



RWTH Aachen University
Lehr- und Forschungsgebiet Informatik 2

# Developing a Library for Accelerating Octagonal Relations

**Bachelor Thesis**

presented by

**Torben Steegmann**

**1st Examiner: Prof. Dr. Jürgen Giesl**
**2nd Examiner: Prof. Dr. Erika Ábrahám**
**Advisor: Dr. Florian Frohn**

Aachen, April 6, 2023

Zentrales Prüfungsamt/Central Examination Office

RWTH AACHEN UNIVERSITY

# Eidesstattliche Versicherung
## Declaration of Academic Integrity

Name, Vorname/Last Name, First Name

Matrikelnummer (freiwillige Angabe)
Student ID Number (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel
I hereby declare under penalty of perjury that I have completed the present paper/bachelor's thesis/master's thesis* entitled

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
independently and without unauthorized assistance from third parties (in particular academic ghostwriting). I have not used any other sources or aids than those indicated. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. I have not previously submitted this work, either in the same or a similar form to an examination body.

Ort, Datum/City, Date

Unterschrift/Signature

*Nichtzutreffendes bitte streichen/Please delete as appropriate

**Belehrung:**
**Official Notification:**

**§ 156 StGB: Falsche Versicherung an Eides Statt**
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.
**§ 156 StGB (German Criminal Code): False Unsworn Declarations**
Whosoever before a public authority competent to administer unsworn declarations (including Declarations of Academic Integrity) falsely submits such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment for a term not exceeding three years or to a fine.
**§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt**
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.
**§ 161 StGB (German Criminal Code): False Unsworn Declarations Due to Negligence**
(1) If an individual commits one of the offenses listed in §§ 154 to 156 due to negligence, they are liable to imprisonment for a term not exceeding three years or to a fine.
(2) The offender shall be exempt from liability if they correct their false testimony in time. The provisions of § 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Ort, Datum/City, Date

Unterschrift/Signature

**Abstract**

Octagonal relations are popular abstractions of software systems used in program verification. While in theory they can be used to decide a range of problems in polynomial time, in practice their effectiveness is limited because they can get computationally expensive. This thesis aims to develope a new library called Scout that utilizes advanced algorithms and data structures introduced in [13] to accelerate the analysis of octagonal relations. The original software was written in java. Using C++ we will see that the implementation of Scout offers great performance increases and is easier to integrate into other tools that are not written in JVM-languages.

Scout is designed to be easy to use and flexible, allowing it to be integrated into various software verification tools. The library also includes certain new optimizations to speed up acceleration that were not present in earlier implementations of acceleration techniques for octagons. The library also includes several optimizations that exploit the properties of octagonal constraints to speed up their analysis.

To evaluate the effectiveness of the library, a series of experiments is conducted on a large dataset of octagons. The experiments compare the performance of the proposed library with existing tools for octagonal relation analysis. The results show that the proposed library is significantly faster than existing tools.

Overall, this thesis contributes to the development of more efficient and effective methods for program analysis and verification. Scout provides a powerful tool for analyzing programs that can be abstracted as octagonal constraints and has the potential to be integrated into a wide range of software verification tools.

# Acknowledgement

I would like to thank Professor Dr. Jürgen Giesl and Professor Dr. Erika Ábrahám for taking their time to review this thesis. I would also like to express my sincere gratitude to Dr. Florian Frohn. His guidance, support, and mentorship throughout this project were instrumental in its success. I am deeply grateful for his patience, encouragement, and willingness to share his expertise and insights with me. His feedback and encouragement were key factors in my progress and achievement. Lastly, I want to thank my parents and everyone who has supported me throughout my studies.

# Contents

# 1 Introduction

## 1.1 Motivation

Software verification has been a fundamental part of the software development life cycle for decades. As today's software becomes increasingly complex, the need for more advanced and efficient verification methods has never been greater. While simple programs can rely on testing, advanced programs require a more elaborate, formal approach, especially in safety-critical environments such as medical devices or cryptography.

One such approach is relational verification. It is important to note that there is no formal definition of relational verification and there exist other methods than the one discussed in thesis. For example, relational verification can refer to comparing two different programs, trying to prove their equivalence. When this thesis refers to relational verification however, we mean that we want to verify certain relationships or properties of sets of constraints. While this method has been successful in some cases, it can be challenging to automate because of its complexity and the need for significant computational resources.

One motivation behind developing new techniques for relational verification is to make the approach more accessible and practical. With the increasing complexity of software systems, there is a growing need for formal verification methods that can analyze and ensure the correctness of such systems. However, current approaches to relational verification can be difficult to use, require significant expertise, and are often computationally expensive. Therefore, new techniques are needed that can make relational verification more accessible and efficient.

Another motivation for developing new techniques is to improve the precision and scalability of relational verification. Current techniques can be imprecise or limited in the types of programs they can analyze. This limits their usefulness in real-world scenarios, where software systems can be highly complex and interconnected. The goal behind developing new techniques for relational verification is to improve the precision and scalability of the approach, allowing it to be used in a wider range of scenarios.

In his dissertation "Relational Verification of Programs with Integer Data" [13], Konečný proposes a novel and automated approach to relational verification. This approach is able to calculate precise solutions by calculating the transitive closure of the transition relation of a given program. By using an acceleration technique called "loop acceleration", the algorithm runs comparatively faster than other techniques. This makes relational verification more accessible, efficient, and precise. It enables the analysis of complex software systems and helps ensure their correctness, making them more reliable and secure.

## 1.2 Goal of this Thesis

Konečný implemented his algorithm in a free software called Flata which is available online[1]. However, as the software is written entirely in java it is not as flexible

[1] `http://nts.imag.fr/index.php/Flata`

as a program written in c++ because almost all programming languages allow interaction with c++ libraries. While Java does have its own upsides, like having a large developer community and garbage collection, c++ sacrifices ease of use for control and speed, thus using this programming language should yield further performance increases. Also, to be able to use Flata, one has to install two pieces of third party software. One of these is an SMT-solver called "Yices", which is, while still available, no longer maintained as it has been replaced by its newer version "Yices 2".

The goal of this thesis is to develop a library for accelerating octagon relations, for which we will implement Konečný's proposed algorithm in C++. The library should be designed to improve the efficiency and scalability of verification techniques for octagon relations, allowing for faster and more precise analysis of software systems. The library should be modular and easy to use, allowing other researchers and practitioners in the field of formal verification to integrate it into their own tools and frameworks.

To achieve this goal, a thorough literature review should be conducted on octagon relations and existing methods for accelerating them. Based on this literature review, the presented algorithms should be implemented in C++.

The performance of the library should be evaluated by calculating the transitive closure of a predetermined dataset of octagonal relations. This performance will be measured in speed and correctness and then compared to Flata's performance. The evaluation will demonstrate the effectiveness of the library in accelerating octagon relations.

Overall, this thesis should contribute to the field of formal verification by improving the efficiency and scalability of verification techniques of octagon relations, making it easier and faster to analyze complex software systems using formal verification methods. The library developed as part of this thesis should be a valuable tool for researchers and practitioners in the field of formal verification.

# 2 State of the Art

This section provides an introduction to general software and relational verification.

## 2.1 Software Verification and Validation

"Software quality matters-more than ever." [21] As requirements for modern software rise [12], so does the need for modern quality insurance. Software verification and validation are essential parts of software engineering that involve a range of activities to ensure the quality and reliability of software systems and components. They are processes performed during the software development life cycle, that may take place during every phase [23] but are typically performed after the software design phase and before the software testing phase. While validation focuses on ensuring that the product meets previously specified requirements,

hence tries to answer the question "Am I building the right product?" [5], the primary goal of verification is to detect and eliminate defects and errors early in the life cycle, which can help reduce the cost and time associated with fixing them later. It tries to answer the question "Am i building the product right?" [5].

Software verification is a broad term that covers a range of activities, including testing, analysis, and formal methods. The activities involved in software verification and validation may include [23]:

1. Requirements verification: This involves ensuring that the software system or component meets the specified requirements and that the requirements are complete, correct, and consistent.

2. Design verification: This involves verifying that the software system or component design meets the specified requirements and that it is implementable and testable.

3. Code verification: This involves verifying that the source code for the software system or component is correct, complete, and consistent with the design and requirements.

4. Testing: This involves verifying the functionality, performance, and reliability of the software system or component using predefined test cases and scenarios.

5. Static code analysis: This involves analyzing the source code for potential issues and vulnerabilities, such as coding errors, security vulnerabilities, and performance bottlenecks.

6. Formal verification: This is a form of static analysis which involves using mathematical techniques to prove that the software system or component meets its specification and that it is free from defects and errors.

Effective software verification can help improve the quality and reliability of software systems and components, reduce the risk of failures and security vulnerabilities and increase user satisfaction. However, software verification can be a complex and time-consuming process and it requires a range of specialized skills and tools [21].

Therefore the specific activities involved in software verification may vary depending on the type and complexity of the software system or component. While testing may be sufficient for simple programs, software systems that are safety-critical or security-critical may require a more rigorous and formal verification process, one of which is the focus of this thesis.

## 2.2 Relational Verification

Relational verification is a formal verification technique used to verify the correctness of software programs. While it does not have a formal definition and there are multiple different forms of it, we will refer to relational verification as analysis

of different types of relations, namely Difference Bounds Relation (see sec. 4.2) and Octagonal Relations (see sec. 4.3). These Relations are abstractions of software systems. Relational Verification aims to mathematically analyze a program's behavior and check whether or not it conforms to its specification or requirements.

In order to be able to perform relational verification, a program is typically first translated into a formal representation. Such representations include but are not limited to:

1. Automata
2. Transition Systems
3. Logical Formulas

As we will see these representations are not necessarily exclusive from one another, meaning that, for example, a program that is represented as a logical formula can sometimes also be represented as an Automaton [13]. These representations aim to capture the program's behavior and are used to verify the program's correctness.

Once the program has been translated into a formal specification, formal verification techniques can be used to compare the program's behavior with its specification. Two commonly used techniques are theorem proving [4] and model checking [13].

Theorem proving uses logical deduction rules to derive the correctness of a program from its formal specification. Theorem provers use various automated reasoning techniques, such as resolution [20], to derive a proof of correctness for a program. One downside of this technique can be false positives leading to inaccurate results [11].

Model checking is a technique used to verify finite-state systems, previously done by algorithms to exhaustively explore all possible states of the system and checking whether or not they satisfy a given specification [8]. This approach, however, can notoriously suffer from the "state explosion" problem, where the number of states that need to be checked grows exponentially with the size of the system [9].

While these are far from the only relational verification techniques that exists they are depicted here to show a common problem that a lot of these methods share. Relational verification does provide a mathematically rigorous method for verifying the correctness of software programs. It does make it possible to detect subtle errors or corner cases that might be missed by other verification techniques and hence it is particularly well-suited for safety-critical systems, such as medical devices [16], cryptography [3], or nuclear power plants [19].

However, the previously shown example of "false positives" and "state explosion" show that major limitations of relational verification include that they can yield inaccurate results and can be computationally expensive, particularly for complex programs. To address these issues, various techniques to improve its efficiency have been developed. One type of these are called acceleration techniques [13].

## 2.3 Acceleration in Relational Verification

As mentioned in the previous chapter, modern software systems are becoming increasingly more complex. Traditional methods of relational verification are thus becoming computationally more expensive and time-consuming. This can lead to scalability and efficiency issues in the verification process, making it difficult for developers to effectively identify and eliminate errors and defects in the software system.

To address these challenges, different techniques and tools to increase the speed of the verification process are being explored. One of the popular used techniques is abstraction [22].

Abstraction is a powerful technique for reducing the complexity of software models used in relational verification. Abstraction techniques can involve simplifying or abstracting the model by removing irrelevant or unnecessary details, or by approximating complex behaviors with simpler models. Developers can then use these abstract interpretations to analyze a program's behavior without considering every possible input or execution path explicitly. By removing details that are not critical to the verification process, abstraction can help reduce the complexity and size of the software model, making it more computationally tractable. One form of such abstractions are octagons (see sec. 4.3).

In addition to this technique, other approaches are also being developed, one of which is loop acceleration that is used in this thesis. While traditional relational verification methods often rely on over-approximation [24] and fixpoint computations [7] which lead to imprecise and slow results, this technique calculates the precise transitive closure of a program [13]. While there is no formal definition of loop acceleration, this particular concept relies on an attribute all relations, to which this algorithm is applicable to, must have. It is called "periodicity" (4.1.4). Loop acceleration does not rely on any kind of searching and will instead calculate an inconsistency after a certain amount of steps in the relation directly. If the relation is consistent for all steps it will instead calculate the precise transitive closure of the relation through its periodicity. Because our goal is exactly to calculate the transitive closure of a relation, we will see that this technique offers promising results.

# 3 Related Work

Before we continue explaining and demonstrating the methods implemented in this thesis, let us first take a look at other papers presenting similar techniques.

**PTIME Computation of Transitive Closures of Octagonal Relations:** In this paper Konečný describes an alternative approach to accelerating octagons which he shows to be computable in polynomial time rather than the exponential time that the algorithm presented as part of this thesis runs in [14]. This polynomial time approach has not yet been implemented though, and because it results in excessively large formulas and constraints the method described in this thesis is expected to be faster in practice, even if it has a worse time complexity.

**Iterating Octagons:** This paper describes a similar approach to the one used in this thesis [6]. Because it does not use loop acceleration, the calculation of the transitive can take a long time even for small relations. The paper does however provide the foundation some of the methods discussed here were build upon.

# 4 Preliminaries

This chapter provides the necessary background information needed to understand the following sections of the thesis.

## 4.1 Basic Notations

This section focuses on introducing basic concepts and notations to the reader.

### 4.1.1 Integer Relations

In this paper we will focus on Integer Relations and will thus predominantly look at the number spaces $\mathbb{Z}$, meaning the set of all integers, $\mathbb{N}$, the set of positive Integers including $0$, and $\mathbb{N}_+$, the set of strictly positive Integers excluding $0$. We denote the set of variables ranging over $\mathbb{Z}$ with $\mathbf{x} = \{x_0, ..., x_N\}, N > 0$, each variable having a primed version $\mathbf{x}' = \{x_0, ...x_N\}$. Let $R_1, R_2 \subseteq \mathbb{Z}^N \times \mathbb{Z}^N$. Relational composition $R_1 \circ R_2$ is said to have an element $s_1 \in R_1$ in relation to and element $s_2 \in R_2$ exactly when there exists a third element $s_3 \in R_1, R_2$ so that $s_1$ in relation with $s_3$ and $s_3$ is in relation with $s_2$. This is formally defined as:

**Definition 4.1.** $R_1 \circ R_2 = \{(s, s') \in \mathbb{Z}^N \times \mathbb{Z}^N \mid \exists s'' \in \mathbb{Z}^N \,.\, (s, s'') \in R_1 \wedge (s'', s') \in R_2\}$.

We say $R^m$ is the m-th power of $R$, meaning that $R$ is composited with itself m times, so that $R^{m+1} = R^m \circ R$, while $I = R^0$ is the identity Relation, where every element is only in relation to itself, or formally $I = \{(s, s) \mid \in \mathbb{Z}^N\}$ and hence $R^m \circ R^0 = R^m$. Lastly, $R^+ = \bigcup_{i=1}^{\infty} R^i$ denotes the transitive closure of $R$.

### 4.1.2 Presburger Arithmetic

In the following chapters we will discuss Difference Bounds- and Octagonal Relations which will be expressed in Presburger Arithmetic. It is thus important to first discuss what that is. Presburger Arithmetic is the first-order theory over the signature $\Sigma = \{\mathbb{Z}, +, -, <\}$ meaning it contains only integers, addition, subtraction and natural ordering. Most notably it does not contain multiplication and is thus decidable.

Other important attributes include that it has quantifier elimination and that the validity of linear integer arithmetic formulas can be reduced to the validity of Presburger Arithmetic formulas. For this reason we can use both notions of Presburger Arithmetic and linear integer interchangeably.

