FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Rotating Workforce Scheduling as Satisfiability Modulo Theories

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Christoph Erkinger

Matrikelnummer 0726020

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Priv.-Doz. Dipl.-Ing. Dr.techn. Nysret Musliu

Wien, TT.MM.JJJJ

(Unterschrift Verfasserin)	(Unterschrift Betreuung)

Erklärung zur Verfassung der Arbeit

Christoph Erkinger
Ettenreichgasse 48/2/22, 1100 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasserin)

Für meinen Opa

Danksagung

Zunächst möchte ich mich an dieser Stelle bei all denjenigen bedanken, die mich während der Anfertigung dieser Diplomarbeit unterstützt und motiviert haben.

In erster Linie gilt dieser Dank meinem Betreuer Nysret Musliu, der mit seinem Fachwissen in vielen Diskussionen stets für wertvolle Hinweise sorgte und mich mit kritischem Hinterfragen auf Schwachpunkte aufmerksam machen konnte.

Daneben gilt mein Dank natürlich meinen Eltern, die mich seit nunmehr 24 Jahren immer unterstützen und mir diese Ausbildung erst ermöglichten. Nicht selten konnten sie mich in schwierigen Situationen, während der Schulzeit und auch des Studiums mit ihrem Rückhalt und ihrem Verständnis wieder aufbauen und mir so die notwendige moralische Unterstützung geben die ich brauchte. Bei meiner Oma möchte ich mich dafür bedanken, dass sie immer alles für ihren Enkel tut und in arbeitsintensiven Phasen oft für die kulinarische Verpflegung sorgte.

Nicht zuletzt möchte ich ein ganz besonderes Dankeschön an meine Freundin Corinna richten. Ohne ihre unschätzbar wertvollen Tipps zur strukturellen Gestaltung wissenschaftlicher Arbeiten und dem stundenlangen Korrekturlesen wäre diese Diplomarbeit nicht so geworden wie sie es ist. Außerdem möchte ich mich vor allem für ihre Geduld während dieser gesamten Zeit bedanken und mich dafür entschuldigen, dass ich oft nicht einfach war.

Abstract

Rotating workforce scheduling (RWS) is an important real-life problem that appears in a large number of different business areas. Well-constructed workforce schedules can have a significant impact on labor costs, improve productivity and thus influence a company or organization's efficiency. Designing such schedules includes a number of constraints that have to be fulfilled in order to respect legal restrictions or to consider the employee's individual preferences. This makes RWS a difficult task which sometimes even exceeds the possibilities of modern computer systems.

Over the last decades many approaches to rotating workforce scheduling have been proposed, including exact methods like constraint programming and heuristic techniques. Although exact approaches provide solutions for many instances, solving large problems is still a challenge to them. Therefore, finding other exact methods able to provide better results for larger problems is an important issue in the field of rotating workforce scheduling.

In this thesis, I propose a new approach to RWS by using the recent advances on *Satisfiability Modulo Theories* (SMT). SMT-solving has been successfully applied to a number of different problems in software verification or test-case generation and moreover to constraint satisfaction problems with practical relevance. While solving can be automated by using a number of so-called SMT-solvers, the most challenging task is to find an efficient formulation of the problem in first-order logic. I developed two new modeling techniques for RWS that encode the problem using formulas over different background theories. The first encoding provides an elegant approach based on linear integer arithmetic. Although this modeling technique showed promising results, it reaches its limits when applied to large-scale problems. To overcome this issue, I proposed a new formulation based on bitvectors in order to achieve a more compact representation of the constraints and a reduced number of variables.

These two modeling approaches were experimentally evaluated on benchmark instances from literature using different state-of-the-art SMT-solvers. By comparing the performance of the two encodings to each other, the superiority of the bitvector approach could be determined. Compared to other exact methods, the results of this approach showed a significant improvement in the number of found solutions. The bitvector encoding was able to solve 18 out of 20 benchmark instances, providing a solution to several instances that had not been solved yet by other exact methods.

Kurzfassung

Für viele Organisationen und Unternehmen ist es von entscheidender Bedeutung ihr Personal optimal einzusetzen, um zum einen die Produktivität zu steigern und zum anderen die Personalkosten möglichst gering zu halten. Zur Verbesserung der Auslastung des Personals, werden Dienstpläne zur Einteilung der Mitarbeiter in verschiedene Schichten erstellt. Hierbei müssen neben den Anforderungen des Unternehmens auch gesetzliche Bestimmungen und indivduelle Interessen der Arbeitnehmer berücksichtigt werden. Dies stellt eine Herausforderung dar, welche selbst mit der Unterstützung aktueller Computersysteme im Allgemeinen nicht effizient gelöst werden kann.

In den letzten 40 Jahren wurden zahlreiche computergestützte Methoden zur Lösung von Schichtplanproblemen entwickelt. Dabei wurden neben exakten Verfahren wie dem Constraint Programming auch heuristische Methoden eingesetzt. Exakte Methoden lösen zwar eine große Anzahl an Schichtplanproblemen, können jedoch Pläne für viele Mitarbeiter nicht effizient erstellen. Es ist deshalb von großem Interesse neue exakte Methoden zu entwickeln, welche auch für schwierige Probleme geeignete Dienstpläne berechnen können.

In dieser Arbeit wird ein neues exaktes Verfahren für ein spezifisches Schichtplanproblem, *Rotating Workforce Scheduling* (RWS), vorgestellt. Dafür wird RWS als *Satisfiability Modulo Theories* (SMT) Problem modelliert. SMT beschreibt ein Entscheidungsproblem welches auf der Erfüllbarkeit von Prädikatenlogik erster Stufe unter der Berücksichtigung bestimmter *Theorien* beruht. Während das Lösen von SMT Problemen mit *SMT-Solvern* automatisiert werden kann, stellt das Finden einer geeigneten und effizienten Formulierung in SMT die größte Herausforderung dar. Im Rahmen meiner Diplomarbeit etnwickelte ich zwei Modellierungen für das RWS Problem. Zuerst entwarf ich mit Hilfe von linearer Arithmetik eine elegante Formulierung, die jedoch bei großen Probleminstanzen an ihre Grenzen stößt. Die zweite Modellierung weist diese Limitierung nicht auf, da durch die Verwendung von Bitvektoren und binären Operationen eine kompakte Formulierung und eine Reduktion der Variablen erzielt werden konnte.

Beide Ansätze wurden anhand von 20 Probleminstanzen aus der Literatur experimentell evaluiert, wofür verschiedene SMT-Solver eingesetzt wurden. Die Bitvektor Formulierung erzielte im Vergleich zu jener mit linearer Arithmetik bessere Ergebnisse. Der Vergleich mit anderen Methoden aus der Literatur zeigte außerdem, dass *SMT-Solving* ein kompetitives Verfahren ist, welches Probleminstanzen löst, die andere exakte Verfahren bisher nicht lösen konnten.

Contents

CHAPTER 1

Introduction

For companies and organizations in a broad range of business areas, it is necessary to provide availability 24 hours a day and 7 days a week. While some companies need their machines up and running all the time in order to fit the production demand, other organizations like hospitals or police stations need their staff to be working day and night to ensure proper health-care or the citizens' safety.

Workforce schedules for such staff-dependent organizations and businesses are important for a smooth work flow. In designing them, certain aspects have to be considered: The first challenge for employers is to determine the right amount of staff that is necessary to perform a set of tasks. Although underestimating the staff demand can have a bad impact on the quality of the products or result in less productivity, one must also bear in mind that a higher number of employees drastically influences the labor costs. To ensure enough resting time, typical schedules consist of different shifts, where each shift covers a certain time period during a day. The number of staff required for each shift of a specific work day are called the temporal requirements. Lastly, there are several restrictions by federal law that have to be considered during the process of workforce scheduling.

These legal restrictions and other constraints that are motivated by the employees' individual preferences make workforce scheduling a difficult problem to overcome when designing the schedules by hand. For exactly this reason, workforce scheduling has been a regular focus of research attention in computer science and operational research for more than 40 years. As the complexity of this problem often exceeds even the possibilities of modern computer systems [32], it is not always possible to find a solution for a given problem instance in a reasonable amount of time.

In this thesis the particular problem of interest in personnel scheduling is *Rotating Workforce Scheduling* (RWS), which is a constraint satisfaction problem (CSP). This problem addresses designing a schedule where a certain number of shifts are assigned to work days over a specific planning period, typically one week. The work schedule is designed for each employee by

considering the necessary temporal requirements and other constraints. Furthermore, in rotating workforce scheduling all employees work the same schedule in a rotating sequence starting at different offsets.

Over the years different approaches to this problem have been introduced in the literature. Early methods were based on exhaustive enumeration [58],[29] and Integer Linear Programming [68]. A network flow model was developed in [10] to solve the problem. In [66] a relaxation of the constraints was proposed that allows the creation of efficient rotating schedules by hand. Heuristic techniques were successfully used to create rotating schedules in [73]. Other methods based on constraint programming techniques were applied in [67, 74, 90]. Although these exact approaches provide a solution for many instances, solving large problems is still a challenge to them. Therefore, it would be intriguing to find other exact methods that may provide better results.

One interesting exact approach that has been successfully applied to a number of other CSPs (e.g. [95], [62]) is formulating the problem in propositional logic and thereby reducing it to the *Boolean Satisfiability Problem* (SAT), which may then be solved with a SAT-solver [30, 88, 92]. Recent advances in SAT-solving have made this a powerful alternative. However, encoding CSPs to SAT often results in a large number of formulas, making it "tedious and lengthy" [87] work as stated in [87].

In comparison, SMT-solving offers the expressive power of first-order theories combined with the good performance of state-of-the-art SAT-solvers. Therefore, translating CSPs into SMT provides a more elegant alternative, which was applied to CSPs in [3, 79].

As both SAT and SMT-solving have been efficiently used to solve CSPs other than rotating workforce scheduling, it stands to reason that this solving paradigm is also a promising approach for RWS. Since SMT is the more elegant approach, this thesis investigates how the RWS problem can be encoded in first-order formulas over certain background theories. Moreover, the performance of modern SMT-solvers is evaluated in order to determine whether SMT-solving can achieve better results than other exact methods and thus is a competitive alternative to other approaches from literature.

1.1 Objectives

The objectives of this work are:

- Determination of how rotating workforce scheduling can be formulated as an SMT problem using first-order formulas.
- Investigation of the different background theories' characteristics and development of formulations for RWS that allow a compact representation of the constraints.
- Experimental evaluation of different modeling approaches using benchmark instances from literature and several state-of-the-art SMT-solvers to identify the best performing

formulation. Furthermore, the aim is to compare this approach to other solving techniques from literature in order to evaluate its competitiveness.

1.2 Main Results

The main results of this thesis are:

- I developed two modeling approaches for rotating workforce scheduling that consider different background theories. By formulating the problem in the theory of quantifier-free linear integer arithmetic, most instances could be solved. However, this approach reached its limits when applied to problems with a large number of employees.
- To overcome this issue I developed a second encoding using the theory of fixed-size bitvectors. By means of machine arithmetic I was able to achieve a very compact representation of the constraints and a reduced number of variables.
- I experimentally evaluated the two modeling approaches on benchmark instances from literature using five state-of-the-art SMT-solvers. I was able to achieve promising results for both formulations. Nevertheless, the bitvector technique outperforms the linear arithmetic method on all benchmark instances.
- As the bitvector approach was considered the faster one, I compared its results with other solving techniques from literature, showing that SMT is a well performing alternative to other exact methods like constraint programming and even outperforms these approaches on some instances.

1.3 Organization

This thesis is organized as follows: Chapter 2 discusses personnel scheduling and provides a classification of different problems in this area. Additionally, it gives a formal definition of rotating workforce scheduling and reviews the previous works on this problem. In Chapter 3, the relevant background information on Satisfiability Modulo Theories is provided. Chapter 4 gives a detailed explanation of the two different modeling approaches I developed for solving RWS. In Chapter 5, the experimental results of my approach are presented and compared to other approaches in the literature. In Chapter 6, the thesis is concluded by discussing the implications of the results of this work and outline some ideas for future work.

CHAPTER 2

Personnel Scheduling

Personnel Scheduling or Rostering has become an essential organizational tool in a multitude of business areas. Many companies find it crucial to provide availability 24 hours on 7 days a week and therefore it is imperative that they have a good personnel schedule. The *right* schedule will help to significantly reduce labor costs by determining the correct amount of staff necessary to implement a 24/7 schedule. Moreover, the right schedule also considers day-off periods, individual preferences and legal restrictions in order to increase the employees' satisfaction and productivity.

Personnel scheduling has been a regular focus of research attention for more than 40 years. Due to the vast number of constraints and the possibly long period for which the schedule is to be planned, the search space for this problem can be substantial. Some sub-problems or variations of personnel scheduling are even considered to be NP-complete as Chuin Lau has shown for *shift scheduling* in [32]. Nevertheless, researchers have tried obtaining an automated construction of solutions for the personnel scheduling problem via a myriad of approaches.

In this chapter I first explain how the process of personnel scheduling can be classified and describe its different stages in detail. Then I lay the foundation for an understanding of workforce scheduling per se. By introducing the element of cyclic scheduling I, in conclusion, circle in on the main topic of my research and thus my thesis.

2.1 The Scheduling Process

One of the first classification methods, proposed by Baker [9], distinguishes three main groups of personnel scheduling problems: shift-scheduling, days-off scheduling and tour-scheduling, where tour-scheduling is a combination of the first two groups. Another classification has been provided by Ernst et al. in [49] that describes the rostering process as a number of different

modules. Many approaches in the literature just try to focus on a specific stage of the scheduling process and others provide solutions for a combination of several stages.

This section gives a short description of the six specific modules as stated in [49].

Stage 1: Demand modeling

In this phase of the rostering process the number of employees needed at certain times over the planning period (length of the schedule) has to be determined. Ernst et al. [49] define three circumstances that affect demand differently:

Task based demand Staff demand may have to fit a list of individual tasks which have a defined starting time and duration. To complete all tasks within the given time frame a certain number of employees is necessary.

Flexible based demand In circumstances where tasks cannot be predicted with certainty, calculations must be based on flexible demand. For example service requests at a help desk arrive at random rates and may have different complexity. Thus, an attempted forecast of how much staff is required at different times throughout the day becomes obligatory.

Shift based demand If work is performed in shifts, staff demand is determined by the specified number of required employees on duty in each shift.

Faced with this multitude of possible requirements, the calculation of staff can be a challenging task. A method to solve *flexible based demand modeling* has been proposed for example by Atlason et al. in [6]. They try to compute the minimal staffing demand for call centers by using an iterative cutting plane method. Dijkstra et al. solve a *task based* demand modeling problem by using a *Decision Support System* (DSS) in [43].

Stage 2: Days-off scheduling

In this module decisions must be made on how day-off periods need to alternate in the schedule, in order to meet several legal restrictions and also to satisfy the employees' health requirements. In the literature this stage is usually discussed in combination with other phases of the scheduling process, but Alfares, for example, focused solely on days-off scheduling in [1], where he presented a two-phase algorithm for cyclic days-off scheduling.

Stage 3: Shift scheduling

At this stage the number of shifts within the planning period and the required number of employees per shift must be determined. This module may be omitted in case of *shift based demand modeling*.

Approaches for shift scheduling have been investigated for many years. Dantzig [35] introduced a set-covering formulation of the problem in 1954. Aykin [8] on the other hand used an inter-programming model to find an optimal solution for "shift scheduling with multiple rest and lunch breaks, and break windows"[8]. Further references to publications in shift scheduling are provided in [48].

Another name for shift scheduling often used in the literature is *shift design*. The term *shift design* was initially mentioned in [72] and [53] to address a slightly different problem than the original *shift scheduling*. This approach's aim was to solve the shift design problem with the help of a tabu search based algorithm. Break scheduling has been considered part of the shift scheduling problem in the past. This approach was fine as long as the number of breaks was rather small in the problem formulations. In nowadays business areas, like airport control or call centers, the scheduling of a large number of breaks has become necessary, however. Additionally, the constraints and restrictions to this scheduling process are more complicated, which is why recently *break scheduling* has been investigated as a stand-alone problem in personnel scheduling. Contributions to break scheduling can be found in [21, 22, 75, 94]

Stage 4: Line of work construction

In this module the actual creation of a specific work schedule for the employees, also defined as a line of work, is accomplished. In case of *shift scheduling* a line of work consists of an assignment of shifts to certain work days in the schedule. Of course this assignment is often restricted by other constraints that may forbid certain shift patterns. A literature overview of this stage is given in [48].

Stage 5 & 6: Task and Staff Assignment

Task assignment In this phase of the rostering process it may be required to assign certain tasks to specific lines of work, because these tasks require special staff skills or a level of seniority.

Staff assignment This module is often combined with the construction of the work schedules, but basically it involves the assignment of individual employees to the lines of work.

Because both of these stages of the scheduling process are beyond the scope of this thesis, the reader is referred to to the annotated bibliography of personnel scheduling by Ernst et al. [48] for further literature on these modules.

2.2 Workforce Scheduling

From now on the term *Workforce Scheduling* will be used for a rostering process that considers shift based demand modeling, days-off scheduling and a line of work that is represented by an assignment of shifts to certain work days in the planning period. Table 2.1 shows a typical work schedule created by the workforce scheduling process. Although different representations of work schedules exist in the literature, the basic concepts of workforce scheduling can be described using this schedule illustration without conflicting with other representations.

Employee	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	-	-	-	A	A	A	A
2	A	N	N	-	-	-	-
3	A	A	A	A	A	A	-
4	-	D	D	D	D	N	N
5	N	-	-	A	A	A	A
6	-	-	-	D	D	D	D
7	N	N	N	-	-	D	D
8	D	A	A	N	N	-	-
9	D	D	D	N	N	N	N

Table 2.1: schedule for one week, 9 employees and three shifts

To gradually illuminate workforce scheduling I first informally describe the main aspects (employee, planning period and shifts) a typical work schedule consists of according to [72]. Then I concentrate on the constraints that restrict the possible solutions for the scheduling problem by restricting shift assignments. In conclusion I introduce the concept of cyclic schedules.

2.2.1 Basic Concepts

A typical work schedule consists of three aspects, employee, planning period and shifts, which are now discussed leaning on the descriptions offered in [72].

Employee The schedule above (Table 2.1) shows the assigned shifts for 9 employees in one week. Here the employees are specified as the numbers $1 - 9$, but in an actual business application the schedule will refer to the employees by name for a more obvious illustration. The number of employees in a schedule is variable and results from the demand modeling phase.

Planning Period The planning period is defined as the length of the schedule. In the literature the planning period is also often referred to as the rostering horizon. Typically it is desired to construct a weekly schedule or sometimes a monthly schedule. Nonetheless, other planning periods may be more suitable for some business areas.

