FernUniversität in Hagen
Faculty of Mathematics and Computer Science

Artificial Intelligence Group

First complete compilation-founded implementation of distance-based belief change

Master's Thesis

in partial fulfillment of the requirements for
the degree of Master of Science (M.Sc.)
in Praktische Informatik

submitted by
Julia Hayat

First examiner:	Dr. Jandson Santos Ribeiro Santos Artificial Intelligence Group
Advisor:	Dr. Jandson Santos Ribeiro Santos Artificial Intelligence Group

Statement

Ich erkläre, dass ich die Masterarbeit selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für enthaltene Zeichnungen, Skizzen oder graphische Darstellungen. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Arbeit nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate geprüft werden kann und ausschließlich für Prüfungszwecke gespeichert wird.

	Yes	No
I agree to have this thesis published in the library.	☒	☐
I agree to have this thesis published on the webpage of the artificial intelligence group.	☒	☐
The thesis text is available under a Creative Commons License (CC BY-SA 4.0).	☒	☐
The source code is available under a GNU General Public License (GPLv3).	☒	☐
The collected data is available under a Creative Commons License (CC BY-SA 4.0).	☒	☐

Arsberg, 12.02.2024 ..... Hayat .....

(Place, Date) (Signature)

Abstract

The study of belief change addresses the question of how a rational agent should adjust its beliefs, when facing new, potentially contradictory, information. Following a belief change operation, there are usually two kinds of checks that are of interest: model and inference checks. The former are concerned with determining whether the new belief state holds for particular interpretations, whereas the latter investigate whether it entails a particular propositional formula. To avoid arbitrariness and ensure reproducibility, several belief change operators have been proposed in the literature. While these operators have been studied extensively on theoretical grounds, there are only few academic works on practical implementations. One reason is likely the high computational complexity of many operators with regards to inference and model checks, which are usually located on the first or second level of the polynomial hierarchy. We address this shortcoming by implementing and evaluating a compilation-based belief change application that makes use of Boolean Satisfiability Solving (SAT), Answer Set Programming (ASP), and Integer Linear Programming (ILP). The application supports inference and model checks for four distinct distance-based revision and contraction operators. Given a belief change instance, it firstly determines the minimum distance via partial Maximum Satisfiability Solving (partial MaxSAT), ASP, or ILP optimization encodings and subsequently compiles the new belief state that results from the change operation into a SAT, ASP, or ILP encoding. The resulting encoding can then be leveraged for inference and model checks on the new belief state, turning these checks into feasibility problems. In an extensive evaluation we compare the usage of the different encodings in terms of runtime. The results show that for the determination of the minimum distance, our partial MaxSAT-based optimization encoding schemes perform best for each of the four supported change operators. With regards to inference and model checks we obtain that both the SAT- and the ASP-based encodings outperform their ILP counterparts. Moreover, the ASP-based encodings outperform the SAT-based ones in inference checks, whereas with regards to model checks the SAT-based encodings perform best. A comparison with a naive baseline implementation further concludes that our compilation-based approach is clearly superior to the naive approach, irrespective of the used encoding schemes. Our results demonstrate that compilation-based approaches to belief change constitute a viable and promising way of implementing inference and model checks and therefore deserve more research attention within the academic field of belief change.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

The study of belief change addresses the question of how a rational agent should adjust its beliefs, when facing a flux of new, potentially contradictory, information. Originally emerging from within the field of philosophy, belief change was first picked up by computer scientists of the database research field, studying the updating of databases with new incoming data. More recently, the subject of belief change has also gained considerable attention from the Artificial Intelligence (AI) research community in the context of artificial rational agents operating in dynamic environments [FH18].

Two of the most important belief change operations are belief revision, defined as the addition of a new belief to the current belief state while ensuring consistency, and belief contraction, the renunciation of a belief. In the study of belief change it has become a common practice to represent beliefs by propositional formulae and sets of beliefs as conjunctions of such formulae. In propositional logic an interpretation with respect to a set of atoms is a truth assignment to each of these atoms, and a model of a formula α is an interpretation of the atoms in α such that α evaluates to true. Accordingly, a model of a set of beliefs, represented by a conjunction, can be regarded as a possible interpretation of the world that is in accordance with the held beliefs. Even though in some cases, particularly when incoming information does not collide with already held beliefs, belief change can be relatively simple, it is generally a non-trivial task as there is usually more than one possible solution. Besides, held beliefs are oftentimes deeply interwoven, making it difficult to remove specific beliefs or to resolve inconsistencies emanating from newly added beliefs. As an example, let there be an agent with the beliefs p and $p \rightarrow q$, where p and q represent propositional atoms. The agent then receives the new information $\neg q$. Simply adding this new belief to the belief state does not suffice, since q would still be implied. In order to remove the indirect belief q, the agent can either give up the belief p or the belief $p \rightarrow q$. To address these issues, several distinct belief change operators have been proposed in the literature. Operators provide rules for belief change operations and consequently make belief change reproducible, rather than arbitrary. Existing operators can be divided into different families of operators sharing a common idea or strategy. One such family is the family of distance-based operators. Operators of this kind conduct belief change by selecting the new belief state's models on the basis of distance between the old belief state's models and those of the new incoming information. Whereas for some application domains, such as iterated change, the exact form of the new belief state (be it formulae or models) is important, for others it is not the precise new belief state that is of interest but rather whether the new belief state implies a certain belief (inference check) or holds in a certain interpretation of the world (model check) [HV91b].