The following is a list of examples of Presburger Arithmetic formulas:

1. $\varphi_1 = x - y \leq c_1$
2. $\varphi_2 = -x + y < z$
3. $\varphi_3 = x - z - y + u = c_2$

Notice that each of these formulas consist of a single atomic proposition (AP), meaning each proposition contains no smaller part that is also a proposition. We denote AP($\phi$) as the set of all atomic proposition in $\varphi$, thus assuming a formula $\varphi = x - y \leq 3 \wedge x' - x \leq 5$, we have AP($\varphi$) $= \{(x - y \leq 3), (x' - x \leq 5)\}$.

### 4.1.3 Consistency

The algorithm presented in this thesis uses consistency (satisfiability) to calculate the transitive closure.

**Definition 4.2.** *A formula $\varphi(\mathbf{x})$ is said to be consistent if there exists a valuation $v : \mathbf{x} \to \mathbb{Z}$ such that $v \models \varphi(\mathbf{x})$. By $[\![\varphi]\!]$, we denote the models of $\varphi(\mathbf{x})$, meaning the set of all valuations satisfying $\varphi(\mathbf{x})$. These are also called models of $\varphi(\mathbf{x})$. Formally $[\![\varphi]\!] = \{v : \mathbf{x} \to \mathbb{Z} \mid v \models \varphi(\mathbf{x})\}$.*

Let us assume the formula $\varphi = x + y \leq 1, x, y \in \mathbb{N}$. This is consistent as you can satisfy the constraint with the valuation $v : x \to 1$, $v : y \to 0$, with the set of all models being $[\![\varphi]\!] = \{(0, 0), (0, 1), (1, 0)\}$. We call a formula $\varphi$ valid when every valuation satisfies $\varphi$, and it is thus logically equivalent to true ($\top$), denoted by $\models \varphi$. An example formula for which this holds is $\varphi = x < 2 \vee x > 1$. Similarly an invalid, or inconsistent, formula is logically equivalent to false ($\bot$), denoted by $\models \neg\varphi$. An example of this is the formula $\varphi = x < 2 \wedge x > 3$. We notate logical implication and equivalence by $\Rightarrow$ and $\Leftrightarrow$. A relation $R$ is *-consistent if $R^m$ is consistent for all $m \in \mathbb{N}$.

### 4.1.4 Periodicity

Another key part of the algorithm is calculating what is called the periodicity of a relation. To understand it we will first discuss the periodicity of integer sequences which will naturally extend into the periodicity of matrices and thus, as we will later see, relations. Let us begin, again, by looking at the formal definition of periodicity of integer sequences.

**Definition 4.3.** *Given an infinite sequence $\{s_k\}_{k=0}^{\infty} \in \mathbb{Z}$, we say that it is periodic if and only if there exist integers $b \geq 0$, $c > 0$ and $\lambda_0, ..., \lambda_{c-1} \in \mathbb{Z}_\infty$ such that $s_{b+(k+1)*c+i} = \lambda_i + s_{b+k*c+i}$ , for all $k \geq 0$ and $i = 0, 1, ..., c - 1$. The smallest values $b \in \mathbb{N}$, $c \in \mathbb{N}_+$ for which the above holds are called the prefix and the period of $\{s_k\}_{k=0}^{\infty}$ [13].*

What this means is that an integer sequence is periodic exactly when there is a value $\lambda$ that separates two values in a sequence in a regular interval or period ($c$) after a prefix ($b$). The easiest example for such a sequence is that of the natural

numbers: $\{1, 2, 3, 4, 5, 6, 7, 8, ...\}$. Here the prefix $b$ is $0$ as the periodicity already starts in the first element $s_0 = 1$. The period $c$ is 1 as every element has the same distance $\lambda_0 = 1$ to the next. We can use this to calculate any element of the sequence recursively, for example for $s_2$:

$$s_{b+(k+1)*c+i} = \lambda_i + s_{b+k*c+i} \Leftrightarrow s_{0+(1+1)*1+0} = 1 + s_{0+1*1+0}$$
$$\Leftrightarrow s_2 = 1 + s_1 = 1 + (1 + s_{0+0*1+0}) \Leftrightarrow s_2 = 1 + s_1 = 1 + (1 + s_0)$$
$$\Leftrightarrow s_2 = 1 + s_1 = 1 + (1 + 1) = 3, \text{ or } s_k = l + 1 \text{ for each } k = l, l \geq 0.$$

Another example of a periodic sequence would be $\{s_k\}_{k=0}^{\infty}$ with $s_0 = 3$ $s_1 = 2$ $s_2 = 2$ $s_3 = 1$ $s_k = 2l$ for each $k = 3l, l >= 1$ were $2$ gets added to every element that has a sequence number that is divisible by three $(s_3, s_6, s_9...)$. Here the prefix is $b = 4$, which means that the prefix consist of $s_0$, $s_1$, $s_2$, $s_3$ and thus the periodicity begins in $s_4$ with period being $c = 3$ and the distance $\lambda$ being $\lambda = 2$.

An example of a non-periodic sequence is that of the Fibonacci-Sequence.

This definition of sequences can naturally be extended to that of the sequences of matrices. Here a matrix is said to be periodic if and only if each element in the matrix is periodic. Let us look at an example of this:

$$\{A_1 : \begin{pmatrix} 3 & 2 & 1 & 1 \\ 2 & 1 & 3 & 1 \\ 1 & 2 & 4 & 3 \\ 3 & 2 & 6 & 4 \end{pmatrix} A_2 : \begin{pmatrix} 5 & 4 & 2 & 1 \\ 2 & 6 & 5 & 1 \\ 1 & 2 & 6 & 3 \\ 4 & 3 & 6 & 5 \end{pmatrix} A_3 : \begin{pmatrix} 5 & 3 & 2 & 5 \\ 2 & 7 & 5 & 1 \\ 1 & 2 & 9 & 3 \\ 6 & 3 & 6 & 5 \end{pmatrix} A_4 : \begin{pmatrix} 5 & 6 & 2 & 1 \\ 3 & 6 & 11 & 1 \\ 1 & 2 & 6 & 5 \\ 4 & 6 & 6 & 6 \end{pmatrix} ...\}$$

Let us assume that $A_1$ is the prefix. Then $b = 1$ and $A_2$ is the first element of the periodic subset of the sequence. Every other element can be defined by:

$$A_{n+2} : \begin{pmatrix} 0*n+5 & 0*n+4 & 0*n+2 & 4*n+1 \\ 0*n+2 & 1*n+6 & 0*n+5 & 0*n+1 \\ 0*n+1 & 0*n+2 & 3*n+6 & 0*n+3 \\ 2*n+4 & 0*n+3 & 0*n+6 & 0*n+5 \end{pmatrix} \forall n, n \text{ is uneven.}$$

$$A_{n+2} : \begin{pmatrix} 0*n+5 & 2*n+4 & 0*n+2 & 0*n+1 \\ 1*n+2 & 0*n+6 & 6*n+5 & 0*n+1 \\ 0*n+1 & 0*n+2 & 0*n+6 & 2*n+3 \\ 0*n+4 & 3*n+3 & 0*n+6 & 1*n+5 \end{pmatrix} \forall n, n \text{ is even.}$$

In other words, the distance between two element of the sequence are either $\Lambda_0$ or $\Lambda_1$:

$$\Lambda_0 = \begin{pmatrix} 0 & 0 & 0 & 4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 2 & 0 & 0 & 0 \end{pmatrix} \Lambda_1 = \begin{pmatrix} 0 & 2 & 0 & 0 \\ 1 & 0 & 6 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 3 & 0 & 1 \end{pmatrix}$$

With this, another way of looking at the above formula is hence:

$$A_{n+2} = A_2 + n * \Lambda_0, \forall n, n \text{ is uneven.}$$
$$A_{n+2} = A_2 + n * \Lambda_1, \forall n, n \text{ is even.}$$

This is formally defined as:

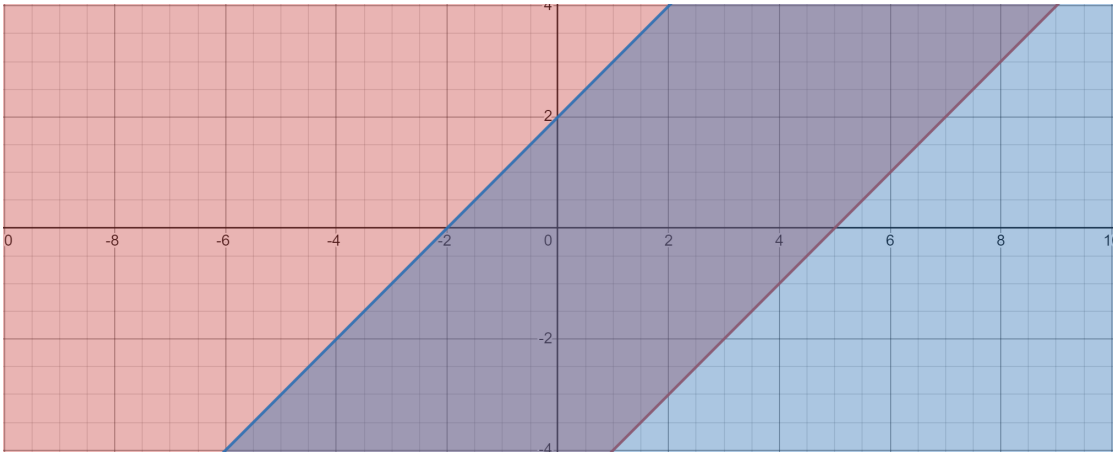


Figure 1: A graphic representation of the DBR: $x_1 - x_2 \leq 5 \wedge x_2 - x_1 \leq 2$. The red zone shows all the possible values of $x_2$ from the first constraint ($x_1 - x_2 \leq 5$) while the blue zone shows all the possible values of $x_2$ from the second constraint ($x_2 - x_1 \leq 2$). In combination, the overlapping zones represent all the remaining possible values for $x_2$ and $x_1$. This is why DBRs are also called 'zones'.

**Definition 4.4.** *An infinite sequence of matrices $\{A_k\}_{k=1}^{\infty} \in \mathbb{Z}_{\infty}^{m \times m}$ is periodic if and only if there exist integers $b \geq 0$, $c > 0$, and $\Lambda_0, \ldots, \Lambda_{c-1} \in \mathbb{Z}_{\infty}^{m \times m}$ such that $A_{b+(k+1)c+i} = \Lambda_i + A_{b+kc+i}$, for all $k \geq 0$ and $i = 0, 1, \ldots, c-1$ [13].*

## 4.2 Difference Bounds Relations

Difference Bounds Relations (DBRs) are types of conjunctions of atomic propositions that capture the relative order of two variables with respect to each other over, for example, different iterations of a loop. They can be useful for reasoning about timing constraints in real-time systems, such as bounds on the time between events, the ordering of events or efficient inclusion checks. They can also be used to check program safety by verifying that certain properties hold for all possible input values. They can also be used to identify all possible program paths, which is what we are gonna use them for.

DBRs are a a special case of linear inequalities. A DBR is typically defined as a set of constraints of the form $x_i - x_j \leq c$, where c is a constant while x and y are variables of either of the sets of variables $\mathbf{x}$ or $\mathbf{x}'$ where any variable $x_i$ of the set $\mathbf{x}$ stands for the value of $x_i$ before- and any variable $x_i'$ of the set $\mathbf{x}'$ stand for the value of $x_i$ after an operation. This constraint expresses that the difference between the values of $x_i$ and $x_j$ is bounded from above by c. For this reason, DBRs are also called zones. A visual representation of this can be found in Figure 1.

One of the main advantages of DBRs is that they can be used to reason about the behavior of a system over an infinite horizon. This is because DBRs define constraints on the relationship between variables over time, rather than just at a single point in time. The reason they are important to us is that the presented algorithm works with relations that are periodic which is a characteristic of DBRs proven by Konečný [13].

### 4.2.1 Difference Bounds Constraints

As we saw earlier, a Difference Bounds Constraint (DBC) is a type of constraint that defines DBRs and is used in program analysis and verification. They are often used to represent the range of values that a variable can take during program execution, and can be used to, for example, reason about the behavior of a program. While a DBR can be defined by a single DBC, they are often a conjunction of multiple DBCs.

DBCs are expressed as linear inequalities over integer or real variables, where each inequality represents a bound on the difference between two variables. While they are inequalities of the aforementioned form ($x_1 - x_2 \leq c$), they can be represented by any formula that is equivalent to it. For example, consider the constraint $x_1 - x_2 < 3$. As we work only with integers, this constraint can be represented as $x_1 - x_2 \leq 2$. Similarly, the constraint $x_1 - x_2 \geq c$ is equivalent to the DBC $x_2 - x_1 \leq -c$. Finally, the constraint $x_1 - x_2 = c$ is equivalent to the DBC $x_1 - x_2 \leq c \wedge x_2 - x_1 \leq -c$.

### 4.2.2 Matrix and Graph Representations

We can represent Difference Bounds Relations using different data structures, such as matrices or graphs, to facilitate their manipulation and analysis. These representations can provide a compact and efficient way to store and manipulate a large number of constraints, making it easier to reason about the program behavior.

One form of representation are graphs. Here, the variables are represented by nodes and the edges represent the constraints, with the weight of the edge being equal to the constant of the constraint. This representation is especially useful when trying to visualize the relationship between variables and constraints, as they portray the original states of the program that got translated into the DBR visually. The graph is formally defined as:

**Definition 4.5** (Constraint Graph [13]). *Let $\mathbf{x} = \{x_1, x_2, ..., x_N\}$ be a set of variables ranging over $\mathbb{Z}$ and $\varphi(\mathbf{x})$ be a difference bounds constraint. Then $\varphi$ can be represented as a weighted graph $G_\varphi = (\mathbf{x}, \rightarrow)$, where each vertex corresponds to a variable, and there is an edge $x_i \xrightarrow{a_{ij}} x_j$ in $G_\varphi$ if and only if there exists a constraint $x_i - x_j \leq a_{ij}$ in $\varphi$. This graph is also called a constraint graph.*

**Example 4.1** (DBR Leading Example). $\varphi = x_1 - x_2' \leq -2 \wedge x_2 - x_1' \leq 1 \wedge x_1' - x_1 \leq -1$

As an example, the constraint graph of the example above ("DBR Leading Example" 4.1) can be found in Figure 2.

Matrices are another way to represent DBCs as a matrix $M_\varphi$ can be defined as the incidence matrix of a given constraint graph $G_\varphi$. Naturally, each row and column represent a variable while the corresponding cell represents the constant of the constraint. What this means is the for any given atomic proposition $x_i - x_j \leq \alpha$ the cell in the $i$-th row and $j$-th column has the entry $\alpha$. This representation allows for efficient matrix operations, such as matrix multiplication and addition. It is formally defined by:

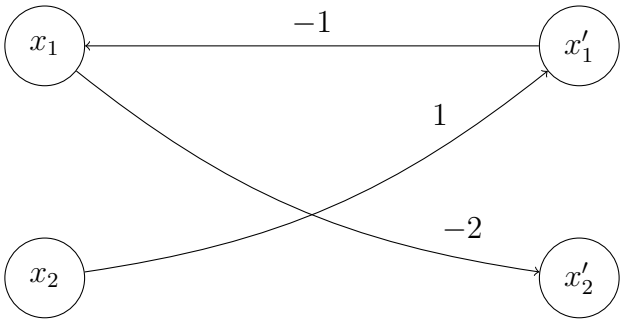


Figure 2: The Graph $G_\varphi$ that is corresponding to the Difference Bounds Relation $\varphi = x_1 - x_2' \leq -2 \wedge x_2 - x_1' \leq 1 \wedge x_1' - x_1 \leq -1$ (example 4.1). It is satisfiable as there are no negatively weighted cycles.