Shifts A shift defines a certain time period during a work day, where the employees have to work. This time period is either determined during the *shift scheduling* phase of the rostering process or the *demand modeling* in case of a *shift based demand*. These time periods have a certain starting time and duration. The starting time often has an influence on the name and therefore the letter of the shift. For example a shift that starts at 22:00 will often be called the night shift and hence it will represented by the letter 'N'. It is also important to note that the letter '-' stands for a day off. In the example schedule (Table 2.1) the different shifts are denoted by the letters 'D' for day shift, 'A' for afternoon shift and 'N' for night shift.

2.2.2 Constraints

I have already mentioned, that in most cases it is necessary to construct work schedules that meet a certain set of constraints. These restrictions can either be based on legal grounds or they may result from the employees' individual preferences that are taken into account. Of course, what constraints are actually considered may vary depending on the area of application. An informal description of the most common constraints in workforce scheduling as stated by Musliu [72] is given below.

Temporal requirements

The temporal requirements, often resulting from the *shift scheduling* module in the scheduling process, specify the number of employees that are required for every shift and each work day of the planning period. In the example schedule (Table 2.1) two staff members per day and night shift are always assigned to each work day. The afternoon shift, however, requires three employees from Wednesday to Friday, whereas on the remaining days of the week two employees are sufficient.

The temporal requirements may be represented by a $m \times w$ matrix, where w stands for the length of the planning period and m for the number of shifts. Below you may see the requirement matrix for the example schedule in Table 2.1:

$$R_{3,7} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 3 & 3 & 3 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix} \tag{2.1}$$

Number of working hours per week

Every employee has a number of obligatory working hours specified in his/her contract with the company. These working hours can differ for every employee. The desired outcome is to assign shifts to employees in such a way that their working hours are met as exactly as possible. A scheduling process that takes this constraint into consideration can be seen as an optimization problem where the optimal solution meets the desired working hours of all employees.

Resting periods

In most countries there are some legal restrictions with respect to the number of consecutive work days an employee may be assigned. To oblige to these restrictions and simultaneously increase employees' productivity and satisfaction, it is necessary to schedule a resting period after a certain amount of work days. The length of this resting period should also be constrained to prevent it from being too long.

In the schedule of Table 2.1 the maximum number of consecutive work days is 7 and the resting period accordingly has to last between two and four days.

Shift constellations

Particular shift sequences need to be prevented to ensure the mandatory resting time between two consecutive work days is being upheld. For example, in many countries it is forbidden to work the day or afternoon shift the day after having worked the night shift, so as to allow for enough regeneration time. Not respecting this resting time would affect employees' concentration badly and may result in a greater risk for woking accidents to happen.

Individual Preferences

For some companies it is also a goal to achieve a schedule that considers the individual preferences of their employees. Possible preferences include vacation times or work day preferences. Some employees may want to have weekends off, while others may not want to work in the night shift or have other preferences for shift assignment.

Hard and soft constraints

In some variations of the workforce scheduling problem it may be possible that some of the constraints are not obligatory but still desired to be fulfilled [72]. These so-called *soft* constraints convert the problem into an optimization problem, where the optimal solution meets all *hard* and *soft* constraints. Often the individual preferences are considered as soft constraints.

2.2.3 Cyclic and non-cyclic schedules

Typical workforce schedules can be divided into two categories, namely *cyclic* (rotating) and *non-cyclic* schedules. Consider the example schedule from Table 2.1. Until now we have specified this schedule as non-cyclic, because it represents a one week schedule for exactly 9 employees. After this week it would be necessary to create a new schedule for the next week or to keep this schedule for several weeks. In contrast, we could also consider the schedule from Table 2.1 as a *cyclic* schedule. In such a cyclic or rotating schedule the employee which is assigned to a specific schedule changes every week. Thereby, in week two of the schedule employee #1 will work the schedule that was worked by employee #2 in the previous week, employee #2 will work the schedule that was worked by employee #3 and so on. To close the cycle the last employee in the list will work the schedule that was worked by employee #1. Thus, cyclicity of the work schedule is achieved. The table below (Table 2.2) illustrates the second week's rotated schedule.

Employee	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	-	-	-	A	A	A	A
2	A	N	N	-	-	-	-
3	A	A	A	A	A	A	-
4	-	D	D	D	D	N	N
5	N	-	-	A	A	A	A
6	-	-	-	D	D	D	D
7	N	N	N	-	-	D	D
8	D	A	A	N	N	-	-
9	D	D	D	N	N	N	N

Table 2.2: Second week of cyclic schedule (9 employees and three shifts)

2.3 Rotating Workforce Scheduling

In this thesis the focus will be on rotating workforce scheduling. Ernst et al. state that cyclic schedules "are most applicable for situations with repeating demand patterns"[49]. Possible examples are train timetables, public transportation in general and industrial plants.

This section defines the specific problem I worked with in my research and review other research that has already been done on this subject.

2.3.1 Problem Definition

The basis for all investigations in my thesis is the rotating workforce scheduling problem as defined in [72], where Musliu describes a problem more specific than the general workforce scheduling problem.

The formal definition of the problem and its instances is based on the definition from [72], [74] and [73] and given below:

Instance

- n: Number of employees.
- A: Set of *m* shifts (activities): $a_0, a_1, ..., a_{m-1}$, where a_0 represents the special "day-off" shift.
- w: Length of the schedule. Typically the value of w is set to 7 (one week). The total length of a planning period is $n \times w$ for one employee, because of the cyclicity of the schedule.
- *Cyclicity*: The required schedule is considered to be cyclic, that is each employee works the schedule for the complete planning period starting at a specific week. For example, employee 1 starts at the first week of the schedule, whereas employee 3 starts with the shift plan for the third week. See 2.2.3 for a more detailed explanation of cyclic schedules.
- R: Temporal requirements matrix, an $(m-1) \times w$-matrix (because the day-off shift a_0 is excluded) where each element $r_{i,j}$ of matrix R defines the required number of employees for the corresponding shift (i) on that day of the week (j).
- *Prohibited shift sequences:* Certain shift sequences must be prevented from being assigned to the employees. For example in some countries it is forbidden to work the day shift the day after working the night shift, since the employees would not have enough resting time between work.
- MIN_w and MAX_w: Minimal and maximal possible length of work blocks. A work block consists of consecutive days on which an employee is working without having a day-off.
- "Maximum and minimum length of periods of consecutive shifts: Vectors $MAXS_m$, $MINS_m$, where each element shows the maximum respectively minimum permitted length of periods of consecutive shifts. Because the schedule is cyclic these constraints for the length of blocks of shift are identical for all employees and the shift sequences can lie in more than one week" [73].

Problem

"Find a cyclic schedule (assignment of shifts to employees) that satisfies the requirement matrix, and all other constraints"[73].

Note that in the original definition from [72] it is required to find as many non-isomorphic cyclic schedules that meet all constraints and also optimize the number of free weekends as possible. In this thesis, however, I concentrate on finding only one solution, i.e. one cyclic schedule that fulfills all hard constraints defined above.

2.3.2 Previous Work

Rotating workforce scheduling has been investigated by many researchers in computer science. Accordingly, many different approaches have been proposed for solving this problem.

Early approaches to solve the problem were based on exhaustive enumeration [58],[29]. As it is stated in [10] these algorithms were not very sophisticated and unable to solve large-scale problems. In [29] and [58] specific problems were investigated that both focus on manpower scheduling in police departments. These two problems are still often used in the literature for comparison with other approaches.

In 1980 Laporte et al. proposed an algorithm based on Integer Linear Programming (ILP) for the construction of rotating schedules [68]. They also concentrated on a specific problem for manpower scheduling of police stations.

Balakrishnan and Wong [10] developed a network flow model for rotating workforce scheduling, where a solution is constructed by finding an optimal path in the network.

Laporte argues in [66] that finding the optimal schedule is more of an art than an exact science. Further he suggests that sometimes the constraints have to be relaxed in order to design a good schedule.

Musliu et al. [74] split up the scheduling process in its different phases to obtain a smaller search space for each stage. The main aspect of this method is, its interactive design, whereby a human decision-maker is involved in each step of the scheduling process. Additionally, intelligent backtracking algorithms are applied in each step of the scheduling process, which allows an efficient generation of good solutions. This approach has been used in the commercial application *First Class Scheduler* (FCS) [53], [74].

In [73], a new algorithm based on tabu-search and a min-conflicts heuristic was developed. Although this technique has proven to perform well, it is not a decision procedure for the problem, meaning there is no guarantee that the algorithm always finds a solution. Nevertheless, it provides good results for almost all practical instances of the problem.

In this same paper [73] Musliu also introduces 20 instances of the rotating workforce problem.[1]. While the first 3 instances are slightly adapted from the problems in [29], [58] and [68], the other

[1] Instances available from *http://www.dbai.tuwien.ac.at/staff/musliu/benchmarks/*

17 instances were collected from real-life problems of different business areas. These instances were also used in this thesis for testing and comparison with previous solution methods.

Another promising possibility to deal with rotating workforce scheduling is constraint logic programming, which was first used by Chan and Weil in [31], however, the problem definition differs from the one I use in this thesis. Further research on this topic was conducted by Laporte and Pesant in [67] who developed a constraint programming algorithm for a more similar problem to the one considered for this work. Finally, Triska and Musliu proposed a constraint programming method implemented with the logic programming language *Prolog* in [90].

Beside these approaches to workforce scheduling there have also been several other publications considering related problems in personnel scheduling like nurse rostering [28] or crew scheduling. For further information the reader is referred to [2], [49] and [91] which provide surveys on personnel scheduling and a good literature overview of this research area.

CHAPTER 3

Satisfiability Modulo Theories

> "Satisfiability is one of the most fundamental problems in theoretical computer science, namely the problem of determining whether a formula expressing a constraint has a solution." [39]

Another term for this problem often used in the literature is *Constraint Satisfaction Problem* (CSP). CSPs arise in many different areas of computer science, but also in real-life situations. For instance, the popular logic based puzzle *Sudoku* can easily be formulated as a constraint satisfaction problem.

One of the most investigated CSPs is *propositional satisfiability* (SAT), which is about determining whether a formula of boolean variables and logical connectives can evaluate to *true* by finding an appropriate assignment for the variables. SAT is the first problem known to be NP-complete, meaning that SAT is a decision problem for which no known algorithm can efficiently find a solution for *any* of its instances. However, there has been a lot of research attention on developing algorithms that find solutions for *many* instances of SAT (e.g. [71],[47]). Tools that efficiently implement such algorithms are called SAT-solvers. Due to improvements in solving techniques and technological advances in general, SAT-solvers have gained a lot of practical use and are now applied in diverse areas of computer science.

Recently researchers have tried to adopt the solving techniques for propositional satisfiability to formulas with more expressive power than propositional logic such as first-order logic. In first-order logic formulas are constructed of variables and logical connectives, but also of quantifiers, functions and predicates. A formula is satisfiable if an assignment for the variables and also an interpretation of the different function and predicate symbols can be found such that the formula evaluates to *true*. In *Satisfiability Modulo Theories* first-order formulas are restricted by certain background theories, which already define the interpretation of some function or predicate symbols.

Within the following sections first the terminology of propositional and first-order logic that is necessary to follow the rest of this chapter is briefly explained. Next, an overview of SAT-solving and a short description of the DPLL algorithm is provided. By introducing the concepts of background theories, I move on to the definition of Satisfiability Modulo Theories, focusing on techniques for solving SMT problems. Afterwards, I provide an overview of the application fields of modern SMT-solvers and conclude with presenting the SMT-LIB standard [18] which is used for modeling the rotating workforce problem throughout this thesis.

3.1 Concepts and Terminology

3.1.1 Propositional Logic

The smallest formula in propositional logic is a propositional variable p or it's negation $\neg p$. To build more complex formulas one can use logical connectives to combine multiple propositional variables and/or sub-formulas. The following logical connectives exist in propositional logic:

$$\begin{array}{ll} \varphi_0 \wedge \varphi_1 & \text{(conjunction)} \\ \varphi_0 \vee \varphi_1 & \text{(disjunction)} \\ \varphi_0 \Rightarrow \varphi_1 & \text{(implication)} \\ \varphi_0 \Leftrightarrow \varphi_1 & \text{(equivalence)} \end{array} \tag{3.1}$$

As mentioned before, the goal of SAT is to find a truth assignment M for a formula φ that assigns either *true* or *false* to every propositional variable in φ so that the formula evaluates to *true* under the intended interpretation of the logical connectives. As a result we can say M *satisfies* φ ($M \models \varphi$). Such truth assignments are also referred to as *models* for φ.

In general, a formula φ is considered to be *satisfiable* iff a truth assignment M exists, where $M \models \varphi$. φ is called a *valid* formula if $M \models \varphi$ holds for all possible M.

A propositional variable p (or its negation $\neg p$) is called a *literal*. A set of literals connected via disjunction is referred to as a *clause* $l_1 \vee l_2 \vee \ldots \vee l_n$. Clauses are used to build the *conjunctive normal form* (CNF) of a formula. It defines a structure of a formula that only consists of a conjunction of clauses $C_1 \wedge C_2 \wedge C_3 \wedge \ldots \wedge C_n$. CNF is often used for SAT-solving as it reduces the number of logical connectives to only conjunction and disjunction. Every propositional formula can be translated into CNF.

3.1.2 First-Order Logic

In addition to propositional logic, first-order logic also provides *quantification*, *predicate symbols* and *function symbols*. These symbols are also called *non-logical symbols*.

For a formal definition of first-order logic, one has to define the set of non-logical symbols. In the general case this set is infinite and called a *signature*.

Definition 1 (Signature)**.** A signature Σ can be defined as a tuple $\Sigma = (\mathcal{F}, \mathcal{P})$. Every non-logical symbol (either function or predicate symbol) has a certain *arity* assigned to it. The arity defines the number of arguments a function or predicate accepts.

- $\mathcal{F}$ is the set of functional symbols. Functions map a certain number of arguments to a specific value. It is not necessarily important to define the type a function returns. A function for which only its name and arity are defined, is called an *uninterpreted function*. Functions with arity $n = 0$ are also called *constants*.
- $\mathcal{P}$ represents the set of predicate symbols. A predicate can have a number of arguments and maps them to a boolean value, i.e. *true* or *false*. A predicate with arity $n = 0$ can be seen as a propositional variable.

In first-order logic the set of logical symbols is extended by two quantifier symbols $\forall$ and $\exists$, where $\forall$ stands for the *universal* quantifier and $\exists$ represents the *existential* quantifier. A variable that is not bound by a quantifier is called a *free* variable. Additionally, in first-order logic the predicate symbol "=" is often used to state equality. It is most commonly interpreted with its intended meaning, that is, if two elements of a domain are compared with the symbol "=", then the predicate evaluates to true if those elements are actually the same.

Below, some important terms in the area of first-order logic are defined:

Definition 2 (Term)**.** A term in first-order logic can consist of the following expressions:

- $x, y, z, \ldots$: A variable is a term
- $a, b, c, \ldots \in \mathcal{F}$: A constant symbol (function of arity 0) is a term
- $f(t_1, t_2, \ldots, t_n)$ A function $f \in \mathcal{F}$ is a term if $t_1, \ldots t_n$ are terms.

Definition 3 (Atom)**.** An atom is defined as follows:

- $t_1 = t_2$ is an atom if t_1 and t_2 are terms of the same sort.
- $p(t_1, \ldots, t_n)$ is an atom if $p \in \mathcal{P}$ and $t_1, \ldots, t_n$ are terms.

3.2 SAT Solving

Although SAT is an NP-complete problem and no known algorithm for efficiently solving *any* instance of SAT exists (if $P \neq NP$), scientists have developed procedures that provide acceptable performance when applied to practical problems. "Most successful SAT solvers are based on an approach called *systematic search*. The search space is a tree with each vertex representing a propositional variable and the out edges representing the two choices (i.e., *true* and *false*) for

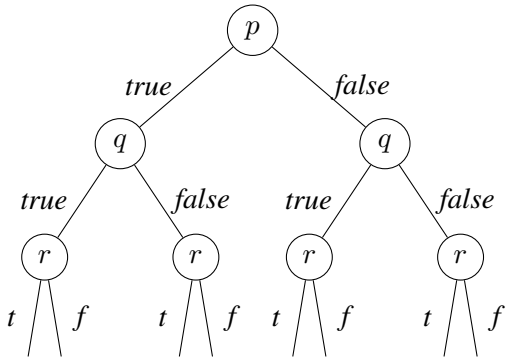

Figure 3.1: possible decision tree for φ

this variable." [39] Consider for example the formula $\varphi : p \vee (\neg q \wedge r)$. A possible decision tree for this formula may look like the one in Figure 3.1.

The tree contains every possible truth assignment for the formula φ. A SAT solver that uses the systematic search approach looks for a truth assignment M that satisfies φ, i.e. it tries to find a path in the tree so that the formula evaluates to *true*. The basic algorithm for the systematic search approach is based on a procedure developed by Davis and Putnam in [36] and further improved by Davis et al. in 1962 [37] and is known today as the DPLL (Davis-Putnam-Logemann-Loveland) procedure.

DPLL

DPLL consists of three phases – decide, propagate and backtrack – and operates on CNF formulas. The procedure starts by heuristically selecting an unassigned variable from the formula (*decide*) and setting it to *true* or *false*. Using propagation rules, simplifications possible under this partial truth assignment are carried out. One commonly used propagation rule considers *unit-clauses*, which are clauses containing only negative literals but for one exception. Thus, the unassigned literal has to be set to *true* in order to ensure that the whole clause evaluates to *true*. The first two phases (decide and propagate) are repeated until either the formula evaluates to *true*, i.e. the formula is *satisfiable*, or a *conflict* occurs. A *conflict* is reached when due to the current partial truth assignment, a clause C_i of the formula contains only literals that evaluate to *false*. At this point the procedure has to backtrack, because one of the earlier decisions (assignments) could have been wrong. If the algorithm can't find a decision to backtrack, the formula is *unsatisfiable*.

Over the years researchers have suggested many adjustments to the DPLL procedure to improve its performance. Many of these adjustments focus on the backtracking part of the algorithm and suggest techniques that increase the efficiency of backtracking, e.g. *lemma learning* [69] or *non-chronological backtracking* [69]. Other improvements try to decrease the number of clauses by *preprocessing* the formula before starting the DPLL procedure [39].

3.3 Theories

According to [81], "a theory T is a set of closed first-order formulas that is satisfiable in the first-order sense". To prove that a formula F is satisfiable with respect to a background theory T, one has to prove that $F \wedge T$ is satisfiable. If this is the case we call F *T-consistent* or *T-satisfiable*. Furthermore, a truth assignment M is referred to as a *T-model* if $M \models F$ and $F \wedge T$ are satisfiable. Satisfiability Modulo Theories describes the problem of finding a T-model for a given formula F so that F is T-satisfiable. Note that the number of background theories is not restricted, hence in SMT it has to be verified whether a formula F has a T-model with respect to multiple theories.

In the following sections, I provide a summary of some theories that often occur in SMT problems and are supported by most SMT-solvers.