While it is commonly agreed in the field of computer science that belief change should be addressed from both a theoretical and practical perspective, the current research situation is characterized by a clear imbalance to the former. This lack of practical research undertakings can be - at least partly - attributed to the high computational complexity of inference and model checks for many of the belief change operators so

far proposed on theoretical grounds [EG92] [LS01]. One way of handling this high complexity in applications is by employing a compilation-based approach. The concept of compilation builds on the idea of dividing intractable problems into two parts, of which one is known beforehand (fixed part) and the other is given at execution time (varying part). The fixed part is preprocessed and encoded into a new data structure. This process is referred to as compilation. The generated encoding is later leveraged in the varying part's solving. Applied to distance-based belief change, it is conceivable to think of the revision or contraction as the fixed part and of the minimum distance determination, which is essentially an optimization problem, as the preprocessing, whose solution can be incorporated into an encoding of the new belief state, which can subsequently be used for inference and model checks, constituting the varying part. The efficiency of compilation approaches draws from the fact that the compilation process needs to be executed only once for an infinite number of problems sharing a common fixed part. Accordingly, the costs of each inference or model check consist of the costs of the compilation process of the corresponding belief operation, divided by the total number of inference and model checks, plus the costs of the actual check. Evidently, the higher the number of such checks is, the more distributed the costs of compilation are, and the more efficient is the overall approach. Even though it might not be the case always, there are many domains, e.g. within the field of AI, where knowledge bases need to be queried several times, making investigations into the feasibility of using compilation for belief change operations worthwhile. Konieczny et al. [KLM17] suggested such a compilation approach for the family of topic-decomposable distance-based revision operators. In their work they proposed a partial MaxSAT (partial Maximum Satisfiability) encoding scheme for determining the minimum distance with a single call to a solver as well as a SAT (Boolean Satisfiability Solving) encoding scheme of the revised belief base that is query-equivalent to the new belief base and thus turns the inference problem into a feasibility problem. They subsequently evaluated the performance of their compilation process, showing that their suggested approach can successfully compile non-trivial belief revision instances within reasonable time into query-equivalent SAT encodings. The main downside of their work, however, is the lack of an investigation into the performance of their belief revision encodings with regards to inference and model checks.

This thesis aims to address the current practical research shortcomings in the field of belief change by implementing and evaluating a compilation-founded belief change application for inference and model checks, supporting four distinct distance-based operators: the revision operators suggested by Dalal [Dal88] and Satoh [Sat88] and their belief contraction counterparts. In the spirit of Konieczny et al. [KLM17], the implementation follows a compilation-based approach by containing a preprocessing step, that carries out the minimum distance determination by generating preprocessing encodings and passing them to corresponding solvers, followed by the generation of encodings of the post-revision or post-contraction belief states for subsequent model and inference checks. For each supported belief change operator, three distinct preprocessing encoding schemes (partial MaxSAT, ASP, ILP) and three distinct belief

change encoding schemes (SAT, ASP, ILP) are proposed and implemented. To identify the most efficient preprocessing and belief change encoding schemes for each operator, a thorough evaluation of the compilation phase and the belief change encodings' performance in model and inference checks is carried out. Moreover, the application is compared to a naive baseline implementation. The motivation behind implementing and evaluating ASP- and ILP-based approaches in addition to partial MaxSAT/SAT, suggested by Konieczny et al., lies in the fact that both are established technologies for solving feasibility as well as optimization problems and therefore constitute promising candidates for both the preprocessing step and the belief change compilation.