**Definition 4.6** (Matrix Representation [13]). *Let $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$ be a set of variables ranging over $\mathbb{Z}$ and $\varphi(\mathbf{x})$ be a difference bounds constraint. Then a difference bounds matrix (DBM) representing $\varphi$ is an $N \times N$ matrix $M_\varphi$ such that:*

$$(M_\varphi)_{i,j} = \begin{cases} \alpha_{i,j} & \text{if } (x_i - x_j \leq \alpha_{i,j}) \in AP(\varphi) \\ \infty & \text{otherwise} \end{cases}$$

The DBM of example 4.1 is hence:

**Example 4.2** (DBM $M_\varphi$ of DBR 4.1).

$$\begin{array}{c} \\ x_1 \\ x_2 \\ x_1' \\ x_2' \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_1' & x_2' \end{array} \\ \begin{pmatrix} 0 & \infty & \infty & -2 \\ \infty & 0 & 1 & \infty \\ -1 & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix} \end{array}$$

It is important to note here, that the ordering of the variables will, unless otherwise annotated, not change throughout this thesis.

The following is a definition relating the consistency of DBCs to the existence of negative weighted cycles[2].

**Definition 4.7** (Defintion of Consistency). *Let $\varphi$ be a difference bounds constraint and $G_\varphi$ be the constraint graph of $\varphi$. Then, the following statements are equivalent:*

- *$\varphi$ is consistent*
- *$G_\varphi$ contains no elementary negative weight cycles. [10]*

Next, we give a formal definition for $M_\varphi^*$ which is called the closed form of the DBM:

**Definition 4.8** (Closure of M). *A consistent DBM $M \in \mathbb{Z}_\infty^{N \times N}$ is said to be closed if and only if $M_{ii} = 0$ and $M_{ij} \leq M_{ik} + M_{kj}$, for all $1 \leq i, j, k \leq N$.*

[2]Written as "Proposition 2.6" in [13] but with a typo that changes its meaning.

This means that a DBM is closed, whenever the value of each of its cells corresponds to the shortest possible path in the corresponding constraint graph. This can be computed by the Floyd-Warshall algorithm with a time complexity of $\mathcal{O}(n^3)$. We say that two DBC are equivalent, if and only if the closed forms of their corresponding DBMs are equivalent.

Let us look at an example. Looking back at the example 4.1 with its corresponding DBM in example 4.2 we can compute the closed form $M^*_\varphi$ of $M_\varphi$ by using the Floyd-Warshall algorithm:

$$M^*_\varphi : \begin{pmatrix} 0 & \infty & \infty & -2 \\ 0 & 0 & 1 & -2 \\ -1 & \infty & 0 & -3 \\ \infty & \infty & \infty & 0 \end{pmatrix}$$

Because there are no negative values on the diagonal of $M^*_\varphi$ we can tell that there is no cycle with negative weight in the constraint graph $G_\varphi$ which is depicted in figure 2. This means that $M^*_\varphi$ is unique and the DBR is consistent as a consequence.

Both matrix and graph representations can be used to perform operations on the constraints, such as composition and intersection. Additionally, these representations can be translated to other formalisms, such as automata, however the ones we defined here will be the only ones used in this thesis.

## 4.3 Octagons

Octagons, or octagonal relations, are a generalization of Difference Bounds Relations and are thus also used as abstractions, for example, in software systems. The following sections will discuss how the methods used for DBRs extend to octagonal relations.

### 4.3.1 Octagonal Constraints

Octagons are defined by octagonal constraints of the form $\pm x_i \pm x_j \leq c_k$. Similarly to difference bounds constraints, $x_i$ and $x_j$ are variables while $c_k$ is a constant. Octagons get their name from the fact that the resulting shape when drawing out the function corresponding to these constraints can maximally be octagonal. This is illustrated in Figure 3.

Variables are either contained in a set $\mathbf{x} = \{x_1, x_2, x_3, ..., x_n\}$, which represents the values of variables before an operation, or in a set $\mathbf{x}' = \{x'_1, x'_2, x'_3, ..., x'_n\}$ which represents the values of variables after a step in the relation. We say that a formula $\varphi(\mathbf{x})$ is an octagon constraint when it is equivalent to a finite conjunction of terms of the form $\pm x_i \pm x_j \leq c_k$. Because $x_i$ may equal $x_j$ this includes constraints of the form $\pm x_i \leq c_k$ as it is equivalent to $\pm 2 \cdot x_i \leq 2 \cdot c_k$.

As example, let us look at the following formula.

$$\varphi(\mathbf{x}) = x_1 < 10 \wedge x'_1 = x_1 - 1.$$

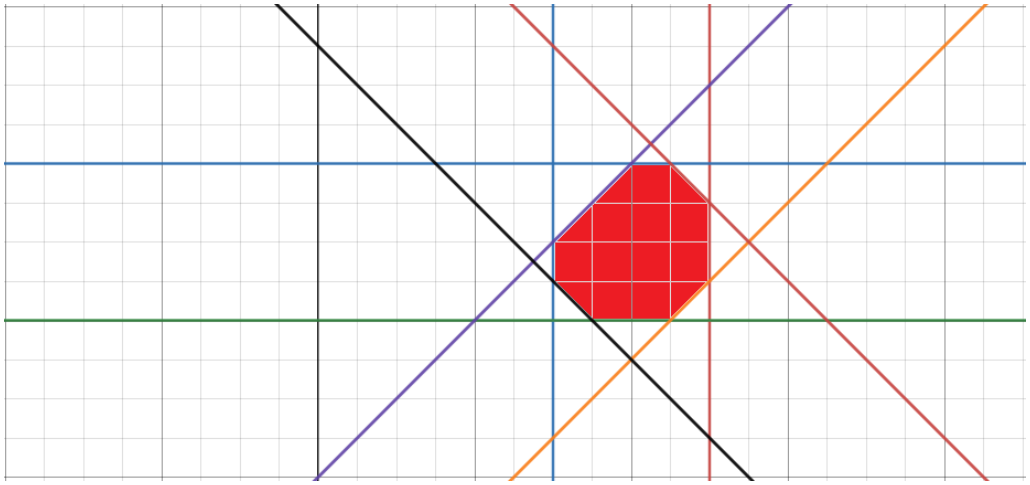

Figure 3: Visual Representation of an octagonal relation. Octagons get their name from the fact that the two dimensional space they encapsulate can be a polyhedra with at most eight edges.

This is an octagonal relation as it is equivalent to the following octagonal constraint.

**Example 4.3** (Octagon Leading Example)**.**

$$\varphi(\mathbf{x}) = 2x_1 \leq 18 \wedge x_1' - x_1 \leq -1 \wedge x_1 - x_1' \leq 1$$

To be able to use graph and matrix notations on octagons, we represent them as DBCs with the notation that a formula $\varphi(x_1, ..., x_n)$ gets transformed into a formula $\bar{\varphi}(y_1, ...y_{2n})$ where $y_{2i-1}$ stands for $+x_i$ and $y_{2i}$ stands for $-x_i$. For this we define the following equivalence relating DBCs to octagonal constraints:

**Definition 4.9** ([13])**.** *Given an octagonal constraint* $\varphi(\mathbf{x}), \mathbf{x} = \{x_1, \ldots, x_N\}$*, its difference bounds representation* $\bar{\varphi}(\mathbf{y}), \mathbf{y} = \{y_1, \ldots, y_{2N}\}$ *is a conjunction of the following difference bounds constraints where* $1 \leq i \neq j \leq N, c \in \mathbb{Z}$*.*

$$\begin{array}{ll}
1 : (x_i - x_j \leq c) \in AP(\varphi) & \Leftrightarrow (y_{2i-1} - y_{2j-1} \leq c), (y_{2j} - y_{2i} \leq c) \in AP(\bar{\varphi}) \\
2 : (-x_i + x_j \leq c) \in AP(\varphi) & \Leftrightarrow (y_{2j-1} - y_{2i-1} \leq c), (y_{2i} - y_{2j} \leq c) \in AP(\bar{\varphi}) \\
3 : (-x_i - x_j \leq c) \in AP(\varphi) & \Leftrightarrow (y_{2i} - y_{2j-1} \leq c), (y_{2j} - y_{2i-1} \leq c) \in AP(\bar{\varphi}) \\
4 : (x_i + x_j \leq c) \in AP(\varphi) & \Leftrightarrow (y_{2i-1} - y_{2j} \leq c), (y_{2j-1} - y_{2i} \leq c) \in AP(\bar{\varphi}) \\
5 : (2x_i \leq c) \in AP(\varphi) & \Leftrightarrow (y_{2i-1} - y_{2i} \leq c) \in AP(\bar{\varphi}) \\
6 : (-2x_i \leq c) \in AP(\varphi) & \Leftrightarrow (y_{2i} - y_{2i-1} \leq c) \in AP(\bar{\varphi})
\end{array}$$

Note that, when talking about exclusively octagons, $\varphi(\mathbf{x})$ and $\bar{\varphi}(\mathbf{y})$ will be used synonymously. Also note that, in the definition above, the condition $1 \leq i \neq j \leq N$ is not necessary but avoids redundant constraints. This is because for $i = j$, equations $3$ ($4$) and $6$ ($5$) are equivalent. Applying Definition 4.9 to our leading example 4.3 results in the following DB representation.

**Example 4.4** (DB Representation of the Leading Example)**.**

$$\bar{\varphi}(\mathbf{y}) = y_1 - y_2 \leq 18 \wedge y_1' - y_1 \leq -1 \wedge y_2 - y_2' \leq -1 \wedge y_1 - y_1' \leq 1 \wedge y_2' - y_2 \leq 1$$

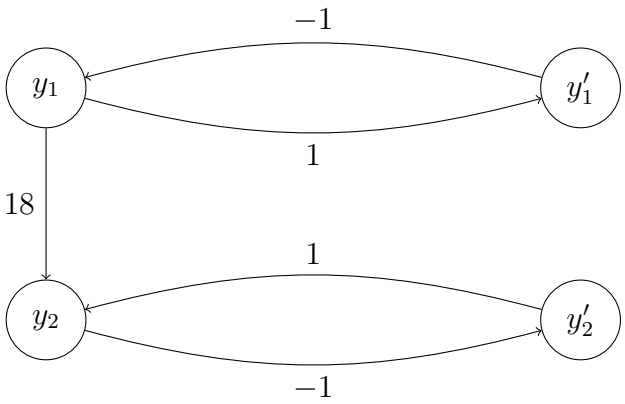


Figure 4: The Graph $G_\varphi$ that is corresponding to the DB representation $\bar{\varphi}(\mathbf{y}) = y_1 - y_2 \leq 18 \wedge y_1' - y_1 \leq -1 \wedge y_2 - y_2' \leq -1 \wedge y_1 - y_1' \leq 1 \wedge y_2' - y_2 \leq 1$ of the octagon relation $\varphi(\mathbf{x}) = 2x_1 \leq 18 \wedge x_1' - x_1 \leq -1 \wedge x_1 - x_1' \leq 1$ (example 4.3). It is satisfiable as there are no negative weighted cycles.

### 4.3.2 Matrix and Graph Representations

After transforming an octagon $\varphi$ into its DB representation we can naturally use the same definition 4.5 for the constraint graph and 4.6 for the matrix representation of $\varphi$. The only thing to note here is the change in size from $N$ to $2N$ because we split each $x_i$ into $y_{2i-1}$ and $y_{2i}$. This results in the constraint graph depicted in Figure 4 and the following matrix representation.

**Example 4.5** (DBM $M_\varphi$ of octagon 4.3)**.**

$$\begin{array}{c} \\ y_1 \\ y_2 \\ y_1' \\ y_2' \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_1' & y_2' \end{array} \\ \begin{pmatrix} 0 & 18 & 1 & \infty \\ \infty & 0 & \infty & -1 \\ -1 & \infty & 0 & \infty \\ \infty & 1 & \infty & 0 \end{pmatrix} \end{array}$$

As with the matrix representation of DBRs, the ordering of variables will not change throughout this thesis an will thus not always be annotated. We define an octagon $\varphi$ to be coherent iff $M_{ij} = M_{\bar{j}\bar{i}}$ for all $1 \leq i, j \leq N$. This is property is important because, for example, the constraint $x_1' - x_1 \leq -1$ is represented by both $y_1' - y_1 \leq -1$ and $y_2 - y_2' \leq -1$. While the definition of the closed form of DBMs (def. 4.8) remains the same, we further define the tight closure of the DBM of an octagonal relation.

**Definition 4.10** (Tight Closure of a DBM m)**.** *An octagonal-consistent coherent DBM $M \in \mathbb{Z}_\infty^{2N \times 2N}$ is said to be tightly closed if and only if the following hold, for all $1 \leq i, j, k \leq 2N$:*

1. $M_{ii} = 0$
2. $M_{i\bar{i}}$ *is even*

3. $M_{ij} \leq M_{ik} + M_{kj}$

4. $M_{ij} \leq \lfloor \frac{M_{i\bar{i}}}{2} \rfloor + \lfloor \frac{M_{\bar{j}j}}{2} \rfloor$

Similarly the definition of consistency (4.7) remains the same but we additionally define octagonal consistency.

**Definition 4.11** (Octagonal Consistency [2]). *Let $M \in \mathbb{Z}_{\infty}^{2N \times 2N}$ be a coherent DBM. Then $M$ is octagonalconsistent if and only if $M$ is consistent and $\left\lfloor \frac{M^*_{ii}}{2} \right\rfloor + \left\lfloor \frac{M^*_{ii}}{2} \right\rfloor \geq 0$, for all $1 \leq i \leq 2N$. Moreover, if $M$ is octagonal-consistent, the tight closure of $M$ is the DBM $M^t \in \mathbb{Z}_{\infty}^{2N \times 2N}$ defined as:*

$$M^t_{ij} = \min \left\{ M^*_{ij}, \left\lfloor \frac{M^*_{i\bar{i}}}{2} \right\rfloor + \left\lfloor \frac{M^*_{\bar{j}j}}{2} \right\rfloor \right\}$$

*for all $1 \leq i, j \leq 2N$ where $M^* \in \mathbb{Z}_{\infty}^{2N \times 2N}$ is the closure of $M$.*

Let us demonstrate all of this on an example. For this recall the previous example (exmpl. 4.5) which showed the DBM corresponding to the leading example (exmpl. 4.3). We first calculate its closure $M*$ using the integer Floyd-Warshall algorithm (see sec. 5.4). We use that result to calculate the tight closure using definition its definition (def. 4.10). Because the closure is equal to the tight closure we get the same result for both. This is depicted below (exmpl. 4.6).

**Example 4.6** (Tight Closure $M^t_{\varphi}$ of octagon 4.3).

$$\begin{array}{c} \\ y_1 \\ y_2 \\ y'_1 \\ y'_2 \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y'_1 & y'_2 \end{array} \\ \begin{pmatrix} 0 & 18 & 1 & 17 \\ \infty & 0 & \infty & -1 \\ -1 & 17 & 0 & 16 \\ \infty & 1 & \infty & 0 \end{pmatrix} \end{array}$$

## 4.4 Powers of Relations

Now that we understand how to calculate with relations, let us discuss their powers. Recall that the second power of a relation $R$ is denoted as $R^2 = R \circ R$. Also, recall that we can depict $R$ as graph $G_R$. With this we can interpret $R^m$ as the $m$ times unfolding of $G_R$, meaning the $m-times$ concatenation of $G$ with itself. This is formally defined below.