3.3.1 Equality

The theory of equality $\mathcal{T}_{\mathcal{E}}$ is the most basic theory because it does not restrict any values of variables and thus includes all possible models [17]. It has an empty set of first-order formulas and therefore is sometimes referred to as the theory of free-functions or simply the *free theory*. Another term also often used for $\mathcal{T}_{\mathcal{E}}$ is theory of equalities and uninterpreted functions (EUF). Although it does not contain any restrictions, it is a particularly important theory, because many other theories can be reduced to $\mathcal{T}_{\mathcal{E}}$.

A decision procedure for $\mathcal{T}_{\mathcal{E}}$ is an algorithm that is based on *congruence closure*. For detailed information on such an algorithm refer to [78] and [44].

3.3.2 Linear Arithmetic

Linear Arithmetic represents the theory of arithmetical operations that use the two functions $+,-$. These functions can be used to form expressions over variables or numerical constants. Additionally, it is also allowed to use multiplication of variables with constants, i.e. $8 \cdot y$. The formula $2 + x = 0 \wedge y - 3 < 5$ is a legal formula in the theory of linear arithmetic and shows how relations between applications of arithmetical functions can be formed using the predicates "$=, <$". Formulas in linear arithmetic can be checked for satisfiability using for instance the *dual simplex algorithm* [46].

3.3.3 Bitvectors

Bitvector theory focuses on the fact that computers solve arithmetical operations in a way very different from normal mathematical arithmetic. Computers work with the binary numeral system and represent integer numbers as a fixed size sequence of 0s and 1s (bits). Due to this different representation, they also use different operations on the bit-wise level of such sequences (vec-

tors). For instance, consider the formula $(x - 1)$ & $x = 0$, where x is an integer number. This formula can be used to check whether x is a power of 2. It uses the bitwise *AND* operation (&) over the binary representations of x and $x - 1$.

Decision procedures for bitvectors are often based on a reduction to Boolean satisfiability using a technique called *bit-blasting* that converts bitvectors to boolean formulas. Since bit-blasting can result in very large propositional formulas researchers have recently investigated different strategies to improve these techniques for fixed-size bitvectors. In [26] a decision procedure based on the lazy approach of SMT was proposed that tries to minimize bit-blasting and instead uses bitvector reasoning as much as possible. Another procedure based on abstraction was developed in [27]. It uses under- and over-approximation of bitvector formulas to optimize the number of boolean variables necessary to encode bitvectors.

3.3.4 Arrays

Since it was introduced by McCarthy in [70], the theory of arrays ($\mathcal{T}_\mathcal{A}$) has become integral for many areas in computer science, especially software verification. $\mathcal{T}_\mathcal{A}$ can basically be divided into two subgroups, the *extensional* and *non-extensional* theory of arrays. The first one establishes the equality relation over arrays. Two arrays are equal iff for every index i, the value at index i in both arrays is the same. Non-extensional theories of arrays do not consider this relation. $\mathcal{T}_\mathcal{A}$ defines two basic function symbols *read* and *write*:

- $read(A, x)$: where A is an array and x represents a certain position (index) in the array. The function returns the value of A at position x.
- $write(A, x, v)$: A and x are the same as above and v is the value to be stored in A at position x. The function returns a modified array A', with the new value stored in it.

Furthermore, it defines some basic axioms via these two functions.

$$\begin{array}{l} x = y \rightarrow read(write(A, x, v), y) = v \\ x \neq y \rightarrow read(write(A, x, v), y) = read(A, y) \end{array} \quad \text{(read-over-write)} \tag{3.2}$$

The first decision procedure for the extensional theory of arrays has been proposed in [85]. Ghilardi et al. provided additional procedures for some extensions to this theory in [55].

3.4 SMT Solving

In the previous section some theories that are very common in SMT problems were mentioned. Now the main question is how such formulas can be efficiently solved. In this context "solving" means proving that the formula is T-satisfiable. In the last few years many techniques have been developed to solve SMT problems and most of them can be categorized into two classes,

namely the *eager* and the *lazy* approach. In the following I give a short overview of how these techniques work and discuss their advantages.

3.4.1 The Eager Approach

This approach is based on a translation of the input formula into an *equi-satisfiable* propositional CNF formula. This formula is then passed to a SAT solver which checks it for satisfiability. The most successful results using the eager approach have been achieved with formulas in $\mathcal{T}_{\mathcal{E}}$ as stated in [84]. Techniques for translation into propositional logic are *Small Domain Instantiation* [82] and *Per-Constraint Encoding* [83]. An advantage of the eager approach is that its effectiveness mostly depends on the used SAT-solver and one can always use the currently best performing one. The main drawback of these methods, however, is that to support different theories one would need efficient translation procedures for each theory. Moreover, eager techniques often fail to find a solution for practical problems, either because they run out of time or exceed the available memory, as shown by De Moura and Rueß in [41]. Nevertheless, the eager approach is still used in current state-of-the-art SMT-solvers like *Beaver* [61] or *Boolector* [25].

3.4.2 The Lazy Approach

In contrast to the eager approach, where the background theory is only used for the translation process into propositional logic, the *lazy* approach uses specific theory solvers (T-solvers as they are called in [80]) that provide specialized decision procedures for determining if a conjunction of theory-literals is satisfiable (T-consistent) or not [80]. Now one could be tempted to solve SMT problems by merely using these T-solvers and simply translating any given formula F into a conjunction of literals (DNF[1]). Unfortunately, this conversion is very inefficient as it may result in an exponential blow up of the formula [80]. Therefore, the lazy approach does not only feature T-solvers but combines them with a SAT-solver. Successful applications of this approach are described in [4, 5, 7, 12, 15, 42, 50]. A more detailed description of the lazy approach is given below.

In a first step every theory atom of a formula F is replaced by a (fresh) propositional symbol, meaning the theory T is *forgotten* or simply ignored. Now this simplified formula is passed to the SAT-solver, which tries to find a truth assignment that satisfies the formula. If the solver cannot find any satisfying assignment, the formula is unsatisfiable and hence the original formula is also T-unsatisfiable. If the solver returns *sat*[2] and provides a successful interpretation for the propositional variables, the variables are substituted by the corresponding theory atoms and this T-model candidate, seen as a conjunction of literals, is passed on to the T-solver. If the T-solver determines that the model candidate is T-consistent, a true model for the formula F has been found. Otherwise the T-solver provides an *explanation* for the inconsistency in form of a clause

[1] Disjunctive Normal Form

[2] short for satisfiable. This abbreviation is often used by SAT/SMT tools.

and adds it to F. Afterwards the SAT-solver is started again. This process repeats until either a true model is found or the SAT-solver returns *unsat*.

The lazy approach provides more flexibility, since the SAT-solver and also the T-solver are completely interchangeable. i.e. using a different SAT-solver and choosing the right T-solver for the according background theory. Nieuwenhuis et al. discuss some refinements to the SMT process in case the used SAT-solver is based on DPLL. These improvements are summarized in the following according to [80].

Improvements

Incremental T-solver By using an incremental T-solver the partial assignment can be checked for T-consistency during the DPLL process, i.e. possible inconsistencies can be discovered immediately following their generation. The T-solver receives the new literal from the SAT-solver and checks if the previously assigned literals plus the new one are still T-consistent. Incremental T-solvers were already proposed by Audemard et al. in [7], where the concept was called *early pruning*. In 2002, Barrett suggested a similar approach under the name *eager notification* [20]. Today most of the state-of-the-art SMT-solvers use incremental T-solvers to perform better.

Online SAT-solver In the basic procedure the SAT-solver has to start fresh, if the T-solver detects a T-inconsistency. This can be improved by implementing an interface for the DPLL procedure's backtracking step, so that the incremental T-solver can instruct the SAT-solver to backtrack after discovering that the assignment is not T-consistent. Current SAT-solvers like the popular *MiniSAT* or *zChaff* support this feature and it has become very common in today's SMT implementations.

Theory Propagation All the improvements we have discussed so far only use the T-solver as a kind of validator that verifies whether a certain propositional assignment is T-consistent or not. *Theory propagation* describes a method where the T-solver is not only used for validation, but also to assist the DPLL procedure in finding a propositional model. This can be achieved by allowing the T-solver to detect literals that occur in a formula F and are currently not part of the assignment M, but are a logical consequence of M with respect to the background theory. These literals can be added to M and thereby speed up the DPLL search.

Theory propagation is also known as *Forward Checking Simplification* and was first introduced by Armando et al. in [4]. This concept had only been integrated in very few SMT implementations, however, until Ganzinger et al. introduced the now widely used DPLL(T) [52] architecture (discussed below), in which theory propagation plays an integral role.

Exhaustive Theory Propagation Exhaustive theory propagation means that the T-solver propagates *all* possible literals that can be deduced from the current partial assignment M. This

offers the advantage of an always T-consistent assignment and thus it becomes unnecessary to validate the model after a literal has been added. Nonetheless, exhaustive theory propagation is not always a suitable choice for every theory: As discussed in [80] it should not be used for formulas of Equality logic and Uninterpreted functions ($\mathcal{T}_{\mathcal{E}}$).

DPLL(T)

A framework that incorporates the improvements listed above is the DPLL(T) framework by Ganzinger et al. which was first introduced in [52]. This DPLL(T) framework is now the basic architecture for almost all modern SMT-solvers (e.g. *Z3* [38], *Yices* [45], *CVC* [19], *MathSAT* [33], etc.). For a detailed description of the framework refer to [52] and [80].

3.5 Combining Decision Procedures

So far we have only discussed implementations of SMT-solvers that can handle formulas of one specific background theory T. However, in many practical problems it is beneficial to use different theories in one formula to have greater expressive power.

3.5.1 Basic Idea

The approach for considering multiple theories builds on the architecture of the lazy approach, where a modern (DPLL-based) SAT-solver is used in combination with a specialized T-solver. This architecture has to be refined, so that it supports multiple T-solvers interacting with each other and the SAT-solver. Each of these T-solvers implements a decision procedure for one of the background theories of the input formula. The idea is based on the so-called *Nelson-Oppen procedure* [77] developed by Greg Nelson and Derek C. Oppen in 1979.

3.5.2 Nelson-Oppen Procedure

The Nelson-Oppen procedure provides a method for proving satisfiability of formulas that use *pairwise disjoint theories*. Two theories T_1 and T_2 are disjoint, if their signatures Σ_1 and Σ_2 do not share any predicate or function symbols.

The first part of the procedure is separating the formula into theory specific partitions. Each partition can then be passed on to the corresponding T-solver. In [64] this step is called *purification*.

Purification

To *purify* a formula in the sense of the Nelson-Oppen procedure, every *pure* subterm t – a *pure* term only contains non-logical symbols of one theory – is replaced by a fresh proxy variable (e.g. x). Thus, a *definitional equality* $x = t$ is added to the set of literals. From now on we will use the equality predicate as a construct that is understood by all specific T-solvers. Hence, we will consider formulas in *first-order logic with equalities*. Consider for instance the following example from [64]:

$$1 + f(x) = f(1 + f(x)) \tag{3.3}$$

This formula has two background theories, namely the theory of uninterpreted functions ($\mathcal{UF}$) and linear arithmetic ($\mathcal{LA}$). By purification we introduce new proxy variables and end up with the following set of literals:

$$\{y = f(x), z = 1 + y, u = f(z), z = u\} \tag{3.4}$$

This set of literals is *equisatisfiable* to the original formula F and can be split into two subsets: $F_{\mathcal{UF}}$, which only contains $\mathcal{UF}$-literals, and $F_{\mathcal{LA}}$, which only consists of $\mathcal{LA}$-literals. As a consequence, we can formally define that F is satisfiable iff $F_{\mathcal{UF}} \wedge F_{\mathcal{LA}}$ is satisfiable.

Equality Propagation

After purification is complete and *definitional equalities* have been added, every decision procedure for a theory T_i (T-solver) tries to deduce equalities that are logical consequences of conjunction of literals of each subset F_i. These equality literals are then shared with the other T-solvers. Consider the following example from [76]:

$$f(f(x) - f(y)) \neq f(z) \wedge x \leq y \wedge y + z \leq x \wedge 0 \leq z \tag{3.5}$$

The background theories of this formula are again $\mathcal{LA}$ and $\mathcal{UF}$. After purification the two subsets of literals are as follows:

$$\begin{array}{c|c} T_{\mathcal{LA}} & T_{\mathcal{UF}} \\ \hline x \leq y & f(c_1) \neq f(z) \\ y + z \leq x & f(x) = c_2 \\ 0 \leq z & f(y) = c_3 \\ c_1 = c_2 - c_3 & \end{array} \tag{3.6}$$

These initial sets of literals are now passed on to the T-solvers of the underlying theories. The solver of $T_{\mathcal{UF}}$ cannot deduce any equalities from its literals, whereas the $T_{\mathcal{LA}}$-solver infers that

$z = 0$, because $x - y \leq 0$ (from the first literal) and $z \leq x - y$ (from the second literal) and $0 \leq z$ (from the third literal). Furthermore, the solver deduces $x = y$ due to $z = 0$ and $z \leq x - y$.

$T_{\mathcal{LA}}$	$T_{\mathcal{UF}}$
$x \leq y$	$f(c_1) \neq f(z)$
$y + z \leq x$	$f(x) = c_2$
$0 \leq z$	$f(y) = c_3$
$c_1 = c_2 - c_3$	$\boldsymbol{x = y}$
$\boldsymbol{z = 0}$	
$\boldsymbol{x = y}$	

(3.7)

Note that the inferred equation $z = 0$ was not propagated to $T_{\mathcal{UF}}$, because it is not an equation over variables. It contains 0 and 0 is not defined in the theory of uninterpreted functions.

Next, the $T_{\mathcal{UF}}$-solver will use the propagated equation $x = y$ to deduce that $c_2 = c_3$ and will propagate this to $T_{\mathcal{LA}}$. The solver of $T_{\mathcal{LA}}$ will detect that $c_1 = 0$ and moreover that $c1 = z$. After $c_1 = z$ has been propagated to $T_{\mathcal{UF}}$, it will find a T-inconsistency, because $f(c_1) \neq f(z)$, but $c_1 = z$. Hence the formula is *unsatisfiable*.

$T_{\mathcal{LA}}$	$T_{\mathcal{UF}}$
$x \leq y$	$f(c_1) \neq f(z)$
$y + z \leq x$	$f(x) = c_2$
$0 \leq z$	$f(y) = c_3$
$c_1 = c_2 - c_3$	$\boldsymbol{x = y}$
$\boldsymbol{z = 0}$	$\boldsymbol{c_2 = c_3}$
$\boldsymbol{x = y}$	$\boldsymbol{c_1 = z}$
$\boldsymbol{c_2 = c_3}$	**unsatisfiable**
$\boldsymbol{c_1 = 0}$	
$\boldsymbol{c_1 = z}$	

(3.8)

Over the years, several variations and improvements of the Nelson-Oppen procedure have been developed, but most modern SMT implementations are supporting multiple theories by sharing and propagating equalities between the T-solvers. One of the first architectures for combined theories was introduced in [16] as an enhancement to the *DPLL(T)* [52] architecture. Krstić and Goel tried to define a more detailed and transparent system which they called *NODPLL* [64]. The *Yices* SMT-solver is a great example of a modern SMT-solver that is based on a DPLL SAT-solver and a generalization of the Nelson-Oppen procedure that allows an exchange of so-called *offset equalities* [45].

3.6 Modern SMT Architecture

As we have already discussed several algorithms that are used for solving SMT formulas, it is time to get a better look at the big picture of a modern SMT-solver's architecture. Most (DPLL(T)-based) SMT-solvers share a lot of similarities in their architectural design. The integral parts are a SAT-solver, which uses the DPLL algorithm, a set of decision procedures for the different background theories and often a core-component to coordinate the communication between the T-solvers and the SAT-solver. Figure 3.2 illustrates this basic architecture used for the Yices SMT-solver.

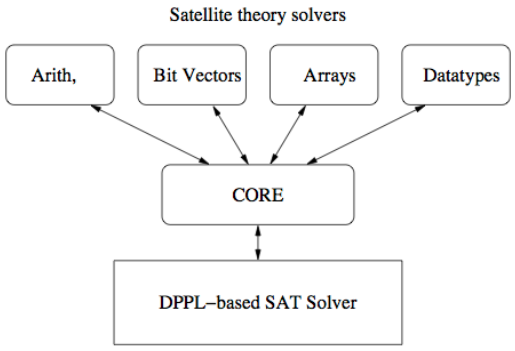

Figure 3.2: Architecture of the Yices SMT-solver [45]

The higher the number of included features, the more complex the architectures of SMT-solvers get. Figure 3.3 gives a schematic overview of Z3 [38], illustrating the design of an SMT-solver that provides additional features like quantifier instantiation or different heuristics to preprocess and simplify formulas before passing them to the SAT-solver and theory solvers. Z3 supports several theories, including linear arithmetic, bitvectors and arrays. For each of these theories a specialized T-solver also called *satellite solver* exists. These satellite solvers communicate with the *congruence closure core* that is responsible for managing the current literal assignments and theory propagations, but also coordinates the equality propagations between the T-solvers. Additionally, Z3 supports different input languages including an API for the *C* programming language and several *.NET* languages. For a more detailed description of the modules of Z3 the reader is referred to [38].

3.7 Areas of Application

SMT-solvers have been successfully applied to a number of different areas in computer science. This section first provides some fields of application in Software Engineering according to [23] and afterwards gives an overview of how SMT-solvers have been used for problem solving.

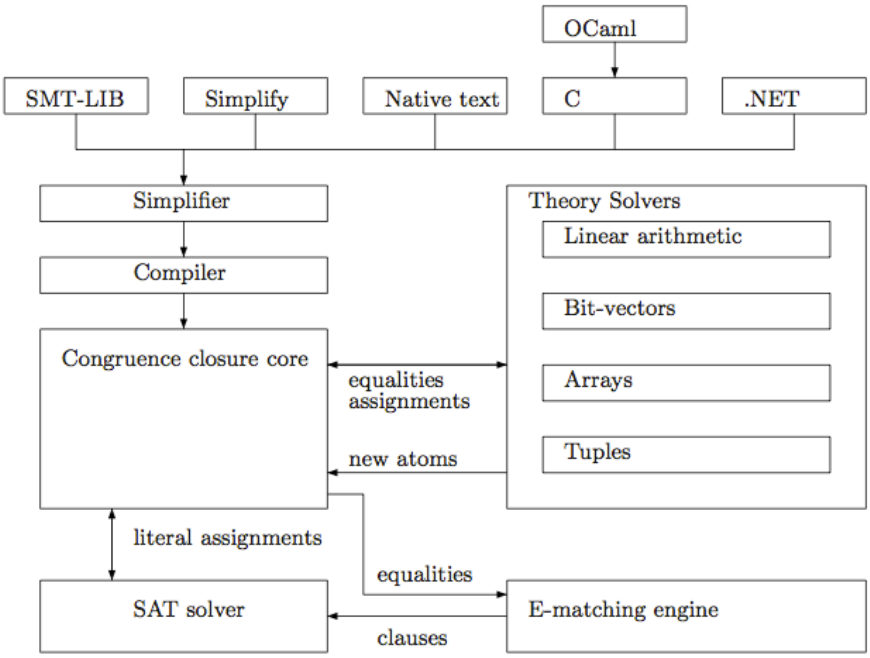

Figure 3.3: Z3 System Architecture [38]

3.7.1 Test Generation

SMT-solvers have gained interest in the field of test-case generation, especially a technique called *dynamic symbolic execution* [56], which is an extension to *static symbolic execution* [63]. Here, the idea is to create a symbolic constraint by negating a *branch condition* of a concrete execution trace to get a new unexplored execution path. This constraint can be formulated using first-order theories that are supported by current SMT-solvers. A model for the constraint is the input of a new test-case that can be used to increase branch coverage. A number of tools have been developed to help a software engineer to automatically create test-cases, for instance SAGE, Pex and Yogi from Microsoft [56].

