( A \cap B \neq \emptyset \)

Therefore proposition is true.

2. Validation for prediction, column 4.

\[
A \supset C \land (B \cap C \neq \emptyset \land B \not\subseteq C \land B \not\supseteq C) \rightarrow
\]
\[
(A \cap B) \cap C \neq \emptyset \land (A \cap B) \not\subseteq C \land (A \cap B) \not\supseteq C
\]

Assume \( (A \cap B) \cap C = \emptyset \land (A \cap B) \not\subseteq C \land (A \cap B) \not\supseteq C \); Using the first term ;

\[
(A \cap B) \cap C = \emptyset \iff A \cap C \cap B = \emptyset \quad \text{(associative law)}
\]
\[
\iff C \cap B = \emptyset \quad \text{(by 2)}
\]
Contradiction with \( B \cap C \neq \emptyset \). Thus proposition holds

(+) Line:31

1. Prediction value: SUBSET.

\[
(A \cap C \neq \emptyset \land A \not\subseteq C \land A \not\supseteq C) \land (B \cap C \neq \emptyset \land B \not\subseteq C \land B \not\supseteq C) \land
\]
\[
(A \cap B \neq \emptyset \land A \not\subseteq B \land A \not\supseteq B) \rightarrow A \cap B \subset C
\]

(i) Assume \( A \cap C = \emptyset \);

\[
A \cap B \subset C \rightarrow A \cap B \cap C = A \cap B \quad \text{(by B.1)}
\]
\[
A \cap B \cap C = A \cap B \iff A \cap C \cap B = A \cap B \quad \text{(associative law)}
\]
\[
\iff \emptyset \cap B = A \cap B
\]
\[
\iff \emptyset = A \cap B \quad \text{(idempotent law)}
\]
Contradiction with \( A \cap B \neq \emptyset \). Thus \( A \cap C \neq \emptyset \)
(ii) Assume $B \cap C = \emptyset$;
\[ A \cap B \subset C \implies A \cap B \cap C = A \cap B \quad \text{(by B.1)} \]
\[ A \cap B \cap C = A \cap B \iff A \cap \emptyset = A \cap B \]
\[ \iff \emptyset = A \cap B \quad \text{(idempotent law)} \]
Contradiction with $A \cap B \neq \emptyset$. Thus $B \cap C \neq \emptyset$

(iii) Assume $A \supset B \land B \supset A$;
\[ A \supset B \land B \supset A \implies A = B \quad \text{(transitivity)} \quad (1) \]
\[ A \cap B \subset C \iff A \cap A \subset C \quad \text{(by 1)} \]
\[ \iff A \subset C \quad \text{(idempotent law)} \]
Contradiction with $A \not\subset C$. Thus $A \not\supset B \land B \not\supset A$

Therefore proposition is true.

2. Prediction value: INTERSECTING. Please refer to validation of Line:33

3. Prediction value: DISJOINT.
\[
(A \cap C \neq \emptyset \land A \not\subset C \land A \not\supset C) \land (B \cap C \neq \emptyset \land B \not\subset C \land B \not\supset C) \\
\land (A \cap B \neq \emptyset \land A \not\subset B \land A \not\supset B) \implies A \cap B \cap C = \emptyset
\]

(i) Assume $A \supset C \land C \supset A$;
\[ A \supset C \land C \supset A \implies A = C \quad \text{(transitivity)} \quad (1) \]
\[ A \cap B \cap C = \emptyset \implies A \cap B \cap A = \emptyset \quad \text{(by 1)} \]
\[ \implies A \cap B = \emptyset \quad \text{(idempotent law)} \]
Contradiction with $A \cap B \neq \emptyset$. Thus $A \not\supset C \land C \not\subset A$

(ii) Assume $B \supset C \land C \supset B$;
\[ B \supset C \land C \supset B \implies B = C \quad \text{(transitivity)} \quad (1) \]
\[ A \cap B \cap C = \emptyset \implies A \cap B \cap B = \emptyset \quad \text{(by 1)} \]
\[ \implies A \cap B = \emptyset \quad \text{(idempotent law)} \]
Contradiction with $A \cap B \neq \emptyset$. Thus $B \not\supset C \land C \not\supset B$
Appendix B. Prediction Table

(iii) Assume \( A \supset B \land B \supset A; \)

\[
\begin{align*}
A \supset B \land B \supset A & \rightarrow A = B \\
& \quad \text{(transitivity)} \\
A \cap B \cap C = \emptyset & \rightarrow A \cap A \cap C = \emptyset \\
& \rightarrow A \cap C = \emptyset \\
& \quad \text{(idempotent law)}
\end{align*}
\]

Contradiction with \( A \cap C \neq \emptyset. \) Thus \( A \not\supset B \land B \not\supset A \)

Therefore proposition is true.

(+) Line:33

\[
(A \cap C \neq \emptyset \land A \not\subset C \land A \not\supset C) \land (B \cap C \neq \emptyset \land B \not\subset C \land B \not\supset C)
\]

\[
\rightarrow (A \cap B \cap C \neq \emptyset \land A \cap B \not\subset C \land A \cap B \not\supset C)
\]

Assume \( A \supset C \land C \supset B; \)

\[
\begin{align*}
A \supset C \land C \supset B & \rightarrow A \supset B \\
& \quad \text{(transitivity)} \\
A \supset B & \rightarrow A \cap B = B \\
& \quad \text{(by B.1)} \\
A \cap B \not\subset C & \rightarrow B \not\subset C \\
& \quad \text{(by 1)}
\end{align*}
\]

Contradiction with \( C \supset B. \) Therefore assumption is wrong. Proposition holds.
Bibliography