**Definition 4.12** (Unfolding of $G$ [13]). *Let $R(\mathbf{x}, \mathbf{x}'), x = \{x_1, \ldots, x_N\}$, be a difference bounds relation and $G_R$ be its constraint graph. The $m$-times unfolding of $G_R$ is defined as $G^m_R = \left( \bigcup_{k=0}^{N} \mathbf{x}^{(k)}, \rightarrow \right)$, where $\mathbf{x}^{(k)} = \left\{ x^{(k)}_i \mid 0 \leq i \leq N \right\}$ and for all $0 \leq k < N$,*

1. $\left( x^{(k)}_i \xrightarrow{c} x^{(k)}_j \right) \in \rightarrow$ *if and only if* $(x_i - x_j \leq c) \in AP(\varphi)$

2. $\left( x^{(k)}_i \xrightarrow{c} x^{(k+1)}_j \right) \in \rightarrow$ *if and only if* $\left(x_i - x'_j \leq c\right) \in AP(\varphi)$

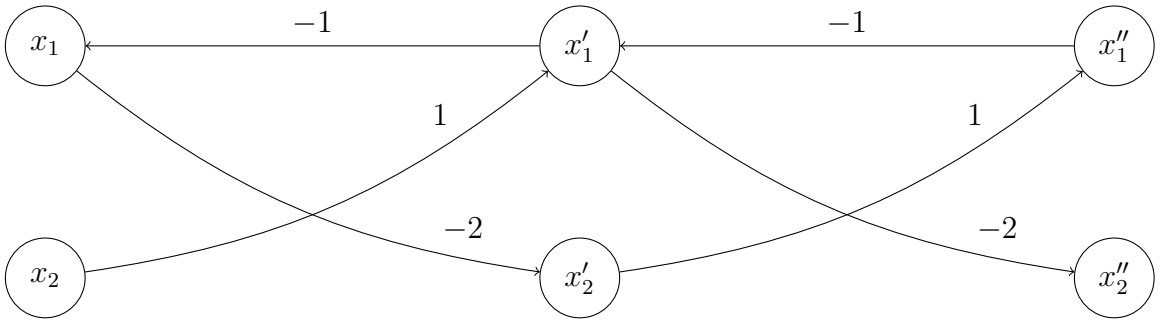


Figure 5: The two times unfolding of the graph Graph $G_\varphi$ that is corresponding to the Difference Bounds Relation $\varphi = x_1 - x'_2 \leq -2 \wedge x_2 - x'_1 \leq 1 \wedge x'_1 - x_1 \leq -1$ (example 4.1). It is not satisfiable as there is a negatively weighted cycle in it ($x_1 \to x'_2 \to x''_1 \to x'_1 \to x_1$).

3. $\left(x_i^{(k+1)} \xrightarrow{c} x_j^{(k)}\right) \in \to$ *if and only if* $(x'_i - x_j \leq c) \in AP(\varphi)$

4. $\left(x_i^{(k+1)} \xrightarrow{c} x_j^{(k+1)}\right) \in \to$ *if and only if* $\left(x'_i - x'_j \leq c\right) \in AP(\varphi)$

*Each constraint in $R^m$ corresponds to a path between extremal points in $G_R^m$. Notice that, since $\mathcal{R}_{db}$ is closed under relational composition, then $R^m \in \mathcal{R}_{db}$ for any $m > 0$. Then we have:*

$$R^m \Leftrightarrow \bigwedge_{1 \leq i,j \leq N} \begin{array}{l} x_i - x_j \leq \min\left\{x_i^0 \to x_j^0\right\} \wedge x'_i - x'_j \leq \min\left\{x_i^m \to x_j^m\right\} \wedge \\ x_i - x'_j \leq \min\left\{x_i^0 \to x_j^m\right\} \wedge x'_i - x_j \leq \min\left\{x_i^m \to x_j^0\right\} \end{array}$$

*where $\min x_i^p \to x_j^q$ is the minimal weight between all paths among the extremal vertices $x_i^p$ and $x_j^q$ in $G_R^m$, for $p, q \in \{0, m\}$.*

We call a vertex extremal if it is either of the form $x_i^0$ or $x_i^m$ in $G_R^m$. Naturally an extremal path denotes a path between between two extremal vertices. We call a path elementary if each vertex in the graph was visited once at maximum. Instead of saying power of relation we may also use the phrases step in the relation or iteration of the relation. To give a better intuition for this, recall the graph of the DB leading example 4.7. We now know that this is the one times unfolding of $G_R$ and represents the relation after one step. With definition 4.12 we can now depict the second step in the relation, $R^2$. This is depicted in Figure 5. Its corresponding DBM is depicted below.

**Example 4.7** (DBM $M_\varphi$ of DBR 4.1 as returned by the Floyd-Warshall (FW) algorithm (see sec- 5.4))**.**

$$\begin{array}{c} \\ x_1 \\ x_2 \\ x'_1 \\ x'_2 \\ x''_1 \\ x''_2 \end{array} \begin{array}{c} \begin{array}{cccccc} x_1 & x_2 & x'_1 & x'_2 & x''_1 & x''_2 \end{array} \\ \begin{pmatrix} -6 & \infty & -5 & -5 & -1 & -7 \\ -6 & 0 & -5 & -5 & -1 & -7 \\ -7 & \infty & -6 & -6 & -2 & -8 \\ -4 & \infty & -3 & -3 & 1 & -5 \\ -5 & \infty & -4 & -4 & -3 & -6 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix} \end{array}$$

The center diagonal contains negative values thus $R^2$ is inconsistent. Notice how the shortest elementary cylce from $X_1 \to x_1$ in $G_\varphi$ (fig. 5) would have the weight $-3$ but since the FW algorithm can only detect negative cylces not calculate elementary ones $M_{11} = -6$ instead. Other values get affected as a consequence. The second part of definition 4.12 also states that the DBM is equivalent to one that erases all non-extremal vertices in $G^2_R$. It is thus equivalent to the following DBM.

$$\begin{array}{c} \\ x_1 \\ x_2 \\ x_1'' \\ x_2'' \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_1'' & x_2'' \end{array} \\ \begin{pmatrix} -6 & \infty & -1 & -7 \\ -6 & 0 & -1 & -7 \\ -5 & \infty & -3 & -6 \\ \infty & \infty & \infty & 0 \end{pmatrix} \end{array}$$

We consider $x_1''$ ($x_2''$) to now be $x_1'$ ($x_2'$) in the next step of the relation.

## 4.5 Periodicity

We have already talked about the concept of periodicity. This chapter focuses on the meaning of periodicity in the context of relations. Three matrices are defined to be periodic if there exists a fourth matrix $\Lambda$. As we have seen we can represent the powers of a relation $R$ as graphs and thus as incidence matrices. It follows that we can calculate the periodicity of $R$ by first computing three of its powers and then calculating whether not there exists a matrix $\Lambda$ and two integers $b \geq 1, c \geq 0$ such that

$$\sigma(R^b) + \Lambda = \sigma(R^{b+c}) \wedge \ \sigma(R^{b+c}) + \Lambda = \sigma(R^{b+2c})$$

We can also rewrite this equation as follows.

$$\sigma(R^b) + 1 \cdot \Lambda = \sigma(R^{b+1 \cdot c}) \wedge \ \sigma(R^b) + 2 \cdot \Lambda = \sigma(R^{b+2c})$$

With this, we call a relation with prefix $b > 0$, period $c \geq 0$ and rate $\Lambda$ to be $*$-periodic iff the following holds for all $l \geq 0$.

$$\sigma(R^b) + l \cdot \Lambda = \sigma(R^{b+l \cdot c})$$

Notice that even if the period is not $c = 1$ we can construct all other powers by composing $\sigma(R^{b+l \cdot c})$ with $R^0, R^1, ..., R^{c-1}$ respectively. We is formally notate it below.

$$\sigma(R^{b+l \cdot c}) \circ \bigvee_{j=0}^{c-1} R^j \tag{1}$$

This results of the property that composing any power of $R^m$ with $R^0$ results in $R^m$ while the composition of $R^m$ with $R^1$ results in $R^{m+1}$. Formally, for any $n, m \geq 0$:

$$R^m \circ R^n = R^{m+n}$$

As an example of this, consider a $*$-periodic relation $R$ with $b = c = 2$. We know all even-powered relations $R^2, R^4, R^6, ...$ to be representable by $\sigma(R^{1+l\cdot1})$. We know composing $\sigma(R^{1+l\cdot1})$ with $R^0$, the identity of $R$, is equivalent to $\sigma(R^{1+l\cdot1})$. We can also represent all odd-powered relations $R^3, R^5, R^7, ...$ after the prefix $b = 2$ by composing $\sigma(R^{1+l\cdot1})$ with $R^1$.

# 5 Implementation

This section demonstrates the algorithm to compute the transitive closure of periodic relations as proposed by Konečný [13]. We implement this method in a library called "Scout" which can be accessed on GitHub through the link below[3].

`https://github.com/TorbenSteegmann/Scout`

In section 6 we will evaluate the results and propose an optimization for the calculation of min terms which get introduced in section 5.4.

## 5.1 Code Philosophy

**Optimization:** When optimizing code one should first clarify what one is optimizing it for. Some common choices include readability and maintainability optimization, memory complexity optimization and speed optimization. In the implementation we will consider all of these fields as important, however we will mostly focus on speed optimization. Readability and maintainability optimization are not as important to us because the program will be implemented as a library and thus largely considered complete. While memory complexity optimization was certainly very important in the early days of the computer where a Macintosh was restricted to 128KB of RAM, it has progressively gotten less important over time as modern systems memory has increased. For these reasons we are only going to optimize these fields using a "non-pessimistic" [18] approach. For memory optimization, this means that, while coding, we ask the question whether or not it is necessary to store a value or data structure. If the answer to that question is not "no" it is considered to be "yes". We will not focus on finding a hyper-optimized storage solution as we should realistically never run into memory problems, even if the memory complexity is slightly underperforming. Decreased memory complexity is also often a trade-off with increased time complexity. As an example, consider two options when calculating the powers of relations:

1. Calculate each power of a relation every time it is needed.
2. Store each power of a relation in a map once it has been calculated.

[3] Note that the repository might receive future updates. The commit of the version discussed here is tagged as "Thesis".

It is easy to see that the first option saves memory by not storing each power separately but the second is much faster, as every power of relation has to be calculated at most once. Similarly, we will use this "non-pessimistic" approach to optimize the speed of the program.

While coding, we will try to minimize the implemented code. This will result in an implementation that avoids loading the CPU with unnecessary calculations, even if there possibly exists a better solution. Only after the implementation is completed we try to find optimizations. One example of this is, again, the calculation of the powers of the relation. In both the non-pessimistic and the optimized approach we store a power in a map where the key is the power and the value is the matrix representation of the power of the relation. Now we know that we will never load the CPU with an unnecessary calculation of such a matrix. Now imagine for some random calculation we need the powers $R^2, R^4, R^6$ but our map contains only $R^1$ and $R^2$. In the non-pessimistic approach we would find $R^2$ and compose it with $R^1$ two times, resulting in the Relations $R^3, R^4$. The same is then repeated for $R^4$, resulting in $R^5, R^6$. One can see that this implementation will be implementable in really few lines of code for which even the calculations of $R^3, R^5$ are not unnecessary as they are used to calculate the higher powers and may be reused in future calculations anyways. In the optimization step we realize that $R^4$ can also be calculated by composing $R^2$ with itself, which avoids the calculation of $R^3$ which we may not end up needing. Similarly $R^6$ can be calculated by composing $R^4$ with $R^2$, avoiding the calculation of $R^5$.

We could achieve another level of optimization by approximating the amount of expected operations of the program and then comparing it against the amount of expected operations the CPU can handle in a time frame. By doing this we could detect whether or not the implementation is slower than what is to be expected of it. However, because the time of most program runs is usually measured in milliseconds that would be unrealistic as the time deviation in between two runs of the same program on the same relation is too big for us to be able to draw definite conclusions. It is usually a technique used for programs with a higher run time as things like scheduling do not play as much of a role here. Instead, we are going to measure the programs run time against the existing implementation from Flata.

In addition to all of this, we will consider the following findings from C. Muratori.

**'Clean' Code - Horrible Performance:** When reading modern implementation guidelines one may often find the term "clean code" being used to describe a well written program. While the term clean code by itself has no clear definition and can be subjectively interpreted differently from one programmer to another, a general consensus is that good code should be aiming to maximize readability and maintainability [1]. For instance, in one chapter of his book "Clean Code - Refactoring, Patterns, Testen und Techniken für sauberen Code" [15], R. C. Martin describes naming conventions for variables, functions and classes. He also goes into detail on the proper way to comment code. Although it is easy to argue that "transitiveClosure" is a more easily readable and understandable name for a variable that holds the transitive closure of a relation than a name like "t" (for transitive closure)

or even "v" (for vector of relations), guidelines like these are ultimately subject to personal preferences. What is important however, is that they obviously do not alter the performance of a program. Consequently they are not harmful to use, even if one subjectively interprets them to be false or bad. This is not true for all proposed rules however. Below is a list of often proposed guidelines that can be proven to decrease performance:

1. Prefer polymorphism to "if/else" and "switch"
2. Code should not know about the internals of objects it is working with (Law of Demeter)
3. Functions should be small
4. Functions should do one thing

In his article " 'Clean' Code, Horrible Performance" [17] C. Muratori describes how the usage of these rules can result in unnecessarily slow programs. By refactoring an existing example of a "clean" program that implemented all of the aforementioned guidelines and choosing to ignore them, he was able to measure a performance increase in speed by up to 25 times. He also mentions the often cited rule "DRY - Do not Repeat Yourself". He explains that this rule can be ignored for small instances of repetitions, but is mostly sensible as repeating the exact same code twice may cause unnecessary bugs when one instance of that code gets changed but the other gets forgotten in the process. Lastly, he states that even in instances where one may chose to violate this rule for performance, the performance increase is hardly ever impactful.

These results are important to us, because our program aims to maximize performance. In our implementation once can see that polymorphism is only used as an efficient way to represent infinity. One may find that the second rule "Code should not know about the internals of objects it is working with" is largely followed, but this is a direct consequence of the program not working with a lot of objects to begin with. Lastly, you will find that functions will mostly do more than "one thing". While functions are also implemented as a way to give a general structure to the code, their primary use is to minimize code repetition.

## 5.2 Precomputations

For the algorithm to work it is important that we feed octagonal or difference bounds relations into it exclusively. For that reason we will first need to identify whether or not the input relation is of one of these types. We implement a Parser that takes a file path as input. It reads the file and filters out the relation from the rest of the file's text. It does that by reading every symbol from the first ':' to the first ';' in the file. This is because the test library it was build to parse defines its relation in between these symbols. An example of such an input file is:

**Example 5.1** (Example Input)**.**

```
nts one;

R1: x_2>-1 && x>0 && x_3>=x_1 && x_1>-1;
print R1^+k;
```

The parser would read the relation of this input as:

```
x_2>-1 && x>0 && x_3>=x_1 && x_1>-1
```

We then tokenize the resulting string recursively, separating each conjunct, or term, from one another. This will result in a token stream that can easily be restructured into the desired form of $\pm x_i - \pm x_j \leq c$. It is important to note that this parsing is not complete, meaning that not all relations that are equivalent to octagons or DBRs can be interpreted as such. For instance, let us look at the relation $x_1 - x_2 \leq x_3 - x_4 \leq 0$. We can see that it is equivalent to an octagon as it is equivalent to $x_1 - x_2 \leq 0 \wedge x_3 - x_4 \leq 0$. To be able to handle this and many other examples we would need to build a more powerful parsing tree which would in turn require a massive codebase to be able to handle all of these niche examples. As the parser is not the main focus of this thesis, we instead define a set reasonable rules that our parser should be able to handle:

1. A term may include exactly one or two variables. It may include more, iff the additional variables can be negated within the term itself. For instance, a term $x_1 - x_2 + x_3 - x_3 \leq 0 \Leftrightarrow x_1 - x_2 \leq 0$ can be parsed, however $x_1 - x_2 - x_3 \leq 0$ can not, even it could be reformed in conjunction with another term like $x_3 = 0$.

2. A term must contain exactly one comparator (<, >, <= , >=, =).

3. A term can only contain multiplication, iff the absolute value of the factor of both variables is equal and the constant is divisible by that factor. An example where multiplication is allowed is $3 \cdot x_1 + 3 \cdot x_2 \leq 3$. An example where it is not be allowed is $3 \cdot x_1 + 5 \cdot x_2 \leq 3$. In addition to this, variable multiplication of the form $x_1 \cdot x_2$ is also not allowed.

With these rules we are able to parse reasonably well written difference bounds and octagonal relations with great efficiency. We then form the matrix representations of the relation.