3.7.2 Program Model Checking

While automatic test-case generation is of great use in software testing, it does not guarantee that all errors are found. Therefore, *Program Model Checking* was developed, which allows to check programs for specific groups of errors by exploring "all possible executions using a finite and sufficiently small abstraction of the program state space."[40] One approach is to find an abstraction for every single statement in the program. SMT-solvers can be used to create such abstractions by computing relations between program variables. This reduces the number of program states since the original statements can be substituted by statements over boolean

variables that express these relations. A finite state model checker can then be used on the abstracted program to check for errors. Tools that are used for program model checking are for example *BLAST*[59] or *SLAM/SDV* [11].

3.7.3 Software Verification

A more general approach for proving program correctness is *verification*. The idea of verification dates back to the 1960s to Robert Floyd [51] and Tony Hoare [60]. They tried to find ways of determining whether a program does what it is intended to do. By inserting logical assertions (*code contracts*) into the source code of a program, a verification condition generator tool should be able to convert these assertions into logical formulas. These formulas are then passed on to an SMT-solver for proving.

Software verification still poses a challenge to current SMT-solvers, because it is often necessary to translate assertions over certain program properties into quantified logical formulas.

Usually SMT-solvers only consider quantifier-free first-order formulas, since quantified formulas are not decidable in general. Nevertheless, a lot of research has been conducted to support quantifiers in SMT-solving. The current main approach to overcome this issue is *quantifier instantiation*. An overview of how quantifier instantiation can be implemented is given in [54].

3.7.4 Problem Solving with SMT

Many real world problems of different fields of application like scheduling, planning, logistics, etc. can be represented as constraint satisfaction problems. A common approach for solving such CSPs is *Constraint Programming* (CP). Another method to solve CSPs is by translating it into propositional logic and solving it with state-of-the-art SAT-solvers. Research has proven this technique to be a powerful alternative to CP for a variety of problems [30, 88, 92]. “Today it is becoming clearer that SAT and CP techniques share many technological similarities and applications (see the CP 2006 Workshop on the Integration of SAT and CP techniques).” [81] While SAT-solvers have proven to be very efficient and robust, CP-techniques allow a more elegant and expressive way to formulate such problems.

As a consequence, researchers have gained interest in using SMT for solving constraint satisfaction problems, since it combines the efficiency of modern SAT-solvers with the expressive power of theories in first-order logic.

Bofill et al. stated in [24] that SMT-Solvers are well suited for solving CSPs and presented a tool for translating instances of the *MiniZinc* constraint modeling language into the *SMT-LIB* standard. Ansótegui et al. solved the *Resource-Constrained Project Scheduling Problem* with SMT-solvers [3] and Nieuwenhuis and Oliveras showed that SMT-solvers can even be used to solve constraint optimization problems [79].

3.8 SMT-LIB

SMT-LIB is an international initiative whose main goal is to facilitate research and development in SMT. SMT-LIB provides a lot of benchmarks for different first-order theories that are used in annual competitions between different SMT implementations. These competitions are important for evaluating modern SMT-solvers and advancing the state-of-the-art in this area.

Another major task of the SMT-LIB initiative is to provide a *standard* input language for SMT-solvers. The current SMT-LIB standard (version 2.0) allows the description of formulas by defining function symbols and assertions. I present a short overview of the SMT-LIB standard's main parts, but refer the reader to [18] for a detailed specification of the syntax and semantics of the SMT-LIB format.

3.8.1 Sorts

SMT-LIB uses an adapted form of *many-sorted* first-order logic with equalities as its underlying logic. In many-sorted logic every non-logical symbol, i.e. variables, functions and predicate-symbols, can be of a certain type (*sort*). These sorts are similar to data types in modern (strongly-typed) programming languages. Sorts that are supported by SMT-LIB are for example:

- Int (Integer numbers)
- Real (Real numbers)
- Array
- BitVec (Bitvector)
- List
- ...

3.8.2 Theories

SMT-LIB provides a catalog of predefined theories. Each of these theories is defined as a signature that consists of a number of sorts and sorted function symbols. The theories in the SMT-LIB catalog do not quite correspond to the theories explained in section 3.3. In fact they are defined in a more general sense. For example the theory of *Ints* in the SMT-LIB standard defines all arithmetical operations over the integers including multiplication, division and modulo. The SMT-LIB theory catalog provides the following theories:

- **Core:** defines the basic boolean operators, the equality function and other useful functions like if-then-else

- **Ints:** the integer numbers and their according arithmetical functions
- **Reals:** the real numbers and their arithmetical functions
- **Reals_Ints:** Real and integer numbers
- **Fixed_Size_BitVectors:** bitvectors of arbitrary length
- **ArraysEx:** extensional theory of arrays

3.8.3 Logics

A *logic* in the sense of the SMT-LIB standard consists of possibly multiple theories that can be restricted to a subset of their signature. Consider the example from before, where the theory of *Ints* allows the application of non-linear arithmetical operations. The SMT-LIB standard defines a logic called *QF_LIA* that restricts the theory of *Ints* in a way that only allows the linear arithmetical functions $+$ and $-$ to be applied and furthermore forbids the use of any quantifiers. Thus, a logic in the SMT-LIB standard is more closely related to the theories explained in Section 3.3 than the theory definitions of SMT-LIB are.

CHAPTER 4

Implementation

In this chapter I present the two modeling approaches I chose to formulate rotating workforce scheduling as an SMT problem. I explain in detail how the necessary constraints can be expressed in first-order logic and moreover how they are formulated in the standardized input format (*SMT-LIB*). These encodings in SMT-LIB are then passed to state-of-the-art SMT-solvers to find a schedule that meets all specified constraints.

4.1 Modeling the Rotating Workforce Scheduling problem

The most important and also most challenging part of this thesis, was to model the rotating workforce scheduling problem in a way that allows solving by an SMT-solver. As discussed in Chapter 3, SMT-solvers provide built-in support for many theories (see Section 3.3). I developed two encodings that use functions and predicates from different background theories. The first formulation is based on linear integer arithmetic, whereas the second formulation expresses the rotating workforce scheduling problem with fixed-size bitvectors and bit-level operations.

4.1.1 Linear Integer Arithmetic

For the first modeling approach the theory of linear arithmetic over the integer numbers is used. The encoding uses the *QF_LIA* logic from the SMT-LIB standard, especially the arithmetical operation "+" and the equality predicate (=). Additionally, the *if-then-else* function is needed to model conditional cases used to formulate the necessary assertions.

I start by defining how the schedule can be represented by first-order variables and then I describe how the different constraints can be formulated.

Basics

The resulting schedule of a rotating workforce scheduling instance can be represented by a table consisting of n rows, one for each employee, and w columns, one for each work day. Each cell $x_{i,j}(1 \leq i \leq n)(1 \leq j \leq w)$ stands for the j-th day of the week for employee i. Due to the cyclicity of the schedule we could also interpret i as the i-th week of the schedule.

Rotating workforce scheduling is a CSP with a number of different constraints that need to be fulfilled. To solve a CSP, a set of variables, their possible values and their relations to each other have to be defined.

So the first step is to define a set of variables to represent the workforce schedule. Therefore, we define one variable for each day of the planning period, i.e. $n \times w$ variables. Next, we have to specify the type of these variables. Usually the description files of instances of the rotating workforce scheduling problem assign a character to each shift, e.g. 'N' for night shift or 'D' for day shift. In order to simplify the constraint formulations we will instead use different integer numbers for the shifts, hence the type of our variables will be the *Integers*.

Definition 4 (Schedule representation)**.** Let a schedule S be defined as a $n \times w$ matrix of variables, where n is the number of employees and w the number of work days.

$$S = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & \cdots & x_{1,w} \\ x_{2,1} & x_{2,1} & \cdots & \cdots & x_{2,w} \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ x_{n,1} & x_{n,1} & \cdots & \cdots & x_{n,w} \end{pmatrix} \tag{4.1}$$

Furthermore, we map every shift $a_0, a_1, \ldots, a_{m-1}$ of A (see Section 2.3.1) to an integer number. Therefore, we define a function:

$$f : A \to \mathbb{N} \setminus \{0\} \text{ s.t. for all } a_i \in A \setminus \{a_0\} : f(a_i) = f(a_{i-1}) * n + 1 \text{ and } f(a_0) = 0 \tag{4.2}$$

Every variable $x_{i,j}$ of schedule S is now assigned exactly one of our calculated integer values.

$$\forall x_{i,j} \in S : x_{i,j} = a'_k, \text{ where } a'_k \in A' \text{ and } A' = \{f(a_k) : \forall a_k \in A\ (0 \leq k \leq m-1)\} \tag{4.3}$$

The *SMT-LIB* standard provides a lot of data types, called *sorts* (see Section 3.8.1) that we could use to represent our schedule, like for example arrays or lists. Nevertheless, I decided to declare each variable of matrix S as a standalone variable, because I wanted to keep the number of used background theories as small as possible. However, the formulation with standalone variables also has a drawback: the resulting encoding is larger because not only do we have to declare $n \times w$ variables, but we must also define constraints for every one of them to restrict its value. The declaration of the variables in SMT-LIB is shown below:

```
(declare-fun x_1_1 () Int)
(declare-fun x_1_2 () Int)
(declare-fun x_1_3 () Int)
...
(declare-fun x_n_w () Int)
```

Now we focus on the constraints of rotating workforce scheduling and how they can be formulated using linear integer arithmetic with equalities.

Shift assignment

For every variable we have to define the allowed range of values and ensure that exactly one of the shifts is assigned to each variable. This is guaranteed by the following assertions:

```
(assert (or (= x_1_1 0) (= x_1_1 1) (= x_1_1 10) (= x_1_1 91))
(assert (or (= x_1_2 0) (= x_1_2 1) (= x_1_2 10) (= x_1_2 91))
...
(assert (or (= x_n_w 0) (= x_n_w 1) (= x_n_w 10) (= x_n_w 91))
```

The numbers in the example above are the results of the function f for a schedule with 9 employees, 7 work days and three shifts ('D', 'A', 'N').

Temporal Requirements

The temporal requirements of the rotating workforce scheduling problem, as described in the definition of Section 2.3.1, are defined as a $(m-1) \times w$ matrix R where every element $r_{i,j}$ specifies the required employees for the shift i on the j-th day of the week. The main reason why we use the function f to map every shift character a_i to a natural number is that it allows us to specify the temporal requirement constraints by means of basic linear integer arithmetic. Thus, we can formally define the temporal requirements $T(w_j)$ for a specific day w_j of the week as follows:

$$T(w_j) = \sum_{i=1}^{m-1} r_{i,j} * f(a_i) \tag{4.4}$$

We can now formulate the temporal requirements constraint as an equation that always has to hold:

$$\sum_{i=1}^{n} x_{i,j} = \sum_{i=1}^{m-1} r_{i,j} * f(a_i), \text{ where} (1 \leq j \leq w) \tag{4.5}$$

Since we defined the calculation $f(a_i)$ as the arithmetic operation $f(a_{i-1}) * n + 1$, it is ensured that the required value for $T(w_j)$ can only be obtained if the correct amount of staff is assigned to the corresponding shifts.

The following example illustrates how the constraint is written in the *SMT-LIB* input format. It considers a schedule of 9 employees, 7 work days and 3 shifts. On each day of the week, there are two employees required for every shift, hence the resulting value for $T(w_j) = 204$.

```
(assert
  (=
   (+ x!1!1 x!2!1 x!3!1 x!4!1 x!5!1 x!6!1 x!7!1 x!8!1 x!9!1)
   204
  )
)
(assert
  (=
   (+ x!1!2 x!2!2 x!3!2 x!4!2 x!5!2 x!6!2 x!7!2 x!8!2 x!9!2)
   204
  )
)
...
```

Shift Block Length

To address the block length issue, we have to think of our schedule as a one dimensional sequence of work days instead of the matrix we defined before. This ensures that the last day of each week is adjacent to the first day of the next week. Additionally, we must not forget that in a cyclic schedule the last day of the last week is also adjacent to the first day of the first week. With that in mind we define the block length constraints by considering every variable to be the first in a new block. The neighbors of each of these possibly block-starting variables are then constrained: The block-starting variable itself has a certain shift value, then all following variables of the same block are restricted to the according shift value. For the following definitions we address every variable $x_{i,j}$ by its absolute position in the one dimensional representation of the schedule. Therefore, we omit the second index j as i will range from 1 to $n \times w$.

Minimum Shift Block Length For every x_i, if $x_i = a_j$ (where a_j defines the shift to be constrained) and $x_{i-1} \neq a_j$, then $x_{i+1} = a_j \wedge x_{i+2} = a_j \wedge \ldots \wedge x_{i+min-1} = a_j$.

Maximum Shift Block Length For every x_i, if $x_i = a_j \wedge x_{i-1} \neq a_j \wedge x_{i+1} = a_j \wedge x_{i+2} = a_j \wedge \ldots \wedge x_{i+max-1} = a_j$, then $x_{i+max} \neq a_j$.

The *SMT-LIB* standard provides an if-then-else construct which is very useful for formulating the block constraints. Usage of the else-branch is not necessary. If $(x_i \neq a_j \vee x_{i-1} = a_j)$ then

x_i is not the beginning of a shift block and therefore we do not have to restrict any neighbors of x_i. The SMT-LIB formulation of the above constraint is given below for a minimum shift block length of 2 and a maximum of 4 (9 employees, 7 work days):

```
(assert (ite
          (and
            (= x!1!1 1)
            (not(= x!9!7 1))
          )
          (and(= x!1!2 1))
          true
        )
)
(assert(ite
          (and
            (not(= x!9!7 1))
            (and(= x!1!1 1)(= x!1!2 1)(= x!1!3 1)(= x!1!4 1))
          )
          (not(= x!1!5 1))
          true
        )
)
```

Note that the formulation of the shift block constraint must also be applied to the day-off shift a_0 to restrict the minimum and maximum number of consecutive days off.

Work Block Length

We can describe the constraints for the work block length very similarly to those for the shift blocks. The only difference is that we do not care about the shifts the block consists of, unless they are assigned the special "day-off" shift a_0. Below, the formal definition of the work block constraints is given.

Minimum Work Block Length For every x_i, if $x_i \neq a_0 \wedge x_{i-1} = a_0$, then $x_{i+1} \neq a_0 \wedge x_{i+2} \neq a_0 \wedge \ldots \wedge x_{i+min-1} \neq a_0$.

Maximum Work Block Length For every x_i, if $x_i \neq a_0 \wedge x_{i-1} = a_0 \wedge x_{i+1} \neq a_0 \wedge x_{i+2} \neq a_0 \wedge \ldots \wedge x_{i+max-1} \neq a_0$ then $x_{i+max} = a_0$.

Illegal Shift Sequences

Safety concerns and legal reasons make it necessary to forbid certain shift sequences from being assigned to the employees. This means that if on a day x_i an employee works in shift a_j, the employee is not allowed to work in shift a_k the day after. Again, we can express such restrictions with the help of the if-then-else construct.

Illegal Shift Constellations For every x_i it has to hold that if $x_i = a_j$, then $x_{i+1} \neq a_k$ $(k \neq j)$, iff $a_j \rightarrow a_k$ is forbidden.

```
(assert (ite (= x!1!1 91) (not (= x!1!2 1)) true))
```

The example above shows the *SMT-LIB* syntax formulating the constraint that after a night shift 'N', an employee is not allowed to work the following day shift 'D' ($D = 1$ and $N = 91$).

This assertion can be easily adapted to restrict illegal sequences of three shifts, by extending the definition above.

4.1.2 Bitvector Theory

The formulation of the problem in bitvector theory is more complex than the formulation using linear integer arithmetic. Nevertheless, I chose this approach hoping that the low-level operations of machine arithmetic would allow a more compact representation of the constraints which might result in a better performance than the linear arithmetic method.

In the following sections I first introduce the basic idea of how the rotating workforce scheduling problem can be formulated with bitvectors and then I discuss in detail how to model the specific constraints.

Basics

By definition a bitvector in *SMT-LIB* format may be of arbitrary length. Therefore I propose using bitvectors that are the size of the complete schedule for the problem. Thereby each bit of a bitvector represents one day in the schedule. With that in mind, we use one bitvector per shift and one extra bitvector to represent the days-off. We will call the bitvector of a shift a *shiftvector*. A shift is assigned to a specific work day iff the according bit in the shiftvector is set to 1. The formal definition of a schedule represented by bitvectors is given below:

Definition 5 (Bitvector schedule representation)**.** A schedule S is represented by m bitvectors $shift_0, shift_1, ...shift_{m-1}$, where $shift_0$ represents the day-off shift. Each bitvector $shift_i$ $(0 \leq i \leq m-1)$ is of size $n \times w$ where n represents the number of employees and w the number of work days.

The following sample shows the bitvector definitions in *SMT-LIB 2.0* of a schedule with 9 employees ($n = 9$), a 7 day week ($w = 7$) and three shifts.

```
(declare-fun shift0 (_ BitVec 63))
(declare-fun shift1 (_ BitVec 63))
(declare-fun shift2 (_ BitVec 63))
(declare-fun shift3 (_ BitVec 63))
```

The subsequent sections discuss the constraints of rotating workforce scheduling using the bitvector encoding.

Shift Assignment

The shift (including day-off) assignment constraint ensures that exactly one shift is assigned to each day of the schedule. This is accomplished in two steps: we have to make sure that there can be no more than one shift per day and employee and furthermore that there must be at least one shift assignment per day and employee.

For the first part of the shift assignment constraint we use the bitwise *AND* operation. We have to apply this operation to every combination of two shiftvectors and assert that the result of the combination is a bitvector of the same length as the original vectors containing only zero-bits (0).

$$\begin{array}{cccccccccc} & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & (shift_1) \\ \& & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & (shift_2) \\ \hline & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \\ = & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\ \hline & & & & & & & & \text{false} & \end{array} \tag{4.6}$$

The example above shows a violation of this constraint because on two days both $shift_1$ and $shift_2$ have the according bits set to 1 meaning the employee would be assigned both shifts on these days. Although we have now excluded all possible solutions where two or more shifts may be assigned to one employee on the same day, this is not enough. To ensure correctness we also have to rule out that no shift is assigned to a specific day. Recall that this also includes the day-off shift ($shift_0$).

The second assertion that is necessary to ensure only a single shift assigned to each work day, uses the bitwise *OR* operation. If the result of the bitwise *OR* applied to all shiftvectors is equal to the bitvector containing only 1-bits, then at least one shift is assigned to each work day. The following example gives a good illustration of this second assertion:

$$
\begin{array}{cccccccccc}
 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & (shift_0) \\
 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & (shift_1) \\
| & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & (shift_2) \\
 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & (shift_3) \\
\hline
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \\
= & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \\
\hline
 & & & & & & & & & \text{true}
\end{array}
\tag{4.7}
$$

It is important to observe that this constraint does not forbid multiple assignments. Hence, only the combination of the two constraints ensures the assignment of **exactly** one shift per day and employee.

Temporal Requirements

Instead of only regarding the structure of our shiftvectors as having the length of the whole schedule ($n \times w$), we have to think about the schedule as a multidimensional array, consisting of n arrays with size w, to ensure that the correct number of employees is assigned to each shift of a day. Note that a shiftvector is read from right to left, meaning that the first week of the schedule starts with the least significant bit.

$$
\begin{array}{cccccccccccccccccccc}
1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\
\underbrace{\qquad\qquad\qquad} & & & & & & & \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad} & & & & & & & \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad} & & & & & & \\
\textit{week 3} & & & & & & & \textit{week 2} & & & & & & & \textit{week 1} & & & & & & \\
 & & & & & & & \Downarrow & & & & & & & & & & & & & \\
 & & & & & & 0 & 1 & 1 & 1 & 0 & 0 & 1 & \textit{(week 1)} & & & & & & & \\
 & & & & & & 0 & 0 & 0 & 1 & 1 & 0 & 1 & \textit{(week 2)} & & & & & & & \\
 & & & & & & 1 & 1 & 0 & 0 & 1 & 1 & 1 & \textit{(week 3)} & & & & & & &
\end{array}
\tag{4.8}
$$

Because of the cyclicity of the schedule, every week also stands for the current shift schedule of one employee. Hence, every column of our multidimensional array represents the shift plan for all employees on that day of the week. In the formulated constraint the number of 1-bits in each column must be equal to the number of employees that should be assigned to the corresponding shift on that day. Therefore, we first have to create a temporary bitvector by extracting the correlating bits of the original shiftvector and concatenating them to a new bitvector that represents that day.

$$
\begin{array}{cccccccccccccccccccc}
1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\
 & & & & & & & & & & \Downarrow & & & & & & & & & & \\
 & & & & & & & & & & 1 & 1 & 1 & \textit{(Monday)} & & & & & & & \\
 & & & & & & & & & & 1 & 0 & 0 & \textit{(Tuesday)} & & & & & & & \\
 & & & & & & & & & & 1 & 1 & 0 & \textit{(Wednesday)} & & & & & & & \\
 & & & & & & & & & & \multicolumn{3}{c}{\ldots} & & & & & & & &
\end{array}
\tag{4.9}
$$

The *SMT-LIB* input language provides an extract operation to select certain bits within a bitvector. These bits can be concatenated with the help of the concat operator. The code sample below extracts every 7-th, bit starting with the first one and concatenates them to a new bitvector.

```
(concat ((_ extract 0 0) shift1)
        (concat ((_ extract 7 7) shift1)
                (concat ((_ extract 14 14) shift1))
        )
)
```

In the next step we count all bits in the bitvector that are set to 1. To accomplish that, we calculate the *Hamming Weight* of our vector. In the following a short definition of the *Hamming Weight* is given:

Hamming Weight "The Hamming distance $d(u, v)$ between two words u and v, of the same length, is equal to the number of symbol places in which the words differ from one another. If u and v are of finite length n then their Hamming distance is finite since $d(u, v) \leq n$."[34] Introduced by *Richard Hamming* in [57], the Hamming distance is used in telecommunication for error detection and error correction. The *Hamming Weight* is simply defined as the Hamming distance of a word u and the *zero-word*. Hence, it represents the number of symbols which are different from the *zero-symbol*.

input : A bitvector v
output: The Hamming weight of bitvector v

```
1 while v ≠ 0 do
2 |   if v & 1 then
3 |   |   weight = weight + 1;
4 |   end
5 |   v = v » 1;
6 end
7 return weight
```

Algorithm 4.1: Hamming Weight algorithm

The original algorithm to calculate the Hamming Weight has a runtime that depends on the length of the input word, because it checks if every single symbol of the word differs from the

zero-symbol. This approach is not ideal for our use, as it cannot be formulated in a declarative way and therefore is not suitable for us in the constraint formulation.

A better way to calculate the Hamming Weight is provided by the algorithm developed by Peter Wegner in 1960 [93], which is based on the fact that if the bitwise AND operation is applied to a binary number N and $N - 1$, then the resulting number has one 1-bit less than the original number N. In other words, the least significant 1-bit is eliminated.

input : A bitvector v
output: The Hamming weight of bitvector v

```
1 while v ≠ 0 do
2 |   weight = weight + 1;
3 |   v = v & (v- 1);
4 end
5 return weight
```

Algorithm 4.2: Wegner's Method to calculate the Hamming Weight

Wegner's method offers the advantage that it can be formulated declaratively using recursion, because the number of iterations (recursive calls) is equal to the number of set bits in the input vector. We formally define:

Definition 6. The number of selected bits n in a bitvector v can be calculated by the recursive application of $v = v \ \& \ (v - 1)$ until $v = 0$.

For formulation in the *SMT-LIB* input format we define a helper function that takes a bitvector v as input and returns the result of the operation $v = v \ \& \ (v - 1)$. The code sample below shows the definition of the helper function for a schedule with 9 employees. bvand is the bitwise AND operator and bvsub provides a bitvector subtraction.

```
(define-fun bitcount ((x (_ BitVec 9))) (_ BitVec 9)
      (bvand x (bvsub x (_ bv1 9)))
)
```

To make sure that a certain bitvector has exactly n bits that are set to 1, we define one assertion which guarantees that after $n - 1$ applications of the bitcount function the bitvector is **not** equal to 0 and another assertion that ensures the vector is equal to 0 after n calls of the bitcount function. These assertions are defined for the previously created bitvectors of each work day.

Maximum Shift Block Length

Shift blocks are a sequence of 1-bits in a shiftvector. The length of this sequence needs to be constrained to prevent an employee from working an illegal number of consecutive work days of the same shift. To accomplish that, we need to count the consecutive 1-bits of a shiftvector.

We can find two consecutive bits in a bit vector v, by shifting the vector one bit to the right $v >> 1$ and combine it with the original vector using the AND operation $v \;\&\; (v >> 1)$. Then the bit v_i is set to 1 iff v_{i+1} is set to 1 in the original vector. Hence, with every application of this operation the left most bit of a block is replaced by a 0.

To find n consecutive bits we can expand our equation to this generalized one:

$$0 = v \;\&\; (v >> 1) \;\&\; (v >> 2) \;\&\; \ldots \;\&\; (v >> n) \tag{4.10}$$

$$\begin{array}{cccccccccccc|l}
 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & (v) \\
\& & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & (v >> 1) \\
\hline
 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & (v') \\
\& & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & (v >> 2) \\
\hline
 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & (v') \\
\& & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & (v >> 3) \\
\hline
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (v')
\end{array} \tag{4.11}$$

Thus, we can use this kind of algorithm to restrict the maximum length (max) of a shift block, since the vector has to be equal to 0 after exactly max iterations of $v' = v' \;\&\; (v >> i)$ with $1 \leq i \leq max$, v being the original shiftvector and v' being the resulting modified shiftvector v (see Example 4.11 for a maximum of 3).

Another problem we have yet to deal with is that of cyclicity. As we are designing a rotating schedule, the least and the most significant bit (*lsb/msb*) of each shiftvector are adjacent, which is important for the shift block constraints. To incorporate that, we simply concatenate every shiftvector with itself so that the *lsb* and the *msb* become adjacent in the middle of the new vector.

$$\begin{array}{cccccccccccccccc|l}
 & & & & & & & & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & (v) \\
 & & & & & & & & & & \Downarrow & & & & & & \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & (v \cup v)
\end{array} \tag{4.12}$$

Below the assertions for the maximum shift block length according to the SMT-LIB standard are given. The example defines a maximum length of 4 for the night-shift (*shift3*) of a schedule with 9 employees and 7 work days. Notice that the shiftvector is concatenated with itself to incorporate the cyclicity issue. The bvlshr operation implements the logical shift right we use to formulate this constraint.

```
(assert (=
    (bvand
      (concat shift3 shift3)
      (bvand
        (bvlshr (concat shift3 shift3) (_ bv1 126))
```

```
      (bvand
        (bvlshr (concat shift3 shift3) (_ bv2 126))
        (bvand
          (bvlshr (concat shift3 shift3) (_ bv3 126))
          (bvlshr (concat shift3 shift3) (_ bv4 126))
        )
      )
    )
  ) (_ bv0 126)))
```

Minimum Shift Block Length

The next step is to ensure the minimum length of a shift block. This constraint is more difficult to formulate declaratively than the maximum shift block constraint. Recall that we encoded the assertion of the maximum shift block length as a comparison of the modified shiftvector (see Example 4.11) to the zero-vector of the same length. Thereby we ensured that after max iterations of the consecutive bitcount method the vector has to equal to 0, otherwise it contains a shift block that is longer than max. In case of the minimum shift block length (min) it is not that simple however.

If we tried the same approach as for the maximum shift block length by applying the consecutive bit count algorithm min times in order to assert that the vector equals to 0, this would clearly be wrong. As min only represents the minimum block length, longer sequences of consecutive bits are still allowed to exist. Hence, the resulting shiftvector not necessarily equals to 0.

Another attempt could be to use the consecutive bit count algorithm only $min - 1$ times and assert that the vector does **not** equal to 0 after these $min - 1$ applications. Nonetheless, this would not lead to the correct result: Even though shiftvectors where no block is of at least length min would be forbidden, this method does not ensure that every block in the shiftvector satisfies the minimum length. It is therefore not sufficient to use only our consecutive bit count algorithm to formulate the necessary assertion.

The method I propose to solve the minimum shift block length assertions consists of three steps:

1. **Delete illegal blocks**: Apply $min - 1$ iterations of the consecutive bit count method to the shiftvector, where min is the minimum block length. After $min - 1$ applications at least one 1-bit of each legal block is still present, while all blocks of illegal size have been replaced by 0-bits.

2. **Restore blocks**: By iteratively shifting the vector from (1) to the left $min - 1$ times and applying the OR operation to the shifted vector and the one from the previous iteration, the length of all blocks is restored (except the already deleted illegal blocks).

3. **Verify**: In the last step we have to compare the bitvector from (2) with the original

shiftvector. These two vectors are equal iff the original vector did not contain any illegal blocks in the first place, because otherwise they would have been deleted in step (1).

input : A bitvector v
input : The minimum block length min
output: A Boolean indicating that the min block length holds for v or not

```
1 v_2 = v
2 for i = 1 to min − 1 do
3 |   v_2 = v_2 & (v_2 >> 1)
4 end
5 for i = 1 to min − 1 do
6 |   v_2 = v_2 | (v_2 << 1)
7 end
8 return v == v_2
```

Algorithm 4.3: Minimum Shift Block Length

For better illustration of how the algorithm works, consider the following example. Take a shiftvector *shift1* and check if the minimal block length $min = 3$ holds for all shift blocks of this vector.

$$0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \tag{4.13}$$

We will start by applying the consecutive bitcount method $min - 1$ times:

$$\begin{array}{cccccccccccc} & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & \\ \& & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & (>> 1) \\ \hline & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & \\ \& & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & (>> 1) \\ \hline & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \end{array} \tag{4.14}$$

The example above shows that with every iteration of the consecutive bit count method, the left most bit of every 1-block is replaced by a zero. Hence, after $min - 1 = 2$ iterations at least one set bit of each legal block has not been replaced, whereas the illegal blocks – blocks with a length less than min – have vanished completely.

As next step we will restore the original shiftvector by first shifting the vector back to the left $min - 1$ times and then applying the bitwise OR operation to all iterations (see 4.15). This will ensure that every 1-bit of a legal block that has been replaced by the consecutive bit count method is being restored, while the blocks with illegal length are not.

$$
\begin{array}{cccccccccccc}
 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \\
| & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & (<<1) \\
\hline
 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & \\
| & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & (<<1) \\
\hline
 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 &
\end{array}
\tag{4.15}
$$

The resulting vector is now compared to the original shift vector. The constraint is successfully fulfilled iff the two vectors are equal.

$$
\begin{array}{cccccccccccc}
 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & \\
= & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & \\
\hline
 & & & & & & & & & & & & \text{false}
\end{array}
\tag{4.16}
$$

So far we have left out the fact that our algorithm does not consider the cyclicity of the schedule meaning that the first and the last bit of a shiftvector are adjacent. To overcome this issue, we have to refine our algorithm in a way that incorporates rotating schedules. Suppose we have the following shift vector:

$$
1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \tag{4.17}
$$

If we constrain the minimal block length to 3, the shiftvector will be legal, because we have two blocks of length 3, one being in the middle of the vector and one that consists of the first and the last two bits of the vector. Before we can make use of our algorithm, we therefore have to concatenate this split block, since otherwise the algorithm would falsely classify two blocks as illegal, one each at the beginning and the end of the vector. Thus, by moving the illegal block at the end in front of the most significant bit, we merge these blocks together and solve this issue.

We start out by creating a vector that only contains the last 1-block of the original shiftvector.

¬	1	0	0	1	1	1	0	1	1	(v)
&	0	1	1	0	0	0	1	0	0	(¬v)
	0	1	1	0	0	0	0	1	1	(¬v - 1)
⊕	0	1	1	0	0	0	0	0	0	(v')
	0	1	1	0	0	0	1	0	0	(¬v)
2-complement	0	0	0	0	0	0	1	0	0	(v')
	1	1	1	1	1	1	1	0	0	(v')
¬	0	0	0	0	0	0	0	1	1	(v')

(4.18)

Table 4.18 shows how the vector can be modified, so that only the least significant 1-block is present. We could now concatenate this vector with the original shift vector (see 4.19) in order to merge the split block.

$$
0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \tag{4.19}
$$

However, there is still a problem. Although we have concatenated the two vectors and thus considered the split block as a legal block, we still have two bits at the end of the vector, that form an illegal block. Therefore, before concatenating the two vectors, we apply the XOR operation to them and then concatenate the vector from 4.18 with the resulting vector of that operation:

$$\begin{array}{cccccccccc} & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ \oplus & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{array} \qquad (4.20)$$

$$\Downarrow$$

$$0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0$$

The vector from 4.20 represents the correct non-cyclic shiftvector. The minimum shift block length algorithm (Algorithm 4.3) applied to this vector will indicate whether the shiftvector contains only legal blocks or not.

Now we want to have a look at the assertions in the SMT-LIB format to understand how this constraint is formulated. The example below shows the constraint for a minimum shift block length of 2. First the non-cyclic shiftvector is created and stored in the noncyclic_shift3 variable. Then a helper variable (shift3_helper) is defined. This variable is then assigned to the resulting bitvector of $min - 1$ applications of our bitcount algorithm. The last assertion defines the comparison between the original non-cyclic shiftvector and the vector of the helper variable after it has been restored (see step *(2)* and *(3)* of the algorithm).

```
(declare-fun shift3_justLastBlock () (_ BitVec 63))
(declare-fun shift3_withoutLastBlock () (_ BitVec 63))
(declare-fun noncyclic_shift3 () (_ BitVec 126))
(declare-fun shift3_helper () (_ BitVec 126))

(assert (=
  shift3_justLastBlock
  (bvnot (bvneg (bvxor
    (bvnot shift3)
    (bvand (bvnot shift3) (bvsub (bvnot shift3) (_ bv1 63)))))
  )))
(assert (=
  shift3_withoutLastBlock
  (bvxor
    shift3
    (bvnot (bvneg (bvxor
      (bvnot shift3)
      (bvand (bvnot shift3) (bvsub (bvnot shift3) (_ bv1 63)))
    )))
  )
```

```
))
(assert (=
  noncyclic_shift3
  (concat shift3_justLastBlock shift3_withoutLastBlock)
))

(assert (=
  shift3_helper
  (bvand noncyclic_shift3
         (bvlshr noncyclic_shift3 (_ bv1 126))
  )
))
(assert (=
  noncyclic_shift3
  (bvor shift3_helper (bvshl shift3_helper (_ bv1 126)))
))
```

Work Block Length

The work block constraints are very similar to the shift block constraints. The only difference is that a work block is a sequence of consecutive days where an employee has to work regardless of what shift he/she is assigned to. We could get a vector encompassing all the work blocks by simply applying the bitwise OR on all shiftvectors, except the special "day-off" vector. Another way to obtain that vector is to negate the "day-off" vector, i.e. replace every 0 by 1 and every 1 by 0. The 1-bits in the resulting vector then represent all work-days, whereas the 0s stand for the days off. The easiest way to negate a bitvector is to use the NOT operation .

$$\begin{array}{cccccccccc} \neg & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & (shift_0) \\ \hline & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & \neg(shift_0) \end{array} \tag{4.21}$$

We now use the same constraint formulations as for the shift block length on this new vector to restrict the length of consecutive work days.

Illegal shift constellations

Certain shift constellations are considered illegal, because every employee needs a fixed amount of resting time between two workdays. Therefore, it is not allowed for example, to work the day-shift the day after working the night-shift.

$$\begin{array}{cccccccccc} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & \text{(Night shift)} \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & \text{(Day shift)} \end{array} \tag{4.22}$$

These constellations can be forbidden with the help of bit rotation of a shiftvector. Considering the example above, to prevent the shift constellation *Night* $\rightarrow$ *Day* we have to rotate the shiftvector of the night shift N by 1 to the right. We can now AND this vector with the shiftvector of the day shift D and check if they have no bits in common that are set to 1. Subsequently, for any position i, $D_i \neq N_i$ has to hold. Hence, the resulting bitvector must be 0.

	0	1	0	0	0	0	1	1	(Night shift)
	1	0	1	1	1	0	0	0	(Day shift)
							$\Downarrow$		
$\circlearrowleft$	0	1	0	0	0	0	1	1	(Night shift)
&	1	0	1	0	0	0	0	1	(rotated Night shift)
	1	0	1	1	1	0	0	0	(Day shift)
=	1	0	1	0	0	0	0	0	
	0	0	0	0	0	0	0	0	
							false		

(4.23)

The example also illustrates why it is important to use bit rotation instead of a logical shift right operation. If we had shifted the night shift vector to the right, we would have missed the fact that the least and most significant bit of each vector are adjacent.