## 5.3 Pseudo Code

Now that we got all the precomputations out of the way we can look at the actual algorithm that results in the transitive closure of an octagon or DBR. We define $\sigma(R)$ to be the matrix representation $M_R$ of $R$ and we define $p(M_R)$ to be the inverse of that, meaning it translates the matrix back into the corresponding relation. With this notation $p(\sigma(R)) \Leftrightarrow R$. Let us look at a slight variation of the Transitive Closure Algorithm presented in [13]:

---

Algorithm 1 Transitive Closures of Periodic Relations

---

**input** octagon or DBR R
**output** The transitive closure of R

1: **function** TransitiveClosure(R)
2: **let** $b \leftarrow 1$ **and** $b_{jump} \leftarrow 1$
3: **while** true **do**
4: **for all** $c = 1, 2, ..., b$ **do**
5: **for all** $l = 0, 1, 2$ **do**
6: **if** $R^{b+lc} \Leftrightarrow \bot$ **then**
7: **return** $R^+ \Leftrightarrow P \vee (\bigvee_{i=b}^{b+lc-1} R^i)$
8: **if** $\exists \Lambda \,.\, \sigma(R^b) + \Lambda = \sigma(R^{b+c})$
$\wedge\, \sigma(R^{b+c}) + \Lambda = \sigma(R^{b+2c})$ **then**
9: $K \leftarrow MaxConsistent(R, b, \Lambda)$
10: $L \leftarrow MaxPeriodic(R, b, \Lambda, c, K)$
11: **if** $(L = \infty)$ **then**
12: **return** $P \vee \exists k \geq 0 \,.\, p(k \cdot \Lambda + \sigma(R^b)) \circ (\bigvee_{j=0}^{c-1} R^j)$
13: $b_{jump} \leftarrow max\{b_{jump}, b + c \cdot (L + 1)\}$
14: $b_{next} \leftarrow max\{b + 1, b_{jump}\}$
15: $P \leftarrow P \vee \bigvee_{i=b}^{b_{next}-1} R^i$
16: $b \leftarrow b_{next}$
17: **function** $MaxConsistent(R, b, \Lambda)$
18: **return** $sup\{n \in \mathbf{N} \mid p(n \cdot \Lambda + \sigma(R^b)) \not\Leftrightarrow \bot\}$
19: **function** $MaxPeriodic(R, b, \Lambda, c, K)$
20: **return** $sup\{n \leq K \mid \forall 0 \leq l < n \,.$
$p(l \cdot \Lambda + \sigma(R^b)) \circ R^c \Leftrightarrow p((l + 1) \cdot \Lambda + \sigma(R^b))\}$

---

The changes to the original algorithm in [13] are as follows:

...
// 2: **let** $P \leftarrow R$ **and** $b \leftarrow 1$ **and** $b_{jump} \leftarrow 1$ (old)
2: **let** $b \leftarrow 1$ **and** $b_{jump} \leftarrow 1$ //(new)
...
// 7: **return** $R^+ \Leftrightarrow P \vee (\bigvee_{i=b+1}^{b+lc-1} R^i)$ (old)
7: **return** $R^+ \Leftrightarrow P \vee (\bigvee_{i=b}^{b+lc-1} R^i)$ //(new)

One contribution of the thesis is to proof that the old pseudo code was slightly incorrect in a special case. This also shows that the initialization of $P$ in line 2 is redundant. We show this with a counter example:

*Old Code Counter Example:* Let us consider an arbitrary DBR or Octagon $R$ that has the following attributes:

1. $R$ is consistent for $R^1, ..., R^4$
2. $R$ is periodic from $R^1, ..., R^3$ but not from $R^3$ to $R^4$

3. $R$ is inconsistent in $R^5$

The specific choice of $R$ is not important. In the beginning (line 2) we set $P = R^1$. We know that $b = 1, c = 1$ in the first iteration of the loop in line 4. Because of the first attribute of $R$ we know that the test in line 6 fails for all $0 \leq l \leq 2$. We know the test in line 8 succeeds because of the second attribute of $R$. From the third attribute we know that $K = 4$. From the second attribute we can conclude that $L = 2$. Because of this the test in line 11 will fail. With this we know $b_{jump} = max\{b_{jump}, b + c \cdot (L + 1)\} = max\{1, 1 + 1 \cdot (2 + 1)\} = 4$ which leads to $b_{next} = max\{b+1, b_{jump}\} = max\{1+1, 4\} = 4$. In line 15 $P$ is now expanded with all powers of $R$ from $R^b = R^1$ up to $R^{b_{next}-1} = R^3$. In this case this means the updated value of $P$ is $P = R^1 \vee R^1 \vee R^2 \vee R^3$. Here we can see that $P = R^1$ in line 2 is redundant. We show that it is still redundant even if the test in line 11 succeeds. The returned transitive closure would then be:

$$
\begin{aligned}
& P \vee \exists k \geq 0.p(k \cdot \Lambda + \sigma(R^b)) \circ (\bigvee_{j=0}^{c-1} R^j) \\
& = R^1 \vee \exists k \geq 0.p(k \cdot \Lambda + \sigma(R^1)) \circ (R^0) \\
& \Leftrightarrow R^1 \vee \exists k \geq 0.p(k \cdot \Lambda + \sigma(R^1))
\end{aligned}
$$

We substitute the existence quantified $k$ with $n$ using skolemizaion leading to $R^1 \vee p(n \cdot \Lambda + \sigma(R^1))$. For $n = 0$ this leads to $R^1 \vee p(0 \cdot \Lambda + \sigma(R^1) \Leftrightarrow R^1 \vee R^1$ showing the redundancy of the initialization of $P$ in line 2. Again, the precise choice of $R$ does not matter.

It remains to be shown that the returned value for the transitive closure in line 7 is false. In the next loop iteration $b = 4, c = 1$. We know that the test in line 6 will fail for $R^{b+0c} = R^4$ because of the first attribute of $R$. Lastly we know that the test in line 6 will succeed and thus conclude the algorithm for $R^{b+1c} = R^5$ and return the transitive closure as $R^+ = R^1 \vee R^1 \vee R^2 \vee R^3$. One can see that $R^4$ is missing even though it is consistent (attribute 1) and the transitive closure is defined as the disjunction of all powers up to the first one which is inconsistent [13]. □

Another important consideration is that line 8 can return too high powers of relations, meaning powers for which a relation $R$ is inconsistent. To illustrate this let us consider the following example where the relation $R$ has the following attributes.

1. $R$ is consistent for $R^1, ..., R^4$ but not $R^5, R^6$
2. $R$ is does is not periodic in $R^1, ..., R^4$

With this, because of the first attribute of $R$ we know the algorithm will start with $b = c = 1$ and fail the test in line 6 for all iterations of the loop in line 5. We also know that the test in line 8 will fail because of attribute 2 and thus $b_{next} = 2$ in line 14. For similar reasons we also know that these same tests will fail for the configuration $b = 2, c = 1$. With the next configuration $b = c = 2$ however, the test in line 6 will succeed for $l = 2$ because we know $R^{2+2*2} = R^6$ is inconsistent from

attribute 1. Because $R^5$ is never checked, we do not find that it is inconsistent thus it is returned as a disjunct of the transitive closure in line 7. Because inconsistent relations are equivalent to false ($\bot$) however, this is not a problem as a formula $\varphi$ with a disjunct $\bot$ is equivalent to $\varphi$ without that disjunct $\bot$.

$$\varphi \vee \bot \Leftrightarrow \varphi$$

With all of this in mind let us discuss the functionality of algorithm 1. The algorithm aims to identify integers $b \geq 0$ and $c > 0$ such that the sequence $\{\sigma(R^{b+nc})\}_{n\geq 0}$ becomes periodic. This guarantees termination in line 7 or 12, provided that the relation $R$ is periodic [13]. For each prefix-period candidate $(b, c)$, the algorithm checks the consistency of the first three equidistant powers, namely $R^b$, $R^{b+c}$, and $R^{b+2c}$, and if they are consistent, the algorithm computes the first rate of the sequence in line 8. The rate computation involves calculating the distance $\Lambda$ between $\sigma(R^{b+c})$ and $\sigma(R^b)$ and the distance between $\sigma(R^{b+2c})$ and $\sigma(R^{b+c})$. If these distances are equal, then $\Lambda$ is considered as a potential candidate for the rate of the sequence $\{\sigma(R^{b+nc})\}_{n\geq 0}$. However, if the consistency check fails, the relation is not $*$-consistent, and the transitive closure is the disjunction of all powers up to the first one which is inconsistent, terminating the algorithm in line 7. In sec. 4.4 we have shown that the leading example (4.1) of DBRs is inconsistent after one step in the relation. For this reason the algorithm terminates in line 7 after passing the test in line 6 for $b = c = l = 1$. To be able to demonstrate further steps of the algorithm we will slightly alter the formula of 4.1. The new formula is depicted below.

**Example 5.2** (Altered DBR Leading Example)**.**

$$\varphi = x_1 - x_2' \leq 2 \wedge x_2 - x_1' \leq 1 \wedge x_1' - x_1 \leq -1$$

All we did was change the weight $c$ of the conjunct $x_1 - x_2' \leq -2$ to $c = 2$. Obviously the graph in Figure 2 does still represent this DBR if you change the weight of the edge from $x_1 \to x_2'$ accordingly. When we talk about the "Leading Example" from this point forth, we will be referring to this slightly altered version 5.2. With this consideration let us continue explaining the algorithm. After checking the distances and identifying the potential candidate for the rate of the sequence, the algorithm proceeds to validate the choices of $b$, $c$, and $\Lambda$ by performing additional checks. Specifically, the algorithm checks on lines 9-11 whether the sequence of relations $\{p(n \cdot \Lambda + \sigma(R^b))\}_{n\geq 0}$ is $*$-consistent, and that it follows the pattern of a periodic sequence. By the definition of MaxPeriodic, either $L = K = \infty$, or else $L \leq K$. If the test on line 11 succeeds for the chosen $b$, $c$, and $\Lambda$, then $\pi(k \cdot \Lambda + \sigma(R^b))$ is the closed form of the sequence $\{\sigma(R^{b+nc})\}_{n\geq 0}$, and consequently, the transitive closure is returned in line 12. This is ensured by the following definitions.

**Definition 5.1.** *Let $R$ be a periodic relation, let $b > 0$, $c > 0$ be integers such that $R^b \not\Rightarrow \bot$, and let $\Lambda \in \mathbb{Z}_\infty^{m\times m}$ be a matrix such that, for all $n \geq 0$:*

1. $p(n \cdot \Lambda + \sigma(R^b)) \not\Rightarrow \bot$*, and*
2. $p((n+1) \cdot \Lambda + \sigma(R^b)) \Leftrightarrow p(n \cdot \Lambda + \sigma(R^b)) \circ R^c$.

*Then, $R$ is $*$-consistent and $\hat{R}_{b,c}(k, \vec{x}, \vec{x}') \Leftrightarrow \pi(k \cdot \Lambda + \sigma(R^b))$ [13].*

With this, we can define the transitive closure as follows [13].

**Definition 5.2.** *Let $R$ be a relation, $b > 0$, $c > 0$ be arbitrary integers and $\hat{R}_{b,c}(l, x, x')$ be the closed form of the infinite sequence $\{R^{b+nc}\}_{n \geq 0}$. Then, the transitive closure of $R$ can be defined as:*

$$R^+ \Leftrightarrow (\bigvee_{i=1}^{b-1} R^i) \vee \exists l \geq 0.\ \pi(k \cdot \Lambda + \sigma(R^b)) \circ (\bigvee_{j=0}^{c-1} R^j) \tag{2}$$

In combination with Definition 5.1 this shows that the return values of Lines 7 and 12 are indeed the transitive closure.

## 5.4 (Parametric) Floyd-Warshall Algorithm

Recall that two relation are said to be equivalent iff the closed form, or tightly closed form for octagons, of their corresponding matrix representations are equal. A matrix $M \in \mathbb{Z}_{\infty}^{N \times N}$ is said to be closed iff $M_{ii} = 0$ and $M_{ij} \leq M_{ik} + M_{kj}$ for all $1 \leq i, j, k \leq N$. For matrices that contain integer values this is known to be computable by the Floyd-Warshall algorithm.

---

Algorithm 2 Integer Floyd-Warshall Algorithm

---

**input** Matrix $M$ of size $m$ of a relation $R$
**output** The closed form $M^*$ of $R$

1: **function** $IntegerFW(M)$
2: **for all** $k = 1, 2, ..., m-1$ **do**
3: **for all** $i = 1, 2, ..., m-1$ **do**
4: **for all** $j = 1, 2, ..., m-1$ **do**
5: $I_0 \leftarrow M[i][j]$
6: $I_1 \leftarrow M[i][k]$
7: $I_2 \leftarrow M[k][j]$
8: $M[i][j] \leftarrow min\{I_0, I_1 + I_2\}$
9: **return** $M$

---

Next, let us discuss how to calculate the tightly closed form of an octagonal relation. Recall that the tight closure of an octagon is defined as:

$$M_{ij}^t = \min\{M_{ij}^*,\ \lfloor \frac{M_{i\bar{i}}^*}{2} \rfloor + \lfloor \frac{M_{\bar{j}j}^*}{2} \rfloor\}$$

Where $M^*$ is the closure and $M^t$ is the tight closure. For integer matrices we can expand the Floyd-Warshall algorithm with the following to calculate the tight closure of an octagon. The correctness of this is proven in [2].

---

Algorithm 2.5 Integer Tight Closure Algorithm

---

**input** Closure $M^*$ of size $m$ of a relation $R$
**output** The tightly closed form $M^t$ of $R$

1: **function** $TightClosure(M*)$
2: **for all** $i = 1, 2, ..., m-1$ **do**
4: **for all** $j = 1, 2, ..., m-1$ **do**
5: $I_0 \leftarrow M^*[i][j]$
6: $I_1 \leftarrow \lfloor \frac{M^*[i][i]}{2} \rfloor$
7: $I_2 \leftarrow \lfloor \frac{M^*[j][j]}{2} \rfloor$
8: $M[i][j] \leftarrow min\{I_0, I_1 + I_2\}$
9: **return** $M$

---

Looking back at algorithm 1 (5.3) one can see that integer matrices will not be the only ones we need to be able to handle. The value of k is not known during the calculations of MaxConsistent and MaxPeriodic and it remains a variable in the transitive closure. For this reason we need to be able to work with the matrices of the form $k \cdot \Lambda + \sigma(R^b)$. One can see that the elements in the cells of this matrix are not Integer values, but univariate linear terms of the form $\alpha \cdot k + \beta$ where $\alpha, \beta \in \mathbb{Z}$ instead. To be able to handle these terms let us introduce a few notations. Let $S, T \subseteq \mathbf{Z}[k]$. We define $S \oplus T = \{(\alpha_1+\alpha_2)\cdot k+\beta_1+\beta_2 \mid \alpha_1 \cdot k+\beta_1 \in S, \alpha_2 \cdot k+\beta_2 \in T\}$. The partial order of two terms $t_1$ and $t_2$ is defined as $t_1 \preceq t_2 \Leftrightarrow \alpha_1 \leq \alpha_2 \wedge \beta_1 \leq \beta_2$. The strict inequality of two terms is $t_1 \prec t_2 \Leftrightarrow t_1 \preceq t_2 \wedge t_2 \not\preceq t_1$. We define the minimal set of linear terms S by $\mathrm{MinTerms}(S) = \{t \in S \mid \forall s \in S \,.\, s \not\prec t\}$. As you can see the minimum of two linear terms is not necessarily a linear term. For this reason a matrix must not only hold integer values, nor just linear terms, but sets of linear terms. We represent $\infty$ with the empty set $\{\varnothing\}$. This definition of min terms is sufficient for now but we will later see how it can lead to an unnecessarily large sets of min terms which will decrease performance. We will also discuss how to solve this issue as one of the main contributions of this thesis.

With these considerations let us adapt the integer Floyd-Warshall algorithm to handle parametric matrices.