The problem definition also allows illegal shift sequences of length 3 to be specified. The algorithm for detecting such illegal sequences is still very similar, except that we have to take three shiftvectors into consideration.

Suppose we want to restrict the shift sequence *Night* $\rightarrow$ *Day-Off* $\rightarrow$ *Afternoon* from being assigned to the employees. These are our three shiftvectors:

0	1	0	0	0	0	0	1	1	(Night shift)
0	0	1	0	0	1	1	0	0	(Day-off shift)
1	0	0	1	1	0	0	0	0	(Afternoon shift)

(4.24)

We start out by rotating the night shiftvector by 1 to the right and AND it with the day-off vector. Now we rotate this resulting vector by 1 to the right and apply the AND operation on this vector and the shiftvector of the afternoon shift. If the result isn't equal to 0, we have found at least one illegal shift sequence of the form *Night* $\rightarrow$ *Day-Off* $\rightarrow$ *Afternoon*.

	0	1	0	0	0	0	0	1	1	(Night shift)
	0	0	1	0	0	1	1	0	0	(Day-off shift)
	1	0	0	1	1	0	0	0	0	(Afternoon shift)
							⇓			
↻	0	1	0	0	0	0	0	1	1	(Night shift)
&	1	0	1	0	0	0	0	0	1	(rotated Night shift)
	0	0	1	0	0	1	1	0	0	(Day-off shift)
↻	0	0	1	0	0	0	0	0	0	(intermediate vector)
&	0	0	0	1	0	0	0	0	0	
	1	0	0	1	1	0	0	0	0	(Afternoon shift
=	0	0	0	1	0	0	0	0	0	
	0	0	0	0	0	0	0	0	0	
						false				

(4.25)

In this chapter I have shown that the rotating workforce scheduling problem and thus all its constraints can be formulated using both linear integer arithmetic and the theory of fixed-size bitvectors.

CHAPTER 5

Experimental Results

In Chapter 4, I proposed two different modeling techniques for the rotating workforce scheduling problem. To evaluate their performance and to get an impression of whether SMT-solving is an efficient approach to rotating workforce scheduling, these encodings were tested on benchmark instances from literature. Five state-of-the-art SMT-solvers were selected to solve these instances.

This chapter starts with a description of the different benchmark instances and continues with an overview of the selected SMT-solvers. Additionally, the experimental settings, including the specific versions of the solvers, hardware specifications and used parameters, are explained in detail. To adjust how SMT problems are solved, different parameters and options are available for most modern SMT-solvers. I tested a subset of options for each of the solvers and repeatedly evaluated the performance impact of different settings in order to obtain the best possible results. Next, I present the results of each modeling approach separately to determine which solver performs best on each approach and then I compare the fastest solvers to each other to verify whether one of the modeling techniques is better suited for rotating workforce scheduling. Finally, I investigate how competitive my approach is compared to other solving techniques that were already proposed in the literature.

5.1 Benchmark Instances

The 20 instances described in [73] consist of three problems that have already been used in the literature and 17 instances that were derived from real life problems of different business areas (see Table 5.1).

The first instance was solved by Butler in [29] and describes a small problem with 9 employees, 7 work days and 3 shifts. The second instance is taken from [68] and the third one was proposed in [58] and characterizes a larger problem with 17 employees, 7 work days and 3 shifts. Note

that although these instances are based on the problems in [58], [29] and [68] there are some minor differences to the original specifications as outlined in [73].

The remaining 17 instances introduced in [73] have been created from the "shifts and temporal requirements of real life problems which appeared in a broad range of organizations, like airports, factories, health care organizations, etc"[73].

Table 5.1 provides a more detailed description of these 20 instances. The table is based on *Table 3* from [73] but moreover includes the three instances from literature ([29], [68], [58]). All instances have a planning period of 7 work days. Note that the temporal requirements are not provided in this table but the complete instance description can be downloaded[1]. Two different illegal shift constellations are defined for the instances in the table (*seq1, seq2*). The letters in squared brackets state one illegal shift sequence, for example [N D] means that it is not allowed to work the day shift after working the night shift.

- **seq1:** [N D], [N A], [A D]
- **seq2:** [N - N], [A - D], [N - A], [N - D]

5.2 The Solvers

To compare the results, I selected five state-of-the-art SMT-solvers able to solve first-order formulas in the theory of bitvectors and linear arithmetic. Another selection criterion was their performance at the annual SMT competition (SMT-COMP[2]). Below is a short description of the solvers used in this thesis:

Z3 [38] is a DPLL(T) [52] based SMT-solver developed at *Microsoft Research*. It supports all popular theories provided by the SMT-LIB standard. Microsoft has integrated Z3 as a core component for many products which are used for testing or software verification like Pex [89], HAVOC [65] and SLAM/SDV [11] that is used for *Windows Device Driver* verification. I have selected Z3 because it won 17 categories (main track) at the 2011 *SMT-COMP* competition.

MathSAT 5 [33] is an SMT-solver that was developed in a cooperation project between the *Foundazione Bruno Kessler*(FBK) research organization in Trento and the University of Trento. MathSAT was also targeted to be used for formal verification of software and is also integrated in some industrial projects. For more detailed information on MathSAT 5 the reader is referred to [33].

CVC 4 [19] The Cooperating Validity Checker, was developed by researchers of New York University and the University of Iowa. The solver's 4th version is a complete architectural

[1] Instances available from *http://www.dbai.tuwien.ac.at/staff/musliu/benchmarks/*

[2] http://smtcomp.sourceforge.net

Instance	Employees	Shifts	Block Constraints			Illegal Seq.
			Shift (D,A,N)	Work days	Day-Offs	
1	9	3	2-7, 2-6, 2-4	4-7	2-4	seq1
2	9	3	4-7, 4-7, 4-7	4-7	2-4	seq1
3	17	3	2-7, 2-6, 2-5	4-7	2-4	seq1
4	13	3	2-6, 2-6, 2-4	3-7	1-4	seq1, seq2
5	11	3	2-6, 2-5, 2-4	4-7	1-4	seq1, seq2
6	7	3	2-6, 2-6, 2-6	4-7	1-4	seq1, seq2
7	29	3	2-7, 2-6, 2-5	4-7	2-4	seq1
8	16	3	2-7, 2-6, 2-5	3-7	2-4	seq1
9	47	3	2-7, 2-7, 2-6	2-7	2-4	seq1
10	27	3	2-7, 2-6, 2-5	4-7	2-4	seq1
11	30	3	2-6, 2-5, 2-4	3-7	2-4	seq1
12	20	2	2-6, 2-5	4-7	2-4	[A D]
13	24	3	2-6, 2-5, 1-4	3-7	2-4	seq1
14	13	3	2-6, 2-5, 2-4	4-7	1-5	seq1, seq2
15	64	3	2-6, 2-6, 2-5	3-6	1-4	seq1, seq2
16	29	3	2-6, 2-5, 2-4	4-7	2-4	seq1
17	33	2	2-6, 2-5	3-7	2-4	[A D]
18	53	3	2-7, 2-6, 2-5	4-7	2-4	seq1
19	120	3	2-7, 2-5, 2-4	3-7	2-4	seq1
20	163	3	2-6, 2-6, 2-5	3-6	1-4	seq1, seq2

Table 5.1: Detailed instance description

redesign of its predecessors (CVC [86], CVC Lite [13], CVC 3 [14]). It is also based on the DPLL(T) architecture and supports a various number of theories.

Yices [45] was developed by Bruno Dutertre and Leonardo de Moura at the Stanford Research Institute (SRI). Leonardo de Moura is also one of the main developers of the Z3 solver and is currently a principal researcher at Microsoft Research. Therefore, some features and algorithms of Yices are also integrated in Z3.

Yices is available in two different versions, namely *Yices 1* and *Yices 2*. While both of them are still under development, Yices 2 provides some improvements in usability and performance, and allows for a better integration into other software products via its API. A detailed description of the Yices architecture can be found in [45].

Boolector [25] is an SMT-solver for fixed-sized bitvectors and the extensional theory of arrays. It is based on the eager approach for SMT that uses term-rewriting and bit blasting for translation of first-order formulas into SAT. Developed by Robert Brummayer and Armin Biere at the Johannes Kepler University in Linz, it first participated in the 2008 SMT-COMP and won both of its categories (QF_BV and QF_AUFBV).

5.3 Experimental Setup

This section provides a detailed account of the experimental setup used to obtain the test results. As I already mentioned, the selected solvers were used on 20 problem instances (see Section 5.1) with practical relevance.

5.3.1 Versions of SMT-solvers

The following versions of the SMT-solvers were used for the experimental evaluation.

- Z3 version 4.3.1
- MathSAT5 version 5.2.5 (gmp 5.0.5, clang/LLVM 3.1, 64-bit)
- CVC4 version 1.0 (compiled with GCC version 4.2.1)
- Yices version 2.1.0
- Boolector version 1.5.115

5.3.2 Hardware Specification

The solving process was performed on an Apple MacBook Pro (early 2011) with the following hardware details:

- 2.0GHz quad-core Intel Core i7 (Sandy-Bridge)
- 8GB DDR3 RAM (1333 MHz)
- SAMSUNG SSD 830 256GB
- AMD Radeon HD 6490M and Intel HD Graphics 3000
- Mac OS X 10.8.3 (Mountain Lion)

5.3.3 Details on the solving process

Each of the 20 instances is translated to the SMT-LIB format using the two modeling approaches I proposed in this work (see Section 4.1). While the solvers Z3, MathSAT 5 and CVC 4 support the SMT-LIB 2.0 format, Yices and Boolector only understand input files that conform to the SMT-LIB 1.2 standard. Each solver is invoked for every input file of the two modeling approaches created from the benchmark instances. Note that Boolector does not support linear integer arithmetic and is therefore not considered in the evaluation of that approach.

The runtime for each solver is determined with the Unix utility time or rather its Mac OS variant gtime. The utility timeout (on Mac OS: gtimeout) is used to stop a solver in case it has not found a solution after a certain time period. The timeout for each solver to find a model for one problem instance is set to 1000 seconds.

Every model is translated into a human readable output format and validated against the instance specification to ensure that it is indeed a solution for the problem.

5.3.4 Parameters and Settings

In order to obtain good results for each solver I evaluated the performance impact of different settings and parameters. Some of the tested solvers provide a large number of different options, making it impossible to evaluate all possible settings within the scope of this thesis, as the timeout of 1000 seconds would have made this process very tedious. Therefore, I selected those parameters that seemed most useful according to their, albeit short, description provided by the solvers' software packages.

Below, the evaluated parameters for the selected solvers per modeling approach are listed. For every option it is stated in squared brackets whether it had any positive impact on the runtimes or not.

Linear Arithmetic Encoding

Z3 provides an auto configuration feature that heuristically chooses the best settings and parameters by analyzing the characteristics of the SMT formulas. This feature seems to do a good job as I was not able to improve the performance of Z3 by manually selecting some parameters. Nonetheless, different settings were tested. Note that Z3 implements so-called *strategies* that have to be specified in the SMT-LIB files of the benchmark instances.

- (check-sat-using qflia) [no-improvements]:
  This is the built-in strategy for QF_LIA formulas.
- (check-sat-using smt [no-improvements]:
  This strategy uses Z3's main solver for the given instance.

MathSAT allows its settings to be changed via command-line options. Different parameters for the supported background theories are available and thus allow a fine-tuning of the solving process. For the LIA approach I tried two different options:

- -preprocessor.simplification=1 [no-improvements]:
  This option controls the simplifications that are performed during preprocessing. The value 1 indicates that the solver tries to simplify if-then-else terms as these terms are extensively used for the LIA encoding.

- theory.la.detect_euf_frag=true [no-improvements]:
  This option detects problems that can be solved by EUF only, to avoid calling the T-solver for linear arithmetic. I chose this option because many assertions in the LIA encoding only use equalities and if-then-else terms.

Yices does not provide any options to configure its solving process.

CVC4 provides a lot of command-line options for its linear arithmetic solver and also allows to specify different simplification methods for the input formulas. I evaluated a small subset of the available options.

- –ite-simp [no-improvements]:
  This parameter enables the simplification rules for if-then-else terms.
- –minisat-elimination [no-improvements]:
  This option turns on elimination for the *Minisat* solver.
- –enable-arith-rewrite-qualities [no-improvements]:
  This option turns on the preprocessing rewrite and thereby translates equalities into a conjunction of inequalities.
- –arih-prop=MODE [no-improvements]:
  This decides which attempts of propagation arithmetic are performed during the search. All different modes were tested (none, unate, bi and both) but none of them had any impact on the solving times.

Bitvector Encoding

Z3 The following strategies were used for the bitvector encoding:

- (check-sat-using qfbv) [no-improvements]:
  The built-in strategy for QF_BV theory is applied.
- (check-sat-using qfbv-sls) [no-improvements]:
  This strategy tries to solve the formulas using stochastic local search for QF_BV.
- (check-sat-using smt) [no-improvements]:
  This strategy uses Z3's main solver for the given instance.
- (check-sat-using (using-params qfbv :blast-full true) [no-improvements]:
  This strategy blasts every term with a bitvector sort.

MathSAT Below, the evaluated settings for the bitvector formulation are described:

- -theory.bv.bit_blast_mode=1 [no-improvements]:
  With that option MathSAT uses an intermediate And-Inverted Graphs (AIG) representation and *Tseitin CNF conversion*.

- -theory.bv.bit_blast_mode=2 [improved results]:
  This option uses intermediate AIG representation and CNF conversion with logic synthesis. The evaluation of this parameter showed significant runtime improvements and is therefore used for the MathSAT solver on all tested benchmark instances.
- -theory.bv.eager=false [improved results]:
  This parameter prevents bitvector atoms from being bit-blasted into the main DPLL solver. This option improved the runtime of MathSAT on all instances except 12 and 18.
- -preprocessor.simplification=2 [no-improvements]:
  Simplifications for bitvector formulas are applied to the given instance.

Yices does not provide any options to configure its solving process.

CVC4 offers the following parameters for the bitvector theory module:

- –bitblast-eager [no-improvements]:
  This option eagerly bitblasts the bitvectors to the main SAT solver.
- –bitblast-share-lemmas [no-improvements]:
  This parameter enables lemma-sharing between the bitblasting solver and the main solver
- –bitblast-eager-fullcheck [no-improvements]:
  This option configures the solver to check the bitblasting eagerly.

Boolector uses a rewriting engine to simplify the input formulas. It specifies three levels of term-rewriting and per default all levels are applied to the input formulas. I experimented with this option to evaluate whether reducing the application of some rewrite levels has an impact on the solving times.

- -rwl0 [improved results]
- -rwl1 [no-improvements]
- -rwl2 [no-improvements]

The actual runtime data consists of the best results for each instance and solver, that is to say the results obtained by using settings that positively influenced the solving times.

5.4 Results for Linear Integer Arithmetic

This section presents the results of the linear arithmetic approach (QF_LIA). Four of the five selected solvers support the QF_LIA theory. I could not test *Boolector* as it can only solve formulas of the quantifier-free bitvector theory (QF_BV) and the theory over quantifier-free bitvectors and bitvector arrays with uninterpreted functions (QF_AUFBV).

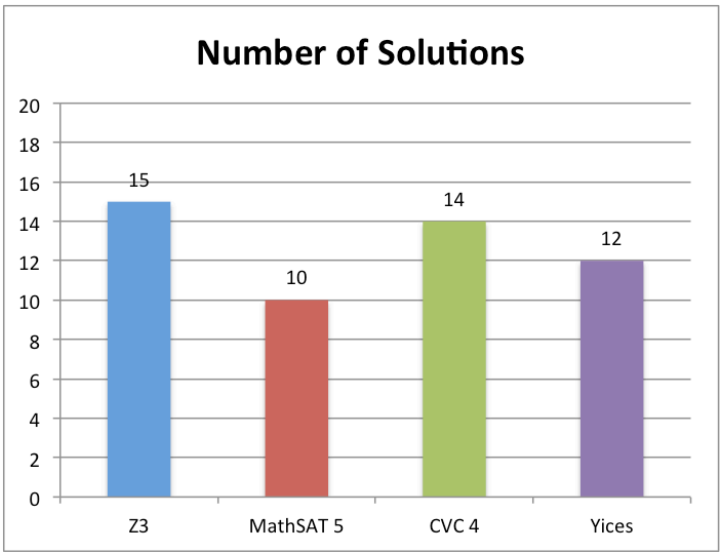

Figure 5.1: Number of found solutions with the LIA encoding

In Figure 5.1 the performance of the different SMT-solvers is compared by the respective number of instances they could solve. Z3 is able to solve 15 out of 20 instances and thereby provides the highest solution rate. CVC finds a solution for 14 instances, Yices is able to solve 12 benchmarks and MathSAT succeeds in 10 instances.

Although these results give a first clue on which solvers perform best and are able to find a solution for most of the instances, it is important to consider the time that was necessary to solve each of the instances. A reasonable solving technique has to find a solution for a specific instance within short time, because human decision makers need the results for further planning or discussion. Thus, we have to compare the runtimes of the solvers for each instance to get a better understanding of the performance of this approach. Figure 5.2 shows that *Z3* provides good solving times on most instances where it can find a model. However, the runtime of Z3 on instance 9 almost reached the timeout of 1000 seconds and is therefore not reasonable anymore. The benchmarks 7, 15, 18, 19 and 20 seem to be problematic for all of the tested solvers. While 15, 18, 19 and 20 have a large number of employees and are therefore hard to solve, it is not clear why instance 7, although it is very similar to instance 16, could not be solved by any of the selected SMT-solvers.

Surprising in Figure 5.2 is the bad runtime of the *Yices* solver on the instances 3, 10, 11, 13, 16

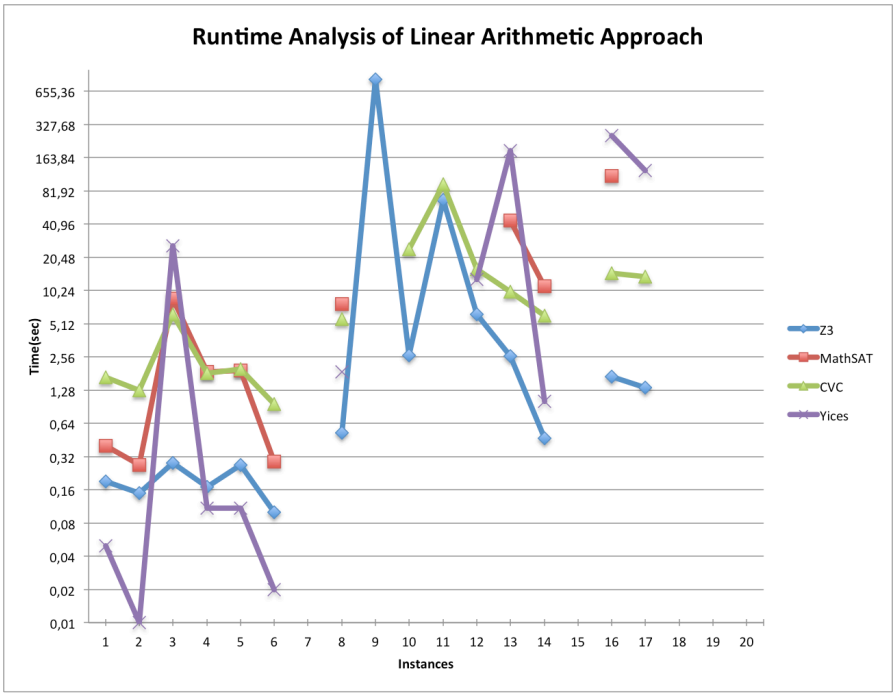

Figure 5.2: Runtime comparison of LIA approach

and 17. According to [38] *Z3*'s theory solver for linear arithmetic is based on the algorithm of *Yices*, which is why I had assumed that *Yices* and *Z3* would provide similar results on the linear arithmetic approach. The detailed solving times for the linear arithmetic approach are listed in Table 5.2.

5.5 Results for Bitvector Theory

The idea of using Bitvector theory as an approach for modeling the rotating workforce scheduling problem was driven by the assumption that it would result in a faster solving time due to its compact representation of the constraints (with respect to the input formulas) and the use of machine arithmetic.

In contrast to the previous approach, all five solvers could be used as the quantifier free theory of fixed-sized Bitvectors is supported by all of them. Again, the SMT-solvers are compared by the respective number of instances they were able to solve, as illustrated in Figure 5.3.

Instance	Z3	MathSAT 5	CVC 4	Yices
1	0.19	0,4	1,68	0,05
2	0,15	0,27	1,27	0,01
3	0,28	8,7	6,22	26,16
4	0,17	1,87	1,82	0,11
5	0,27	1,92	1,99	0,11
6	0,1	0,29	0,96	0,02
7	>1000?	>1000?	>1000?	>1000?
8	0,53	7,7	5,66	1,88
9	833,83	>1000?	>1000?	>1000?
10	2,66	>1000?	24,41	>1000?
11	67,33	>1000?	93,62	>1000?
12	6,22	>1000?	16,06	13,01
13	2,61	44,36	10,01	188,42
14	0,47	11,27	6,06	1,02
15	>1000?	>1000?	>1000?	>1000?
16	1,7	111,51	14,68	260,23
17	1,36	>1000?	13,6	125,31
18	>1000?	>1000?	>1000?	>1000?
19	>1000?	>1000?	>1000?	>1000?
20	>1000?	>1000?	>1000?	>1000?

Table 5.2: Detailed solving times for linear arithmetic approach

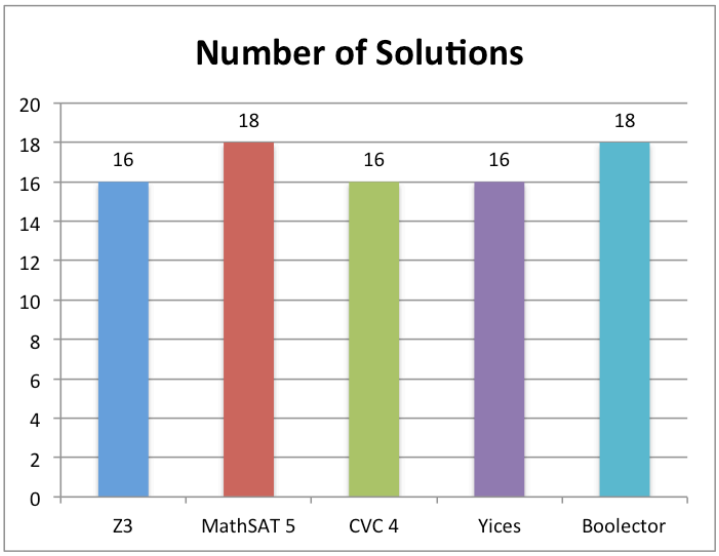

Figure 5.3: Number of found solutions with bitvector encoding

All five solvers achieved a higher solution rate with the bitvector approach than with the linear arithmetic method. MathSAT and Boolector were even able to solve 18 instances, which is two instances more than the best solver (Z3) of the LIA approach. For a better comparison Figure 5.4 shows the runtimes of the different solvers.

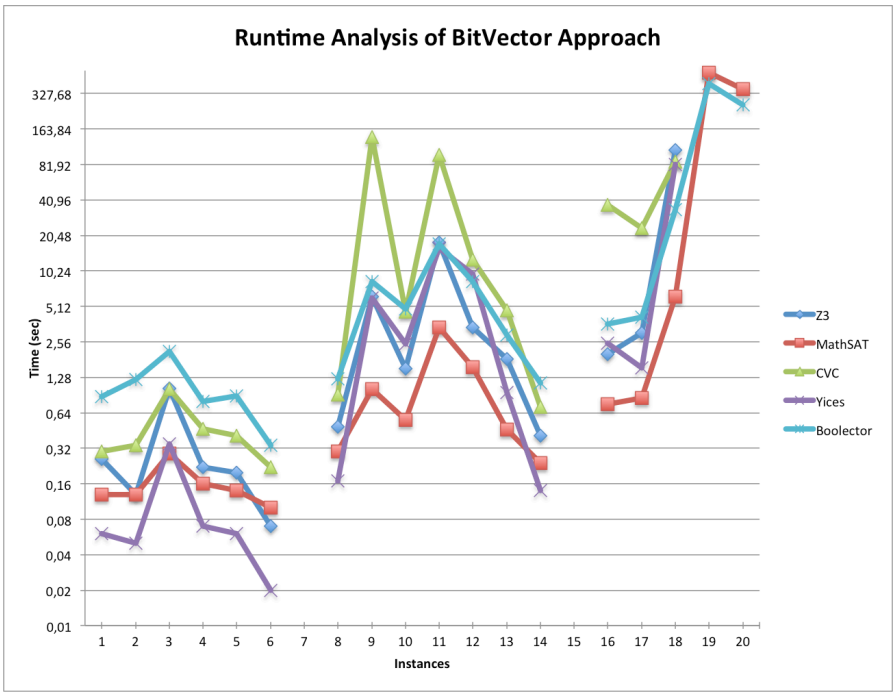

Figure 5.4: Runtime comparison of bitvector approach

The runtime analysis shows that MathSAT performs best at almost all instances. Although Yices provides better results on the first six instances, the runtimes on these instances only differ by a small range of 0.05 to 0.09 seconds between Yices and MathSAT. This variation could be caused by the parsing process of the contrasting SMT-LIB files, since Yices only supports version 1.2 while MathSAT uses the SMT-LIB 2.0 standard. The detailed solving times of each SMT-solver are provided in Table 5.3. All instances on MathSAT were solved with the options theory.bv.bit_blast_mode=2 and theory.bv.eager=false, except for instances 12 and 18, which were solved only with the parameter theory.bv.bit_blast_mode=2. The results of Boolector were achieved with the option rwl0.

Instances 7 and 15 still pose a problem for all considered SMT-solvers. A closer review of the attributes of those two instances shows their peculiarity in comparison to the other ones. While 7 has the lowest ratio between work days and days off, 15 specifies the highest ratio of all instances. This might be the reason for why the two problems are so hard to solve. In instance 7 52% of all days in the schedule have to be assigned to a certain shift, but the remaining 48% need to be days-off. However, the block-length for days off is restricted to the range 2-4 and the work block length is constrained by 4-7. I assume that this makes it hard to meet the temporal

requirements and still assign enough days off. Instance 15, on the other hand, demands that 81% of all days in the schedule are assigned to working shifts. Since the work block length is restricted to 3-6 days, it seems hard to find a schedule that meets the temporal requirements and also fulfills all the other constraints.

Instance	Z3	MathSAT 5	CVC 4	Yices	Boolector
1	0,26	0,13	0,3	0,06	0,88
2	0,13	0,13	0,34	0,05	1,22
3	1,03	0,27	1,01	0,35	2,11
4	0,22	0,16	0,47	0,07	0,8
5	0,2	0,14	0,41	0,06	0,89
6	0,07	0,1	0,22	0,02	0,34
7	>1000?	>1000?	>1000?	>1000?	>1000?
8	0,49	0,3	0,91	0,17	1,24
9	6,16	1,01	140,2	6,04	8,32
10	1,52	0,56	4,6	2,47	3,69
11	17,75	3,37	98,11	16,1	9,76
12	3,37	1,56	12,78	9,63	8,32
13	1,83	0,46	4,76	0,95	2,92
14	0,41	0,24	0,72	0,14	1,14
15	>1000?	>1000?	>1000?	>1000?	>1000?
16	2,01	0,76	37,4	2,5	3,63
17	3,03	0,85	23,62	1,53	4,14
18	109,12	6,23	86,59	81,46	33,92
19	>1000?	489,32	>1000?	>1000?	398,09
20	>1000?	355,21	>1000?	>1000?	262,22

Table 5.3: Detailed solving times for bitvector approach

5.6 Comparison of the two Modeling Approaches

The fastest performing SMT-solvers in each modeling approach are compared to each other in order to evaluate whether one of them is superior for solving the rotating workforce scheduling problem (Figure 5.5). The best performing solver of the linear arithmetic approach was Z3 and MathSAT provides the best results for bitvector theory (see Figures 5.2 and 5.4). We can deduce from Figure 5.5 that the bitvector approach outperforms the linear arithmetic method on all instances and achieves significantly faster solving times. This seems to confirm the initial assumption that modeling the problem with bitvectors has a positive impact on the solvers' runtime due to a more compact representation of the constraints. In contrast to the LIA encoding, the bitvector formulation uses one bitvector variable per shift instead of defining one variable per work day. Consequently, this minimizes the necessary assertions in the encoding and decreases

the length of the SMT-LIB input file. The results show that especially for instances with a large number of employees the solving times using the bitvector approach are considerably faster.

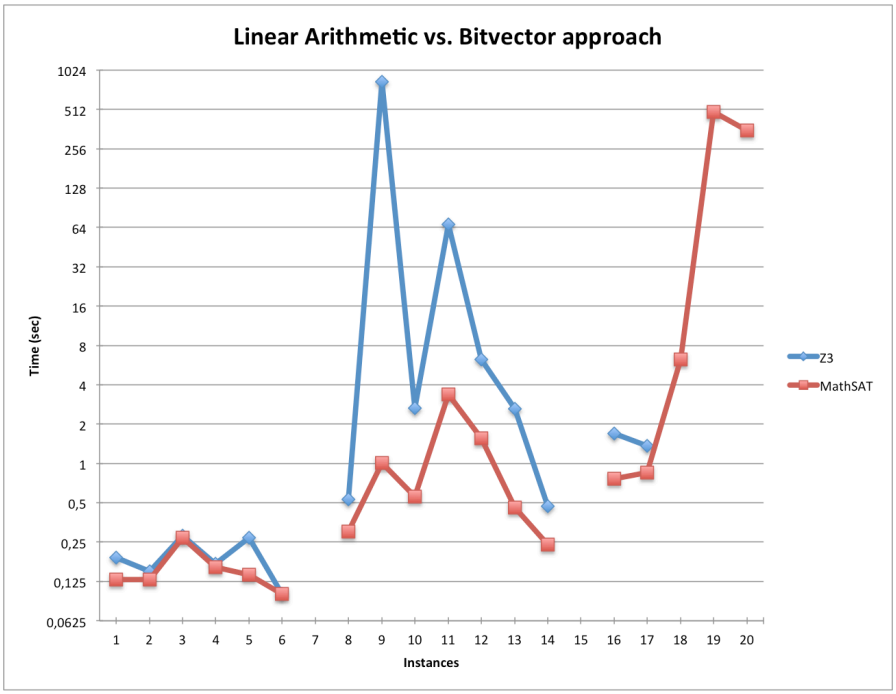

Figure 5.5: Comparison of the fastest solver of each approach

5.7 Comparison to other Approaches

In order to evaluate whether the approach of solving the rotating workforce scheduling problem with SMT-solvers is competitive, we have to compare the results to other solving techniques from literature. Since the MathSAT solver applied to the bitvector approach proved to be the fastest, the solving times from that solver were selected as the best representation for the SMT approach.

First, we compare the results of MathSAT to two other approaches from literature. These approaches have only been tested on three existing instances from literature (Table 5.4). In [10] the problem was solved with a network flow model and in [67] a constraint programming algorithm was used. Table 5.4 provides the solving times for both of these approaches and MathSAT. Although the results cannot be compared directly due to different hardware specifications, it can be observed that MathSAT solves these three instances in a short span of time and thus is competitive to the two approaches.

Instance	MathSAT 5	[10]	[67]
1 [29]	0,13	73,54	3,78
2 [68]	0,13	310,84	0,03
3 [58]	0,27	457,98	10,26

Table 5.4: Comparison of solving times to previous methods in [10], [67]

We now compare the number of solutions and the runtimes to those of the *First Class Scheduler* (FCS) proposed in [74], the min-conflicts heuristic with tabu-search algorithm (MC-T) of [73] and the exact method of [90] that uses constraint programming. The results of these three methods are taken from their respective publications.

Figure 5.6 illustrates the number of benchmark instances that were solved by the different approaches. MC-T solves all 20 instances, however, it is a heuristic technique and should therefore not be directly compared to the exact methods. It is more suitable to compare the SMT-technique to FCS and the constraint programming approach from [90], as these two are also exact methods.

In comparison to the *First Class Scheduler*, the SMT approach solves 5 instances more than FCS. While MathSAT can find a solution to the instances 9,12,13,17,19 and 20, FCS did not finish within 1000s on these instances. On the other hand, FCS is able to solve instance 7 which is not the case for any of the tested SMT-solvers in this thesis.

Comparing the constraint programming approach CP-Rota to the results of MathSAT shows that CP-Rota solves 13 out of 20 instances, meaning five less than the SMT-solver. Although MathSAT is able to find a model for the instances 10, 11, 12, 18, 19 and 20, CP-Rota finds a solution for instance 7 in return.

It is indeed surprising that the other exact methods were not only able to solve instance 7, but to do so in a short time. I could not find an explanation for why SMT-solvers fail to find a solution for this instance, especially as they share many similarities with constraint programming

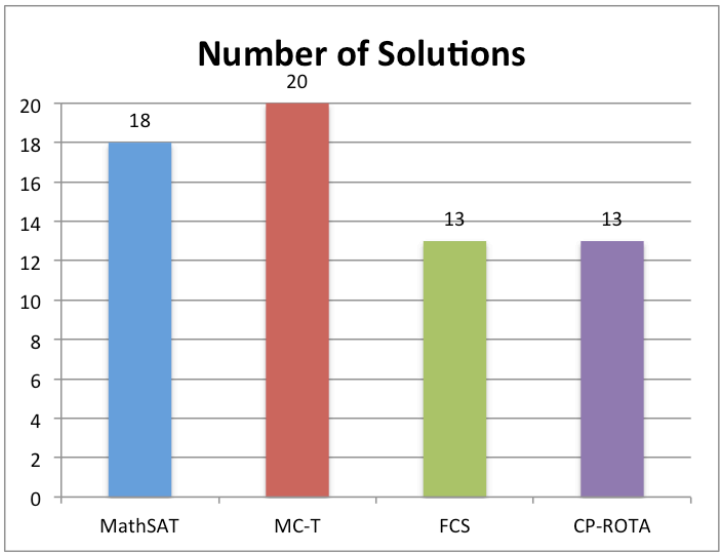

Figure 5.6: Found solutions for different approaches from literature

techniques as stated in [81]. Nevertheless, MathSAT has the highest solution rate of all exact methods considered in this thesis.

Although these results indicate the superiority of SMT-solving over other exact methods, this conclusion should be taken with caution since the runtimes of these techniques were measured on different hardware and are therefore not directly comparable. This raises the question whether MathSAT would still outperform FCS and CP-Rota, if all approaches were run on the same hardware. Nonetheless, we now have a look at the detailed solving times given in Table 5.5:

The runtimes show that the two approaches from literature achieve good results on a number of instances, even though they were tested on a slower hardware. This indicates that FCS and CP-Rota might achieve better results on a faster hardware and thus even outperform the MathSAT solver. However, the SMT approach is particularly fast on most instances (9, 10, 11, 12, 17, 18) that could not be solved by FCS or CP-Rota, allowing the assumption that MathSAT would still outperform the other techniques on these instances if they were tested on the same hardware.

In conclusion, the encoding in SMT is an efficient approach to rotating workforce scheduling, which provides promising results when compared to other exact methods.

Instance	MathSAT 5	MC-T [73]	FCS [74]	CP-Rota [90]
1	0,13	0,07	0,9	0,02
2	0,13	0,07	0,4	0,02
3	0,27	0,42	1,9	0,24
4	0,16	0,11	1,7	0,03
5	0,14	0,43	3,5	0,98
6	0,1	0,08	2	0,02
7	>1000?	52,79	16,1	0,07
8	0,3	0,74	124	964
9	1,01	15,96	>1000?	19
10	0,56	0,60	9,5	>1000?
11	3,37	13,15	367	>1000?
12	1,56	1,17	>1000?	>1000?
13	0,46	0,87	>1000?	114
14	0,24	0,76	0,54	940
15	>1000?	159,04	>1000?	>1000?
16	0,76	0,54	2,44	216
17	0,85	2,16	>1000?	18
18	6,23	6,83	2,57	>1000?
19	489,32	75,83	>1000?	>1000?
20	355,21	71,38	>1000?	>1000?

Table 5.5: Comparison of solving times with other approaches

CHAPTER 6

Conclusions and Future Work

6.1 Summary

In this thesis I have proposed a new approach for solving the rotating workforce scheduling problem. By formulating rotating workforce scheduling as an SMT problem, I was able to use state-of-the-art solving tools (SMT-solvers) for the generation of cyclic workforce schedules. With the help of different background theories I developed two modeling approaches for the problem. To evaluate the performance of these modeling techniques, different SMT-solvers were applied to existing (real-life) problem instances from literature. The results were used to estimate the competitiveness of this new approach to workforce scheduling.

This work covered the following parts:

- Modeling the rotating workforce scheduling problem as first-order formulas with certain background theories
- Solving benchmark instances from literature with different state-of-the-art SMT-solvers and evaluating their performance on two modeling approaches
- Comparing this new approach to existing techniques from literature and evaluating its competitiveness

The first modeling approach uses linear arithmetical operations and equalities to formulate the constraints of the problem. Every work day of the schedule is represented as an integer variable which can be assigned to one of the defined shifts. The second approach was motivated by the assumption that fixed-size bitvectors and the use of machine arithmetical operations allow a more compact formulation of the problem and hence a faster solving time. This compact formulation relies on the fact that in contrast to the linear arithmetic approach, where every day

of the schedule was represented by a separate variable, the bitvector method models the whole schedule as one bitvector for each shift and thus reduces the number of variables to the number of required shifts.