---

Algorithm 3 Parametric Floyd-Warshall Algorithm

---

**input** Matrix $M$ of size $m$ of a relation $R$
**output** The closed form $M^*$ of $R$

```
 1: function ParametricFW(M)
 2:     for all k = 1, 2, ..., m − 1 do
 3:         for all i = 1, 2, ..., m − 1 do
 4:             for all j = 1, 2, ..., m − 1 do
 5:                 T_0 ← M[i][j]
 6:                 T_1 ← M[i][k]
 7:                 T_2 ← M[k][j]
 8:                 M[i][j] ← MinTerms(T_0 ∪ (T_1 ⊕ T_2))
 9:     return M // M[i][j] in [13] (typo)
10: function MinTerms(S)
11:     return {t ∈ S | ∀s ∈ S.s ⊀ t}
```

---

Notice how this results in the same procedure as its integer counterpart, if every value of $\alpha$ in $S$ the same. For this reason we can redefine our integer matrices to be parametric matrices where every value of $\alpha$ is equal to 0. This will allow us to use the same Floyd-Warshall algorithm for both types of matrices eliminating the need for polymorphism or switch statements. We do also no longer have to think about representing infinity in integer matrices as we can use the same encoding that we use for parametric matrices where infinity is represented by a set of size $0$. We will discuss how to calculate the tight closure of parametric matrices in 5.8. It is important to note that if $M$ contains a negative cylce this may not always return only elementary paths. This will not lead to wrong results in MaxConsistent and MaxPeriodic, however it increases the execution time of algorithm 1. For now, we improve the result of algorithm 3 for these cases slightly by adding the condition that line 8 will only execute if $i \neq j \wedge i \neq k \wedge j \neq k$. This way we can at least somewhat limit the amount a negative cycle counts itself down because it stops the graph from finding a shorter way from vertex $x$ over itself to itself. We will discuss possible methods to improve this in section 8.

With these results we can discuss how to implement the procedures for MaxConsistent and MaxPeriodic.

## 5.5 MaxConsistent for Difference Bounds Relations

Recall that at the point where Algorithm 1 (5.3) calls MaxConsistent, we have already calculated a rate $\Lambda$ such that $R^{b+2c} \Leftrightarrow (2 \cdot \Lambda + \sigma(R^b))$. From the consistency check in line 6 we also know that $(l \cdot \Lambda + \sigma(R^b))$ is consistent for all $0 \leq l \leq 2$. With this in mind, MaxConsistent returns the maximal integer value $n \geq 2$ such that $(n \cdot \Lambda + \sigma(R^b))$ is consistent. For example, if the Relation $(2 \cdot \Lambda + \sigma(R^b))$ is consistent, but $(3 \cdot \Lambda + \sigma(R^b))$ is not, MaxConsistent will return the value $2$. If a relation $R$ is $*$-consistent then MaxConsistent returns $\infty$ instead. We denote

$M_{R,b,\Lambda}[n] = (n \cdot \Lambda + \sigma(R^b)) \in \mathbf{Z}_\infty^{2N \times 2N}$ and $M(n)$ to be the return value of Algorithm 3 (sec. 5.4) with $M_{R,b,\Lambda}[n]$ as its input. Because returning the maximum value $n$ for which $(n \cdot \Lambda + \sigma(R^b))$ is consistent is equivalent to returning the first one that is inconsistent and decrementing that value by one, we can define MaxConsistent as follows [13]:

**Definition 5.3** (MaxConsistent)**.**

$$\textit{MaxConsistent}(R, b, \Lambda) = \inf\{n \in \mathbb{N} \mid M_{R,b,\Lambda}[n] \textit{ is inconsistent}\} - 1. \tag{3}$$

The inconsistency of a parametric matrix $M(n)$ is defined as the existence of a strictly negative elementary cycle for some valuation of $n \in \mathbf{N}$. For a fixed value of n this is the same as for the non parametric counterpart and would thus be equivalently done by first calculating the shortest paths and then looking to see whether or not there is a value smaller than 0 on the center diagonal of the matrix.

Recall that the cells of $M(n)$ are vectors of univariate terms of the form $\alpha \cdot n + \beta$. The following lemma aims to identify a value for n such that $\alpha \cdot n + \beta < 0$ [13].

**Lemma 5.1** ([13])**.** *Let $T = \{\alpha_i \cdot k + \beta_i\}_{i=1}^m$ be a set of univariate linear terms and $l \in \mathbb{N}$ be a constant. Then $\min_{n \geq l} T(n) < 0$ if and only if there exists $1 \leq i \leq m$ such that either $\alpha_i < 0$, or $\alpha_i \cdot l + \beta_i < 0$. Moreover, the smallest value $n$ such that $\min_{n \geq l}\{\alpha_i \cdot n + \beta_i\}_{i=1}^m < 0$ is $\Gamma(T) = \min_{j=1}^m \gamma_j$, where:*

$$\gamma_j = \begin{cases} \max(l, \lfloor -\frac{\beta_j}{\alpha_j} \rfloor + 1) & \textit{if } \alpha_j < 0 \\ l & \textit{if } \alpha_j \geq 0 \textit{ and } \alpha_j \cdot l + \beta_j < 0 \\ \infty & \textit{otherwise} \end{cases}$$

Because we know that $2 \cdot \Lambda + \sigma(R^b)$ is consistent, we also know that $M_{ii} = \alpha_i \cdot l + \beta_i \geq 0$ with $0 \leq l \leq 2$ and $0 \leq i \leq 2N$. For $l = 0$ this results in $\beta \geq 0$. With this we could conclude that $\alpha_i \cdot 2 \geq -\beta_i$ however, if there exists a negative cycle in $M$, then $ParametricFW$ does not necessarily return elementary paths. Because we know $\beta \geq 0$ and $M_{ii} = \alpha_i \cdot 2 + \beta_i \geq 0$, we can also conclude that $(\lfloor -\frac{\beta_j}{\alpha_j} \rfloor + 1) > 2$ if $\alpha < 0$. We keep these conditions for similar reasons as these assumptions are only true in elementary paths and if there exists a cycle with a negative value of $\alpha$ it may get more negative with each iteration of the loops in the $ParametricFW$ algorithm (5.4) and to the point where $\alpha > -\beta$. One might also raise the valid question of why we increment the value of $\lfloor -\frac{\beta_j}{\alpha_j} \rfloor$ by $1$ just to decrement the return value of MaxConsistent by $1$ when we could just not increment the value in the first place. While this is correct for DBRs, we will want to reuse this function when we extend MaxConsistent to work with octagons where we will need this incremention (see sec. 5.7). With this in mind we can could reduce equation (5.1) the following way, if we could ensure elementary paths. We will discuss how we could use this for better results in sec 8.

$$\gamma_j = \begin{cases} \lfloor -\frac{\beta_j}{\alpha_j} + 1 \rfloor & \text{if } \alpha_j < 0 \\ \infty & \text{otherwise} \end{cases} \tag{4}$$

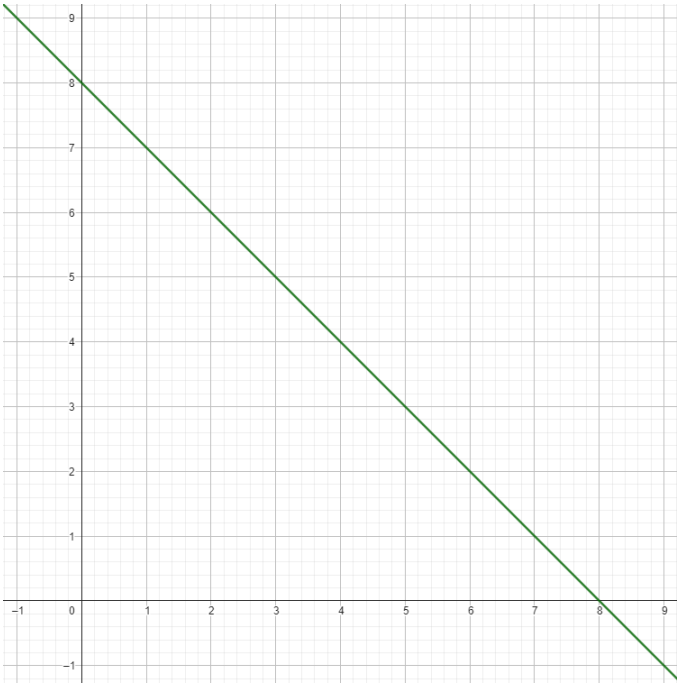

Figure 6: Visual representation of MaxConsistent of the term $t_1 = -1n+8$ (green). Obviously $t_1 \geq 0$ for all $n \leq 8$ and $t_1 < 0$ for all $n > 8$.

We use this equation on every term of every cell $M_{ii}$ with $0 \leq i \leq 2N$ and return the minimum value $n$ that we find across all such cells. A visual representation of the return value of MaxConsistent can be found in Figure 6.

Given a DBR $R$, integers $b \geq 0, c > 0$ such that $R^b$ is consistent and a Matrix $\Lambda \in \mathbf{Z}_{\infty}^{2N \times 2N}$, MaxConsistent runs in time at most $O((b+c)^3 \cdot ||R||^3 \cdot N^9)$ where $||R||$ denotes the sum of absolute values of the coefficients of a relation [13].

## 5.6 MaxPeriodic for Difference Bounds Relations

Similar to MaxConsistent, MaxPeriodic returns the maximum value $K \geq n \geq 2$ such that $p(n \cdot \Lambda + \sigma(R^b)) \circ R^c \Leftrightarrow p((n+1) \cdot \Lambda + \sigma(R^b))$. This means we want to find the maximum value $n \geq 0$ so that, given a distance matrix $\Lambda$ and $M(n)$, we can calculate the power of the next period ($R^{b+n \cdot c+c}$) by increasing the value of $n$ in the formula $p(n \cdot \Lambda + \sigma(R^b))$ by one. If this holds for all values of $n$ and $MaxConsistent = \infty$ we return $\infty$ instead. We call the following the left hand side of the equivalence which gets encoded in a matrix $M_1$.

$$p(n \cdot \Lambda + \sigma(R^b)) \circ R^c$$

Similarly the following is the right side of the equivalence and gets encoded in a Matrix $M_2$.

$$p((n+1) \cdot \Lambda + \sigma(R^b))$$

We first calculate the closed form of $M_1$ with algorithm 3 (sec. 5.4). This results in a matrix of min terms of the form $min\{t_i\}_{i=1}^{m}$. Remember that these sets

can also be empty in which case they represent $\infty$. Because we know $\Lambda$ to be closed and $\sigma(R^b)$ is also closed it follows that $M_2$ is also closed [13]. This is a result of the following Lemma [13].

**Lemma 5.2.** *Let $R \in R_{db}$ be a difference bounds relation, and $\Lambda$ be the rate of the periodic sequence $\{\sigma(R_i)\}_{i=0}^{\infty}$. Then, for all $b \geq 0$ and $n > 0$, the DBM $n \cdot \Lambda + \sigma(R^b)$ is closed.*

For this reason $M_2$ is simply a matrix of univariate terms instead of a matrix with sets of such terms as $M_1$ is. Because we want to show the equivalence of $M_1$ and $M_2$ we have hence got to check the equivalence of the min terms in each cell of $M_1$ and univariate terms in each cell of $M_2$. As previously mentioned we do this by finding the maximum value $n$ for which $M_1 \Leftrightarrow M_2$. This is equivalent to the minimal value $min\{\alpha_i \cdot n + \beta_i\}_{i=1}^{m} = \alpha_0 \cdot n + \beta_0$ across all cells of $M_1$ and $M_2$. Here, $m$ is the number of terms in a cell. The term $\alpha_0 \cdot n + \beta_0$ represents the corresponding cell of $M_2$. We save the minimal value across all cells for which the condition holds and increment it by one. This is because the found value represents the highest periodic power of $R$, meaning $R^{b+n \cdot c}$ to $R^{b+(n+1) \cdot c}$ is still periodic and $R^{b+(n+1) \cdot c}$ is the first power which is not periodic. Konečný [13] formally describes the method to calculate this as follows:

**Lemma 5.3.** *Let $T = \{\alpha_i \cdot k + \beta_i\}_{i=1}^{m}$ be a set of univariate linear terms, $t_0 = \alpha_0 \cdot k + \beta_0 \in \mathbb{Z}[k]$ be a term, and $l \in \mathbb{N}$ be a constant. Then there exists an integer $\kappa > l + 1$ such that $\min T(q) = t_0(q)$, for all $l \leq q \leq \kappa$, if and only if the following hold:*

1. $\bigvee_{i=1}^{m} (\alpha_i = \alpha_0 \wedge \beta_i = \beta_0) \wedge \bigwedge_{i=1}^{m} \bigwedge_{j=0}^{2} [\alpha_0 \cdot (l + j) + \beta_0 \leq \alpha_i \cdot (l + j) + \beta_i]$
2. $\kappa \leq \min \left\{ \left\lfloor \frac{\beta_i - \beta_0}{\alpha_0 - \alpha_i} \right\rfloor \mid 1 \leq i \leq m, \alpha_0 \neq \alpha_i, \left\lfloor \frac{\beta_i - \beta_0}{\alpha_0 - \alpha_i} \right\rfloor > l + 1 \right\}$

This definition is defined to broadly for our use case. Because we already know that $p(n \cdot \Lambda + \sigma(R^b)) \circ R^c \Leftrightarrow p((n+1) \cdot \Lambda + \sigma(R^b))$ holds for $0 \leq n \leq 2$ we already know that condition must hold for $l = 0$ and we do not need to check for it. All that this condition says is that the term $t_0$ is also contained in $T$, and that no term in $T$ has a smaller value than $\alpha_0 \cdot n + \beta_0$ for $0 \leq n \leq 2$. Condition two states that, if a term different from $t_0$ exists in $T$, then the maximum value $\kappa$ for which $t_0$ is greater than that term $t_i \in T$, is equal to $\left\lfloor \frac{\beta_i - \beta_0}{\alpha_0 - \alpha_i} \right\rfloor$. This is equivalent to the intersection of the terms in the two dimensional states after which $t_i < t_0$. This can also be represented visually as shown in Figure 7. It is important to note that we do not need to iterate over different values of $l$ and can just set $l = 0$. This results from the fact that lemma 5.3 returns the same value for all choices of $l$ until $\kappa < l + 1$. We do need the sub-conditions $1 \leq i \leq m$ and $\alpha_0 \neq \alpha_i$. What we do not need is the third sub-condition $\left\lfloor \frac{\beta_i - \beta_0}{\alpha_0 - \alpha_i} \right\rfloor > l + 1$. This is because, again, we already know this to be the case for $l = 0$ since $n \geq 2$. However because we know that algorithm 3 (sec. 5.4) does not return elementary paths for graphs with negative cylces we will still need to keep this third subcondition for now. With this we can reduce the conditions of lemma 5.3 as follows:

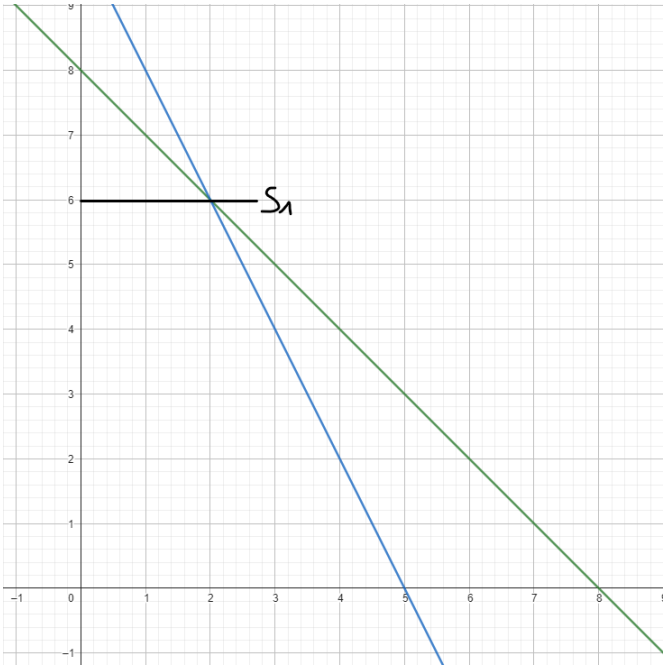


Figure 7: Visual representation of MaxPeriodic of the term $t_0 = -1n + 8$ (green) and the set of min terms $T = \{t_1 = -1n + 8$ (green)$, t_2 = -2n + 16$ (blue)$\}$. One can see that $t_1 < t_2$ for all $0 \leq n < 6$. They intersect in $s_1$ for $n = 6$ after which $t_1 > t_2$ for all $n \geq 6$.

$$\kappa \leq \min \left\{ \left\lfloor \frac{\beta_i - \beta_0}{\alpha_0 - \alpha_i} \right\rfloor \mid 1 \leq i \leq m, \alpha_0 \neq \alpha_i, \left\lfloor \frac{\beta_i - \beta_0}{\alpha_0 - \alpha_i} \right\rfloor > l + 1 \right\} \tag{5}$$

If there exists no term so that $\alpha_0 \neq \alpha_i$ then $\kappa = \infty$. As aforementioned we return the minimal value $\kappa + 1 = n \leq K$ we find across all cells.