Although encoding the problem with bitvectors and low-level machine arithmetic was a challenging task due to the complexity of the constraints, the assumption of the approach's superiority was indeed confirmed by the comparison of the experimental runtime results of the two modeling techniques. Since the bitvector approach requires less variables and assertions to formulate the problem, it achieves faster solving times. All tested SMT-solvers were able to increase the number of solved instances in contrast to the first approach.

The competitiveness of solving the problem with SMT-solvers was verified by the comparison to other methods from literature. Although heuristic approaches provide a better runtime on most instances and are also able to solve more problem instances, I concentrated on the comparison of SMT-solving to other exact methods. The results show that SMT-solving is a well performing alternative to these approaches, because it outperformed the other solving techniques on many instances and is able to provide a solution in a reasonable time frame, for some instances where the other methods fail to do so.

6.2 Future Work

This work has shown that SMT is an efficient approach for solving the rotating workforce scheduling problem. Nonetheless, there are some topics that were not addressed in this thesis which would be interesting to investigate in future work.

Other Encodings It is still interesting to investigate whether other first-order theories supported by modern SMT-solvers would allow a more efficient formulation of the problem.

Incremental Solving Many of the tested SMT-solvers implement a technique called *incremental* solving, which allows solving SMT formulas in sequence, reusing information from previous runs to avoid restarts. It would be interesting to investigate, if this technique can be used to decompose the search process and thereby reduce the search space for each step. Furthermore, the user can be involved in the search process similar to the approach in FCS [74].

Optimization Problems There has already been some research to use SMT-solvers for optimization problems [79]. By incorporating optimization into the encodings, SMT-solvers could create rotating schedules that represent the best solution with respect to some soft-constraints like optimal number of weekends.

Bibliography

[1] Hesham K. Alfares. An efficient two-phase algorithm for cyclic days-off scheduling. *Computers & Operations Research*, 25(11):913 – 923, 1998. ISSN 0305-0548.

[2] Hesham K Alfares. Survey, categorization, and comparison of recent tour scheduling literature. *Annals of Operations Research*, 127(1-4):145–175, 2004.

[3] Carlos Ansótegui, Miquel Bofill, Miquel Palahí, Josep Suy, and Mateu Villaret. Satisfiability modulo theories: An efficient approach for the resource-constrained project scheduling problem. In *Proceedings of SARA - Symposium on Abstraction, Reformulation and Approximation*. AAAI, 2011.

[4] Alessandro Armando, Claudio Castellini, and Enrico Giunchiglia. SAT-based procedures for temporal reasoning. In *Proceedings of ECP - European Conference on Planning*, volume 1809 of *Lecture Notes in Computer Science*, pages 97–108. Springer, 1999.

[5] Alessandro Armando, Claudio Castellini, Enrico Giunchiglia, and Marco Maratea. A SAT-based decision procedure for the boolean combination of difference constraints. In *Proceedings of Theory and Applications of Satisfiability Testing, Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2005.

[6] Júlíus Atlason, Marina A Epelman, and Shane G Henderson. Call center staffing with simulation and cutting plane methods. *Annals of Operations Research*, 127(1-4):333–358, 2004.

[7] Gilles Audemard, Piergiorgio Bertoli, Alessandro Cimatti, Artur Korniłowicz, and Roberto Sebastiani. A SAT based approach for solving formulas over boolean and linear mathematical propositions. In *Proceedings of Automated Deduction - CADE-18*, volume 2392 of *Lecture Notes in Computer Science*, pages 195–210. Springer, 2002.

[8] Turgut Aykin. Optimal shift scheduling with multiple break windows. *Management Science*, 42(4):591–602, 1996.

[9] Kenneth R Baker. Workforce allocation in cyclical scheduling problems: A survey. *Operational Research Quarterly*, pages 155–167, 1976.

[10] Nagraj Balakrishnan and Richard T Wong. A network model for the rotating workforce scheduling problem. *Networks*, 20(1):25–42, 1990.

[11] Thomas Ball and Sriram K Rajamani. The SLAM project: debugging system software via static analysis. *ACM SIGPLAN Notices*, 37(1):1–3, 2002.

[12] Thomas Ball, Byron Cook, Shuvendu Lahiri, and Lintao Zhang. Zapato: Automatic theorem proving for predicate abstraction refinement. In *Proceedings of Computer Aided Verification*, pages 373–374. Springer, 2004.

[13] Clark Barrett and Sergey Berezin. CVC Lite: A new implementation of the cooperating validity checker. In *Proceedings of Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 515–518. Springer, 2004.

[14] Clark Barrett and Cesare Tinelli. CVC3. In *Proceedings of Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302. Springer, 2007.

[15] Clark Barrett, David Dill, and Aaron Stump. Checking satisfiability of first-order formulas by incremental translation to SAT. In *Proceedings of Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 236–249. Springer, 2002.

[16] Clark Barrett, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Splitting on demand in SAT modulo theories. In *Proceedings of Logic for Programming, Artificial Intelligence, and Reasoning*, volume 4246 of *Lecture Notes in Computer Science*, pages 512–526. Springer, 2006.

[17] Clark Barrett, Roberto Sebastiani, Sanjit A Seshia, and Cesare Tinelli. Satisfiability modulo theories. *Handbook of Satisfiability*, 185:825–885, 2009.

[18] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB standard: Version 2.0. In *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, England)*, volume 13, 2010.

[19] Clark Barrett, Christopher L Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *Proceedings of Computer Aided Verification*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011.

[20] Clark Wayne Barrett. *Checking validity of quantifier-free formulas in combinations of first-order theories*. PhD thesis, Stanford University, 2002.

[21] Andreas Beer, Johannes Gärtner, Nysret Musliu, Werner Schafhauser, and Wolfgang Slany. Scheduling breaks in shift plans for call centers. In *Proceedings of PATAT - The 7th International Conference on the Practice and Theory of Automated Timetabling*, 2008.

[22] Andreas Beer, Johannes Gärtner, Nysret Musliu, Werner Schafhauser, and Wolfgang Slany. An AI-based break-scheduling system for supervisory personnel. *Intelligent Systems, IEEE*, 25(2):60–73, 2010.

[23] Nikolaj Bjørner and Leonardo De Moura. Z310: Applications, enablers, challenges and directions. *Constraints in Formal Verification, CFV*, 9, 2009.

[24] Miquel Bofill, Josep Suy, and Mateu Villaret. A system for solving constraint satisfaction problems with SMT. In *Theory and Applications of Satisfiability Testing – SAT 2010*, volume 6175 of *Lecture Notes in Computer Science*, pages 300–305. Springer, 2010.

[25] Robert Brummayer and Armin Biere. Boolector: An efficient SMT solver for bit-vectors and arrays. In *Proceedings of TACAS - Tools and Algorithms for the Construction and Analysis of Systems*, volume 5505 of *Lecture Notes in Computer Science*, pages 174–177. Springer, 2009.

[26] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzen, Alberto Griggio, Ziyad Hanna, Alexander Nadel, Amit Palti, and Roberto Sebastiani. A lazy and layered SMT(BV) solver for hard industrial verification problems. In *Proceedings of CAV - Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 547–560. Springer, 2007.

[27] Randal E Bryant, Daniel Kroening, Joël Ouaknine, Sanjit A Seshia, Ofer Strichman, and Bryan Brady. Deciding bit-vector arithmetic with abstraction. In *Proceedings of TACAS - Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *Lecture Notes in Computer Science*, pages 358–372. Springer, 2007.

[28] Edmund K Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *Journal of scheduling*, 7(6):441–499, 2004.

[29] B. Butler. Computerized manpower scheduling. Master's thesis, University of Alberta, Canada, 1978.

[30] Marco Cadoli, Toni Mancini, and Fabio Patrizi. SAT as an effective solving technology for constraint problems. In *Proceedings of ISMIS - Foundations of Intelligent Systems*, volume 4203 of *Lecture Notes in Computer Science*, pages 540–549. Springer, 2006.

[31] Peter Chan and Georges Weil. Cyclical staff scheduling using constraint logic programming. In *Proceedings of PATAT - The 3th International Conference on the Practice and Theory of Automated Timetabling*, volume 2079, pages 159–175. Springer, 2001.

[32] Hoong Chuin Lau. On the complexity of manpower shift scheduling. *Computers & operations research*, 23(1):93–102, 1996.

[33] Alessandro Cimatti, Alberto Griggio, Bastiaan Schaafsma, and Roberto Sebastiani. The MathSAT5 SMT solver. In *Proceedings of TACAS - Tools and Algorithms for the Construction and Analysis of Systems*, volume 7795 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2013.

[34] J. Daintith, V. Illingworth, and I.C. Pyle. *Oxford Dictionary of Computing*. Oxford paperback reference. Oxford University Press, 2004. ISBN 9780198608776.

[35] George B Dantzig. Letter to the editor - a comment on edie's traffic delays at toll booths. *Operations Research*, 2(3):339 – 341, 1954.

[36] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960.

[37] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[38] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Proceedings of TACAS - Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, chapter 24, pages 337–340. Springer, 2008.

[39] Leonardo de Moura and Nikolaj Bjørner. Satisfiability modulo theories: An appetizer. In *Proceedings of SBMF - Formal Methods: Foundations and Applications*, volume 5902 of *Lecture Notes in Computer Science*, pages 23–36. Springer, 2009.

[40] Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: Introduction and applications. *Communications of the ACM*, 54(9):69–77, 2011.

[41] Leonardo De Moura and Harald Rueß. An experimental evaluation of ground decision procedures. In *Proceedings of CAV - Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 162–174. Springer, Springer, 2004.

[42] Leonardo De Moura, Harald Rueß, and Maria Sorea. Lemmas on demand for satisfiability solvers. In *Proceedings of the Fifth International Symposium on the Theory and Applications of Satisfiability Testing*, 2002.

[43] Matthijs C. Dijkstra, Leo G. Kroon, Jo A.E.E. van Nunen, and Marc Salomon. A DSS for capacity planning of aircraft maintenance personnel. *International Journal of Production Economics*, 23(1):69–78, 1991. ISSN 0925-5273.

[44] Peter J Downey, Ravi Sethi, and Robert Endre Tarjan. Variations on the common subexpression problem. *Journal of the ACM (JACM)*, 27(4):758–771, 1980.

[45] Bruno Dutertre and Leonardo de Moura. The Yices SMT solver. Technical report, SRI International, 2006.

[46] Bruno Dutertre and Leonardo De Moura. A fast linear-arithmetic solver for DPLL(T). In *Proceedings of CAV - Computer Aided Verification*, volume 4144, pages 81–94. Springer, 2006.

[47] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Proceedings of SAT - Theory and Applications of Satisfiability Testing*, volume 2919, pages 502–518. Springer, 2004.

[48] Andreas T Ernst, Houyuan Jiang, Mohan Krishnamoorthy, Bowie Owens, and David Sier. An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research*, 127(1):21–144, 2004.

[49] Andreas T Ernst, Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. Staff scheduling and rostering: A review of applications, methods and models. *European journal of operational research*, 153(1):3–27, 2004.

[50] Jean-Christophe Filliâtre, Sam Owre, Harald Rueß, and Natarajan Shankar. ICS: Integrated canonizer and solver. In *Proceedings of CAV - Computer Aided Verification*, volume 2102, pages 246–249. Springer, 2001.

[51] Robert W Floyd. Assigning meanings to programs. *Mathematical aspects of computer science*, 19:19–32, 1967.

[52] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): Fast decision procedures. In *Proceedings of CAV - Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2004.

[53] Johannes Gärtner, Nysret Musliu, and Wolfgang Slany. Rota: a research project on algorithms for workforce scheduling and shift design optimization. *AI Communications*, 14(2): 83–92, 2001.

[54] Yeting Ge, Clark W. Barrett, and Cesare Tinelli. Solving quantified verification conditions using satisfiability modulo theories. *Annals of Mathematics and Artificial Intelligence*, 55 (1-2):101–122, 2009.

[55] Silvio Ghilardi, Enrica Nicolini, Silvio Ranise, and Daniele Zucchelli. Decision procedures for extensions of the theory of arrays. *Annals of Mathematics and Artificial Intelligence*, 50(3):231–254, 2007.

[56] Patrice Godefroid, Peli de Halleux, Aditya V Nori, Sriram K Rajamani, Wolfram Schulte, Nikolai Tillmann, and Michael Y Levin. Automating software testing using program analysis. *Software, IEEE*, 25(5):30–37, 2008.

[57] Richard W Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950.

[58] Nelson B Heller, J Thomas McEwen, and William W Stenzel. *Computerized scheduling of police manpower*. St. Louis Police Department, St. Louis, MO, 1973.

[59] Thomas Henzinger, Ranjit Jhala, Rupak Majumdar, and Gregoire Sutre. Software verification with BLAST. In *Proceedings of SPIN - Model Checking Software, 10th International SPIN Workshop*, volume 2648 of *Lecture Notes in Computer Science*, pages 235–239. Springer, 2003.

[60] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[61] Susmit Jha, Rhishikesh Limaye, and Sanjit Seshia. Beaver: Engineering an efficient smt solver for bit-vector arithmetic. In *Proceedings of CAV - Computer Aided Verification*, volume 5643 of *Lecture Notes in Computer Science*, pages 668–674. Springer, 2009.

[62] Henry Kautz. Deconstructing planning as satisfiability. In *Proceedings of AAAI - The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference*, number 2, pages 1524–1526. AAAI Press, 2006.

[63] James C King. Symbolic execution and program testing. *Communications of the ACM*, 19 (7):385–394, 1976.

[64] Sava Krstić and Amit Goel. Architecting solvers for SAT modulo theories: Nelson-oppen with dpll. In *Proceedings of FroCoS - Frontiers of Combining Systems, 6th International Symposium*, volume 4720 of *Lecture Notes in Computer Science*, pages 1–27. Springer, 2007.

[65] Shuvendu Lahiri and Shaz Qadeer. Back to the future: revisiting precise program verification using SMT solvers. In *Proceedings of POPL - Principles of Programming Languages*, volume 43, pages 171–182. ACM, 2008.

[66] G Laporte. The art and science of designing rotating schedules. *Journal of the Operational Research Society*, 50:1011–1017, 9 1999.

[67] Gilbert Laporte and Gilles Pesant. A general multi-shift scheduling system. *Journal of the Operational Research Society*, 55(11):1208–1217, 2004.

[68] Gilbert Laporte, Yves Nobert, and Jean Biron. Rotating schedules. *European journal of operational research*, 4(1):24–30, 1980.

[69] João P. Silva Marques and Karem A. Sakallah. GRASP – a new search algorithm for satisfiability. In *Proceedings of ICCAD - International Conference on Computer Aided Design*, pages 220–227, 1996.

[70] John McCarthy. Towards a mathematical science of computation. *Information processing*, 62:21–28, 1962.

[71] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of DAC - Design Automation Conference*, pages 530–535. ACM, 2001.

[72] Nysret Musliu. *Intelligent Search Methods for Workforce Scheduling: New Ideas and Practical Applications*. PhD thesis, Vienna Univercity of Technologie, 2001.

[73] Nysret Musliu. Heuristic methods for automatic rotating workforce scheduling. *International Journal of Computational Intelligence Research*, 2(4):309–326, 2006.

[74] Nysret Musliu, Johannes Gärtner, and Wolfgang Slany. Efficient generation of rotating workforce schedules. *Discrete Applied Mathematics*, 118(1-2):85–98, 2002.

[75] Nysret Musliu, Werner Schafhauser, and Magdalena Widl. A memetic algorithm for a break scheduling problem. In *Proceedings of the VIII Metaheuristic International Conference (MIC 2009), Hamburg, Germany*, 2009.

[76] Charles Gregory Nelson. *Techniques for program verification*. XEROX Research Center, 1981.

[77] Greg Nelson and Derek C Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1(2):245–257, 1979.

[78] Greg Nelson and Derek C Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM (JACM)*, 27(2):356–364, 1980.

[79] Robert Nieuwenhuis and Albert Oliveras. On SAT modulo theories and optimization problems. In *Proceedings of SAT - Theory and Applications of Satisfiability Testing*, volume 4121 of *Lecture Notes in Computer Science*, pages 156–169. Springer, 2006.

[80] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract davis-putnam-logemann-loveland procedure to dpll (t). *Journal of the ACM*, 53(6):937–977, 2006.

[81] Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. Challenges in satisfiability modulo theories. In *Proceedings of RTA - Term Rewriting and Applications*, volume 4533 of *Lecture Notes in Computer Science*, pages 2–18. Springer, 2007.

[82] Amir Pnueli, Yoav Rodeh, Ofer Shtrichman, and Michael Siegel. Deciding equality formulas by small domains instantiations. In *Proceedings of CAV - Computer Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 1999.

[83] Sanjit A Seshia, Shuvendu K Lahiri, and Randal E Bryant. A hybrid SAT-based decision procedure for separation logic with uninterpreted functions. In *Proceedings of DAC - Design Automation Conference*, pages 425–430. ACM, 2003.

[84] Hossein Sheini and Karem Sakallah. From propositional satisfiability to satisfiability modulo theories. In *Proceedings of SAT - Theory and Applications of Satisfiability Testing*, volume 4121 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2006.

[85] Aaron Stump, Clark W Barrett, David L Dill, and Jeremy Levitt. A decision procedure for an extensional theory of arrays. In *Proceedings of LICS - Logic in Computer Science*, pages 29–37. IEEE Computer Society, 2001.

[86] Aaron Stump, Clark W Barrett, and David L Dill. CVC: A cooperating validity checker. In *Proceedings of CAV - Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 500–504. Springer, 2002.

[87] Josep Suy Franch. *A satisfiability modulo theories approach to constraint programming*. PhD thesis, Universitat de Girona, 2012.

[88] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear CSP into SAT. *Constraints*, 14(2):254–272, 2009.

[89] Nikolai Tillmann and Wolfram Schulte. Unit tests reloaded: Parameterized unit testing with symbolic execution. *Software, IEEE*, 23(4):38–47, 2006.

[90] Markus Triska and Nysret Musliu. A constraint programming application for rotating workforce scheduling. In *Developing Concepts in Applied Intelligence*, volume 363 of *Studies in Computational Intelligence*, pages 83–88. Springer Berlin / Heidelberg, 2011. ISBN 978-3-642-21331-1.

[91] Jorne Van den Bergh, Jeroen Beliën, Philippe De Bruecker, Erik Demeulemeester, and Liesje De Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367–385, 2013.

[92] Toby Walsh. SAT v CSP. In *Proceedings of CP - Principles and Practice of Constraint Programming*, volume 1894 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2000.

[93] Peter Wegner. A technique for counting ones in a binary computer. *Commun. ACM*, 3(5): 322, 1960. ISSN 0001-0782.

[94] Magdalena Widl and Nysret Musliu. An improved memetic algorithm for break scheduling. In *Proceedings of Hybrid Metaheuristics - 7th International Workshop*, Lecture Notes in Computer Science, pages 133–147. Springer, 2010.

[95] Hantao Zhang, Dapeng Li, and Haiou Shen. A SAT based scheduler for tournament schedules. In *Proceedings of SAT - Theory and Applications of Satisfiability Testing*, 2004.