Given a DBR $R$, integers $b \geq 0, c > 0$ such that $R^b$ is consistent and a Matrix $\Lambda \in \mathbf{Z}_\infty^{2N \times 2N}$, MaxPeriodic runs in time at most $O((b + c)^3 \cdot ||R||^3 \cdot N^9)$ [13].

## 5.7 MaxConsistent for Octagonal Relations

As was the case for DBRs, MaxConsistent for octagons returns the value of the largest value $n \geq 0$ such that $n \cdot \Lambda + R^b$ is consistent. Again, because we already computed the first two rates of this to be consistent, we know that $n \geq 2$. Recall that a relation is defined to be octagonal consistent if its matrix representation $M \in \mathbb{Z}_\infty^{2N \times 2N}$ is coherent and it satisfies the following:

1. $M$ is consistent $\Leftrightarrow M_{ii}^* \geq 0$, for all $1 \leq i \leq N$
2. $\left\lfloor \frac{M_{i\bar{i}}^*}{2} \right\rfloor + \left\lfloor \frac{M_{\bar{i}i}^*}{2} \right\rfloor \geq 0$, for all $1 \leq i \leq 2N$

We can check condition one the same way we did for regular DBRs thus returns a value $K_{db}$. For the second condition we have to consider the fact that rounding down a negatively, uneven value of $\alpha$ results will result in false results. For this

consider the example for two cells $M_{i\bar{i}}$ and $M_{\bar{i}i}$ that both contain the same term $-5 \cdot k + 10$. Obviously this term is equal to $0$ for $k = 2$. Halving this term and adding it to itself results in the term $-6k + 5$ which is smaller than $0$ for $k = 2$ thus changes the result. The following lemma offers a solution to this problem [13].

**Lemma 5.4.** *Let* $T = \left\{ t_i + \sum_{j=1}^{m_i} \left\lfloor \frac{u_{ij}}{2} \right\rfloor \right\}_{i=1}^{n}$, *where* $t_i = \alpha_i \cdot k + \beta_i, u_{ij} = \gamma_{ij} \cdot k + \delta_{ij}$ *are univariate linear terms. Then there exist two sets* $\mathcal{L}, \mathcal{U}$ *of univariate linear terms such that for all* $k \geq 0$ *:*

$$\min\{\mathcal{L}(k)\} = \min\{T(2k)\}$$
$$\min\{\mathcal{U}(k)\} = \min\{T(2k+1)\}$$

*Moreover* $|\mathcal{L}\| \leq \|T\|$ *and* $|\mathcal{U}\| \leq \|T\|$.

This is defined to broadly for our use case as we will define $t_i = 0$ and $m_i = 2$ so that each term in $T$ has the following form.

$$\min \left[ \left\lfloor \frac{t}{2} \right\rfloor + \left\lfloor \frac{u}{2} \right\rfloor \mid t \in M_{i\bar{i}}, u \in M_{\bar{i}i} \right]_{\geq 0}, \text{ for some } 1 \leq i \leq 4N.$$

With this consideration we calculate the sets $\mathcal{L}(k)$ and $\mathcal{U}(k)$ the following way.

$$\mathcal{L} = \left\{ \left( \sum_{j=1}^{m_i} \gamma_{ij} \right) \cdot k + \sum_{j=1}^{m_i} \left\lfloor \frac{\delta_{ij}}{2} \right\rfloor \right\}_{i=1}^{n}$$
$$\mathcal{U} = \left\{ \left( \sum_{j=1}^{m_i} \gamma_{ij} \right) \cdot k + \sum_{j=1}^{m_i} \left\lfloor \frac{\gamma_{ij} + \delta_{ij}}{2} \right\rfloor \right\}_{i=1}^{n}$$

Doing this for all pairs of cells $M_{i\bar{i}}$ and $M_{\bar{i}i}$ we denote the following.

$$K_{db} = \min \left\{ \Gamma \left( \mathcal{M}_{ii} \right) \right\}_{i=1}^{4N},$$
$$K_{\mathcal{L}} = \min \left\{ \Gamma \left( \mathcal{L}_i \right) \right\}_{i=1}^{4N},$$
$$K_{\mathcal{U}} = \min \left\{ \Gamma \left( \mathcal{U}_i \right) \right\}_{i=1}^{4N}$$

Applying this to our example where $M_{i\bar{i}} = M_{\bar{i}i} = \{-5k + 10\}$ will result in the following sets $\mathcal{L}$ and $\mathcal{U}$.

$$\mathcal{L} = \{(-5-5) \cdot k + \frac{10}{2} + \frac{10}{2}\} = \{-10 \cdot k + 10\}$$
$$\mathcal{U} = \{(-5-5) \cdot k + \left\lfloor \frac{-5+10}{2} \right\rfloor + \left\lfloor \frac{-5+10}{2} \right\rfloor\} = \{-10 \cdot k + 8\}$$

All of these values $K$ are computed with lemma 5.1. With this we know that $K_{\mathcal{L}} = 2$ and $K_{\mathcal{U}} = 2$. MaxConsistent will then return:

$$\text{MaxConsistent}(R, b, \Lambda) = \min \left\{ K_{db}, 2 \cdot K_{\mathcal{L}}, 2 \cdot K_{\mathcal{U}} - 1 \right\} - 1$$

With this in mind the previous example returns the following.

$$\min\{K_{db}, (2 \cdot 2), (2 \cdot 2 - 1)\} - 1 = 2$$

The value of other cells does not matter as we know $\mathrm{MaxConsistent} \geq 2$ (see sec. 5.5).

## 5.8 MaxPeriodic for Octagonal Relations

Similar to its DB counterpart (sec. 5.6), MaxPeriodic for octagon relations returns the maximal positive integer $K \geq n \geq 2$ such that the following equivalence holds for all $0 \leq l \leq n$.

$$p(n \cdot \Lambda + \sigma(R^b)) \circ R^c \Leftrightarrow p((n+1) \cdot \Lambda + \sigma(R^b))$$

We call the following the left hand side of the equivalence which gets encoded in a matrix $M_1$.

$$p(n \cdot \Lambda + \sigma(R^b)) \circ R^c$$

Similarly the following is the right side of the equivalence and gets encoded in a Matrix $M_2$.

$$p((n+1) \cdot \Lambda + \sigma(R^b))$$

The right hand side can be shown to be tightly closed for all valuations of $n$. The result of this is that we can encode $M_2$ as a matrix of univariate terms instead of sets of such terms.

**Lemma 5.5** ([13]). *Let $R \in R_{oct}$ be an octagonal relation, and $\Lambda$ be the rate of the periodic sequence $\sigma(R^i)_{i=0}^{\infty}$. Then, for all $b, n \geq 0$, the DBM $(n+1) \cdot \Lambda + \sigma(R^b)$ is tightly closed.*

Matrix $M_1$ is still needed to be calculated by algorithm 3 (see sec. 5.4) and is thus a matrix of sets of univariate terms and sums of half terms after the tightening step. The result of this is that verifying $M_1 \Leftrightarrow M_2$ requires us to check the equivalence of the sets of univariate terms and sums of half terms in $M_1$ and the univariate terms in $M_2$. Calculating these half terms results in the same issues that we had in MaxConsistent (see sec. 5.7). For this reason we will once again use lemma 5.4 but this time not only for a $M_{i\bar{i}} + M_{\bar{i}i}$ but all combinations of cells $M_{i\bar{i}} + M_{\bar{j}j}$. This is a consequence of the definition of the tight closure of octagons (see def. 4.10). Applying lemma 5.4 to all such cells will split $M_1$ into $M_{1L}$ and $M_{1U}$ where $min\{(M_{1L})_{ij}(k)\} = min\{(M_{ij}(2k)\}$ and $min\{(M_{1U})_{ij}(k)\} = min\{Mij(2k+1)\}$ for all $k \geq 0$. We split $M_2$ into $M_{2L}$ and $M_{2U}$ in a similar way, but remember that $M_2 = p((n+1) \cdot \Lambda + \sigma(R^b))$ and thus term of each cell is $\alpha_{ij} \cdot (k+1) + \beta_{ij} = \alpha_{ij} \cdot (k) + \beta_{ij} + \alpha_{ij}$. We can now use lemma 5.3 on every cell of the equivalences $M_{1L} \Leftrightarrow M_{2L}$ and $M_{1U} \Leftrightarrow M_{2U}$ resulting in the values $P_L$ and $P_U$ respectively. Then MaxPeriodic returns the max interval in which $M_1 \Leftrightarrow M_2$ as $min\{2 \cdot P_L, 2 \cdot P_U - 1\} - 1$.

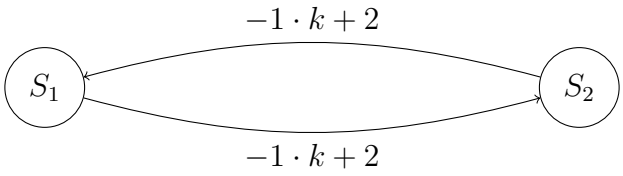


Figure 8: The vertices in the graph above represent two parts of an arbitrary graph. They are connected by two edges that form a cycle that has a combined weight such that $\alpha < 0$ and $\beta > 0$. Here the cycle has the weight $-2 \cdot n + 4$ but the precise choice does not matter.

# 6 Evaluation and Optimization

This section will propose an optimization on the definition of min terms used in the algorithms that were presented in section 5. We will then evaluate the program by running it over a collection of almost 1100 example octagons. In the next section (sec. 7) we will then compare the results to those of the existing tool Flata. We will draw a conclusion and give insights to future work in sec. 8.

## 6.1 Improvement of the Definition of Min Terms

In section 5.4 we defined a term $t \in T$ to be a min term if there exists no other term $s \in T$ for which $s \prec t$. In other words, if a term $t$ is part of a set of terms $T$ then $T$ does not contain another term $s$ that is strictly smaller than $t$. This definition is problematic if the graph that the parametric FW algorithm iterates over contains a cycle that has a negative weight for $\alpha$ and a positive weight for $\beta$. An example of such a graph is depicted in Figure 8. Here, the cycle has a combined weight of $-2 \cdot n + 4$. It is easy to see that iterating the parametric FW algorithm over such a graph would add another min term to every cell that is part of this cycle in every iteration of every loop. This is because using this loop twice will result in a combined weight of $-4 \cdot n + 8$ which is again, by definition, a min term. Even worse, in the next iteration of the loop the combined weight consist of the set of min terms $\{-2 \cdot n + 4, -4 \cdot n + 8\}$. This means after this iteration the size of this set grows not by $1$ as it did in the first iteration but by $2$ instead. The resulting set is $\{-2 \cdot n + 4, -4 \cdot n + 8, -6 \cdot n + 12, -8 \cdot n + 16\}$. One can tell that this has terrible implication for the run time of the algorithm whenever such a cycle exists. In practice, during the testing of Scout, such program runs had to be forcefully terminated. The longest measured runtime before such a manual termination had to be enforced was over $9$ hours.

The proposed solution of this problem serves as a main contribution of this thesis. The solution works as follows. When considering whether or not a term is a min term the original method of verifying this only involved comparing pairs terms separately. However if three or more such terms exist in the set not every one of them is necessarily minimal for any valuation of $n$. Consider the following set of min terms:


Figure 9: Visual Representation of three terms of a set of min Terms $T = \{t_0 = -2 \cdot n + 4(green), t_1 = -8 \cdot n + 16(blue), t_2 = -16 \cdot n + 32(red)\}$ where $t_1$ will never actually be minimal for any valuation of $n$.

$$T = \{t_0 = -2 \cdot n + 4, t_1 = -8 \cdot n + 16, t_2 = -16 \cdot n + 32\}$$

Obviously, all three of these evaluate to $0$ for $n = 2$. For all $n < 2$, $t_0$ will be the smallest term. For all $n > 2$, $t_2$ will be the smallest term. The term $t_1$ will never be minimal thus we do not consider it a min term anymore. This is visualized in Figure 9.

We can compute this as follows. For each set of min terms $T$, if the set contains at least three min terms, we compare each triple of pairwise distinct terms. In each triple we determine which term has the smallest value for $\beta$, which we will call $t_\beta$, and which term has the smallest value for $\alpha$, which we will call $t_\alpha$. Because of the earlier definition of min terms we know these to be two different terms. Next we calculate their point of intersection. After this we calculate the point of intersection of $t_\beta$ and the third term which we will call $t_{mid}$. If $t_\beta$ intersects with $t_{mid}$ in the same point or after $t_\beta$ intersects with $t_\alpha$ then we know that $t_{mid}$ will never be minimal and we will thus no longer consider it a min term. The updated version of a set of min terms is thus computable by:

$$T = \{t \in T | \forall s \in T.s \not< t, \nexists u \in T. \frac{\beta_s - \beta_u}{\alpha_u - \alpha_s} \leq \frac{\beta_t - \beta_s}{\alpha_u - \alpha_t}, \alpha_s > \alpha_t > \alpha_u, \beta_u > \beta_t > \beta_s\}$$

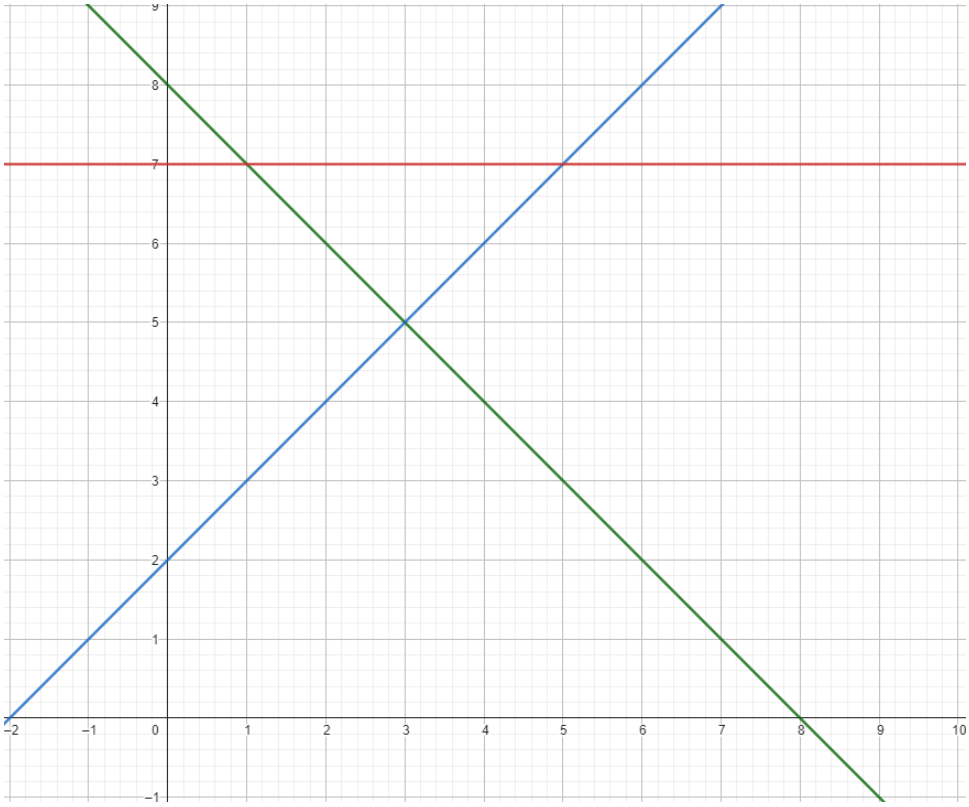


Figure 10: Visual Representation of three terms of a set of min Terms $T = \{t_0 = -1 \cdot n + 8(green), t_1 = 1 \cdot n + 2(blue), t_2 = 0 \cdot n + 7(red)\}$ where $t_2$ will never actually be minimal for any valuation of $n$.

The primary run time improvement of this will be when encountering the aforementioned types of loops. In practice, the same run of the program that did not terminate earlier, now computes in milliseconds. It will also reduce the amount of min terms in relations that do not contain such loops. This is depicted in Figure 10.

## 6.2 Evaluation

This section will evaluate the run time performance of the implemented program. As mentioned in section 5, we implement the program as a library called Scout. We use a dataset of almost 1100 octagonal relations and let Scout compute their transitive closures. We do this by running a bash script that executes Scout for all files in the directory that contains the dataset. We time the total execution time as well as the execution time of each individual program run. When we speak of the execution of different programs we mean the execution of Scout with different relations as its input. The processor used in these tests is an "Intel Core I7-4790K CPU" which was not overclocked. We run the script twice and use the average time from both runs. This helps reduce the effect CPU noises such as scheduling differences, have on the results. With this the results are as follows. The total execution time for the entire bash script averaged out at $4.4$ seconds.

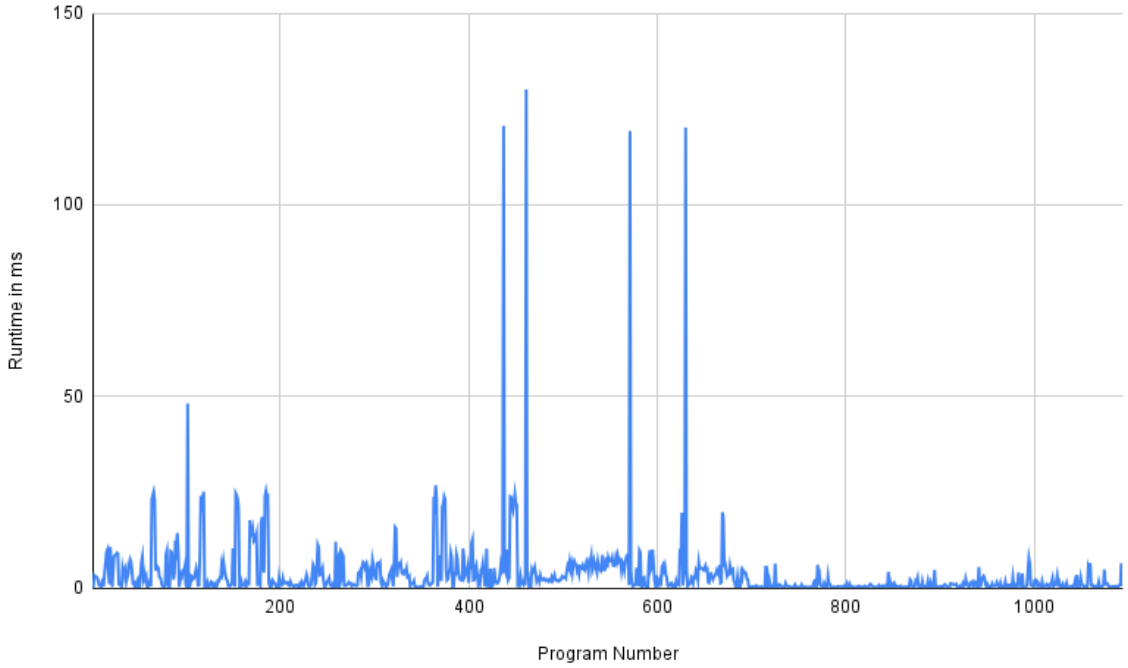


Figure 11: Run Time of Scout. The $x$-axis is the program number. The $y$-axis is the execution time in milliseconds.

The average execution time of specific program runs was $3.6$ milliseconds with the median execution time being $1.3$ milliseconds. The maximum execution time was $130$ milliseconds while the minimum execution time was $0.16$ milliseconds. The results are demonstrated in a scatter plot in Figure 11.

# 7 Comparison to Flata

In this section we compare the results obtained in section 6.2 with the result from the same tests with Flata. We thus use a dataset of almost 1100 octagonal relations which we execute Flata with through the use of a bash script. Naturally we use the same CPU as in 6.2. We also run this bash script twice thus all following results are the averages between these runs. With this the total execution time of the bash script averages at $3$ minutes and $15.437$ seconds. The average execution time of a specific program run was $112.6$ milliseconds the median being $108$ milliseconds. The maximum run time was $476$ milliseconds with the minimal being $83$ milliseconds. All results are presented in the scatter graph of Figure 12.

The output transitive closures for these tests were saved as files in an SMT-Lib version 2.6 format. They were evaluated for equivalency using the Z3-SMT solver. We assume the output of Flata to be correct. This test verified the correctness of Scouts results in almost all cases. The cases where Z3 did not verify the equivalence of the transitive closures all had in common, that they returned in line 7 of algorithm 1 (see sec. 5.3). Here, Flata's output format drastically

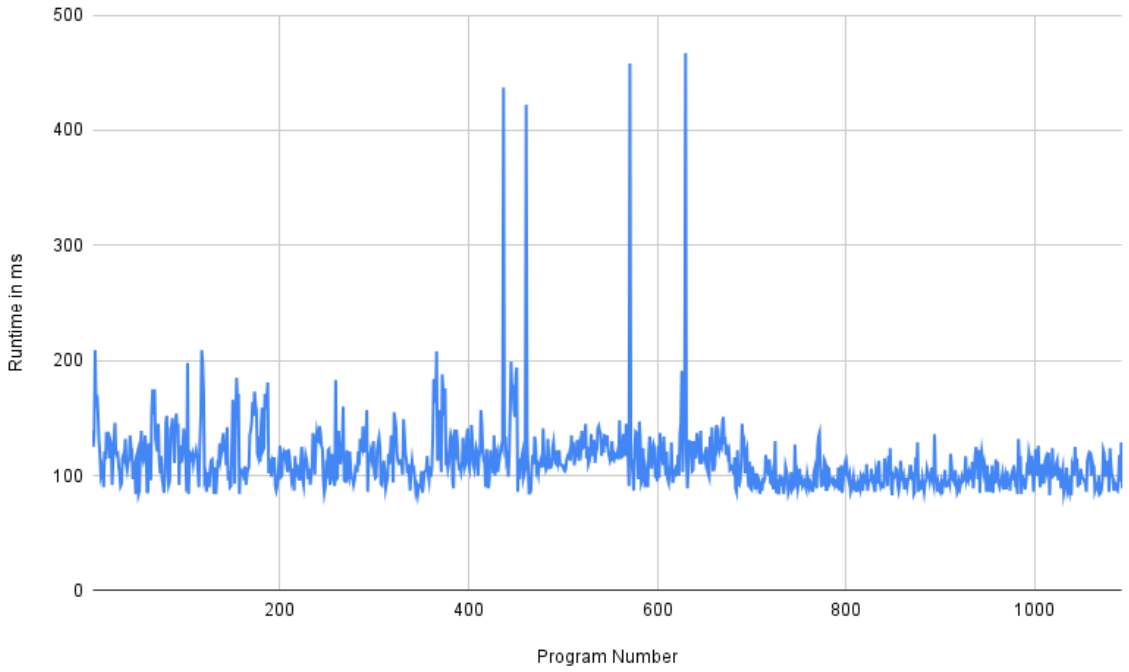


Figure 12: Run Time of Scout. The $x$-axis is the program number. The $y$-axis is the execution time in milliseconds.

differs from the output format of the one that returns at line 12. To illustrate this consider the following. Suppose the transitive closure of a relation $R$ consists of all powers of $R$ up to $R^7$ which is inconsistent. For this, Scout would return $R^1 \vee R^2 \vee R^3 \vee R^4 \vee R^5 \vee R^6$. Flata however will sometimes introduce a variable $k$ such that the returned transitive closure is, for example, of the form $(\varphi_1 \wedge k \geq 0 \wedge k \leq 3) \vee (\varphi_2 \wedge k \geq 4 \wedge k \leq 6)$. Because this variable $k$ does not exists in the transitive closure returned by Scout, Z3 will not return equivalency. Even if, for example, $R^2 \Leftrightarrow (\varphi_1 \wedge k \geq 0 \wedge k \leq 3)$ for $k = 2$, Z3 can set $R^2$ to true and $(\varphi_1 \wedge k \geq 0 \wedge k \leq 3)$ to false by choosing $k = 4$. I have verified the correctness of about $10\%$ of these examples by hand and have concluded that the returned value of Scout was correct for all them, however automating this verification process is not feasible within the scope of this thesis thus it is important to mention that there exists the possibility of a margin for error.

Looking back at the execution times of both implementations however, one can see that Scout offers huge improvement to Flata. While Flata took about $195$ seconds to compute the transitive closures of all relations in the dataset, Scout only takes about $4.4$ seconds. This is a speed increase of over $44$ times. Even more drastically, the fastest execution time of flata was $83$ milliseconds, while Scout only took $0.16$ milliseconds for the same program. This is a speed increase of over $518$ times.

Of course, one has to mention that Flata does not only have drawbacks. While Scout calculates the transitive closure with impressive speed, Flata generally outputs them in a nicer, minimized format. Its output does not include redundant

formulas. While redundancies do not make the result wrong, eliminating them results in better readability.

# 8 Conclusion and Future Work

In this thesis we have presented an algorithm for accelerating octagons and implemented it as a library that showed significant efficiency improvements to older implementations. As part of this we have proposed a new definition of min terms that can have a huge impact on the runtime of the proposed algorithm. With this in mind let us conclude this thesis by looking at a few examples of improvements we could make in the future.

**Finite Monoid Affine Relations:** Finite Monoid Affine Relations are another type of periodic relation that can be proven to be periodic [13]. Their constraints have the for $x' = A \times x + b \wedge \phi(x)$. One can see that this differs vastly from octagons and DBRs which is why they were not considered in the scope of this thesis even though the presented algorithm can be adapted to handle them as well.

**Calculation of Elementary Paths:** We have seen that negative cycles may lead to the computation of non elementary paths in a constraint graph. This may result in slower acceleration because the procedure MaxConsistent does not immediately find the first inconsistent power of the relation. One possibility of ensuring that a calculated path is elementary is by storing a calculated path of one note to all others in a separate matrix or list. This would however also decrease the efficiency of the Floyd-Warshall algorithm which we need to use a lot throughout the algorithm. Whether or not this would actually result in performance increases would thus require extensive testing.

**Using Reals instead of Integers** We have only considered constraints of the form $x_i - x_j \leq c$ where c was an integer. Future work may also include expanding the algorithm to be able to work with real values as well. One consideration to make would be how to form a constraint of the form $x_i - x_j < c$ into the form $x_i - x_j \leq c$. We might have to subtract an infinitely small, real value $\epsilon$ from c and keep track of which constraints were effected by this so we can add it back at the end of the algorithms. Some aspects of the algorithm may however also get easier. As an example, we would no longer have to worry about rounding when dividing terms.

# References


[1] Mariano Anaya. *Clean Code in Python: Refactor your legacy code base*. Packt Publishing Ltd, 2018.

[2] Roberto Bagnara, Patricia M Hill, and Enea Zaffanella. “An improved tight closure algorithm for integer octagonal constraints”. In: *Verification, Model Checking, and Abstract Interpretation: 9th International Conference, VMCAI 2008, San Francisco, USA, January 7-9, 2008. Proceedings 9*. Springer. 2008, pp. 8–21.

[3] Gilles Barthe et al. “Probabilistic Relational Verification for Cryptographic Implementations”. In: *SIGPLAN Not.* 49.1 (Jan. 2014), pp. 193–205. ISSN: 0362-1340. DOI: 10.1145/2578855.2535847. URL: https://doi.org/10.1145/2578855.2535847. "(accessed: 01.03.2023)".

[4] Rudolf Berghammer, Peter Höfner, and Insa Stucke. “Automated verification of relational while-programs”. In: *Relational and Algebraic Methods in Computer Science: 14th International Conference, RAMiCS 2014, Marienstatt, Germany, April 28–May 1, 2014. Proceedings 14*. Springer. 2014, pp. 173–190.

[5] Barry W. Boehm. “Verifying and validating software requirements and design specifications”. In: *IEEE software* 1.1 (1984), p. 75.

[6] Marius Bozga, CodruŢa Gîrlea, and Radu Iosif. “Iterating Octagons”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Stefan Kowalewski and Anna Philippou. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 337–351. ISBN: 978-3-642-00768-2.

[7] Anca Browne et al. “An improved algorithm for the evaluation of fixpoint expressions”. In: *Theoretical Computer Science* 178.1-2 (1997), pp. 237–255.

[8] J.R. Burch et al. “Symbolic model checking for sequential circuit verification”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13.4 (1994), pp. 401–424. DOI: 10.1109/43.275352.

[9] Edmund M Clarke et al. “Model checking and the state explosion problem”. In: *Tools for Practical Software Verification: LASER, International Summer School 2011, Elba Island, Italy, Revised Tutorial Lectures* (2012), pp. 1–30.

[10] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2022.

[11] David Detlefs, Greg Nelson, and James B. Saxe. “Simplify: A Theorem Prover for Program Checking”. In: *J. ACM* 52.3 (May 2005), pp. 365–473. ISSN: 0004-5411. DOI: 10.1145/1066100.1066102. URL: https://doi.org/10.1145/1066100.1066102. "(accessed: 25.03.2023)".

[12] Rashina Hoda, Norsaremah Salleh, and John Grundy. “The Rise and Evolution of Agile Software Development”. In: *IEEE Software* 35.5 (2018), pp. 58–63. DOI: 10.1109/MS.2018.290111318.

[13] Filip Konecnỳ. “Relational verification of programs with integer data”. PhD thesis. Citeseer, 2012.

[14] Filip Konečnỳ. “PTIME computation of transitive closures of octagonal relations”. In: *Tools and Algorithms for the Construction and Analysis of Systems: 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings 22*. Springer. 2016, pp. 645–661.

[15] Robert C Martin. *Clean Code-Refactoring, Patterns, Testen und Techniken für sauberen Code: Deutsche Ausgabe*. MITP-Verlags GmbH & Co. KG, 2013.

[16] Paolo Masci et al. “Verification of interactive software for medical devices: PCA infusion pumps and FDA regulation as an example”. In: *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*. 2013, pp. 81–90.

[17] Casey Muratori. *"Clean" Code, Horrible Performance*. https://www.computerenhance.com/p/clean-code-horrible-performance. 2023. "(accessed: 03.03.2023)".

[18] Casey Muratori. *Refterm Lecture Part 1 - Philosophies of Optimization*. https://www.youtube.com/watch?v=pgoetgxecw8&t=388s. 2023. "(accessed: 03.03.2023)".

[19] David Lorge Parnas, GJK Asmis, and Jan Madey. “Assessment of safety-critical software in nuclear power plants.” In: *Nuclear safety* 32.2 (1991), pp. 189–198.

[20] Lawrence C Paulson. “The foundation of a generic theorem prover”. In: *Journal of Automated Reasoning* 5 (1989), pp. 363–397.

[21] Moises Rodriguez, Mario Piattini, and Christof Ebert. “Software Verification and Validation Technologies and Tools”. In: *IEEE Software* 36.2 (2019), pp. 13–24. DOI: 10.1109/MS.2018.2883354.

[22] Ashish Tiwari. “Hybridsal relational abstracter”. In: *Computer Aided Verification: 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings 24*. Springer. 2012, pp. 725–731.

[23] D.R. Wallace and R.U. Fujii. “Software verification and validation: an overview”. In: *IEEE Software* 6.3 (1989), pp. 10–17. DOI: 10.1109/52.28119.

[24] Fang Yu, Tevfik Bultan, and Oscar H Ibarra. “Relational string verification using multi-track automata”. In: *International Journal of Foundations of Computer Science* 22.08 (2011), pp. 1909–1924.