Due to Konieczny et al.'s sole focus on the compilation process, as well as their restriction to revision operators and partial MaxSAT/SAT technologies, this thesis can be regarded as a complement to their work in a three-fold way:

1. by expanding their compilation-based approach to address additional operators, including contraction operators
2. by making use of alternate technologies (ASP, ILP)
3. by evaluating the performance of belief change encodings, including the SAT-based one suggested by Konieczny et al., with regards to inference and model checks

Moreover, given the lack of further research on compilation-based approaches to belief change, our application constitutes the first complete compilation-founded implementation of belief change, in that it supports not only the compilation process, but also inference and model checks on the generated encodings.

The remaining part of this work is organized as follows. In Chapter 2 important terms and concepts are defined and all relevant theoretical aspects of belief change presented. Besides, Dalal's and Satoh's belief change operators are introduced and the technologies used in our application are described. The last section of the chapter elaborates further on the shortcomings of current practical research in the area of computational belief change, stressing the relevance of our work. This is followed by a detailed description of our application in Chapter 3, including illustrations of the implemented algorithms, and definitions and proofs of the encoding schemes. Chapter 4 contains the evaluation of our application, which consists of a runtime analysis of the compilation phase, wherein the different preprocessing encoding schemes are compared, as well as a comparison of the proposed belief change encoding schemes with regards to inference and model checks. The chapter starts with a detailed description of the experimental setup and the implemented naive approach and concludes with the presentation and discussion of the obtained results. Lastly, in Chapter 5 we summarize our findings, point out shortcomings of our evaluation and conclude the thesis by suggesting directives for future research activities.

2 Background

The aim of this chapter is to introduce the most important aspects of belief change and the belief change operators proposed by Dalal and Satoh. The chapter begins with the introduction of a set of formal preliminaries that are needed for a thorough understanding of the former.

2.1 Formal preliminaries

A *(propositional) atom* is an expression that can be either *true* or *false* and that does not contain any logical connectives. Throughout this paper, unless otherwise stated, we represent atoms by lower case Latin letters, e.g.

p: *The sky is blue.*
q: *Today is Monday.*

The truth values *true* and *false* of propositional logic are sometimes denoted by 1 and 0, respectively.

A *signature* is a finite set of atoms.

A *propositional formula* is a combination of one or more atoms, using the usual logical connectives $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction), $\rightarrow$ (implication) and $\longleftrightarrow$ (equivalence). Given a particular truth assignment to the atoms contained in a formula, the formula can be either *true* or *false*. Unless otherwise stated, we denote propositional formulae by lower case Greek letters, e.g.

α: $\neg p$
β: $p \vee q$

A *literal* is an atom or the negation of an atom.

We denote the set of all atoms occurring in a given propositional formula α, i.e. its *signature*, by $Var(\alpha)$.

A *propositional logic language* L consists of the set of all propositional formulae that can be formed from the language's signature $Var(L)$.

An *interpretation* of a signature is a truth assignment to each atom of the signature and can be thought of as a bit vector. As an example, given the signature $\{\alpha, \beta\}$, the corresponding set of possible interpretations is $\{10, 01, 11, 00\}$. Assuming that a given signature consists of n atoms with $n > 0$, there consequently exist 2^n different interpretations for the given signature. An interpretation of a propositional formula γ is an interpretation of $Var(\gamma)$.

An interpretation M that makes a formula α evaluate to *true* is called a *model* of α, also denoted by the expression $M \models \alpha$. The set of all models of α is expressed by $Mod(\alpha)$.

In this work we use the notation $value(I, x)$ to represent the truth value of the atom x in a given interpretation I. Moreover, we use $proj(I, V)$ to indicate the projection of a bit vector (interpretation) I over those variables that are contained in the set V. The projection is an interpretation of the signature V. As an example, let us assume a formula $(\alpha \vee \beta) \wedge \gamma$ and a model $M = 101$. Let $V = \{\beta, \gamma\}$, then $proj(M, V) = 01$ and $value(M, \gamma) = 1$. Furthermore, $proj(N, V) = \{proj(I_1, V), ..., proj(I_n, V)\}$ for a given set of interpretations $N = \{I_1, ..., I_n\}$. Note that $proj(N, V)$ is a set and hence does not contain any duplicates.

The expression $\alpha \models \beta$ states that satisfaction of α entails satisfaction of β, hence $Mod(\alpha) \subseteq Mod(\beta)$. Moreover, two formulae γ and μ are *logically equivalent*, also expressed by $\gamma \equiv \mu$, when $Mod(\gamma) = Mod(\mu)$.

Given three propositional formulae α, β and γ with $Var(\gamma) \subseteq Var(\alpha) \subseteq Var(\beta)$, formulae α and β are *query-equivalent* iff $\beta \models \gamma$ whenever $\alpha \models \gamma$ and $\beta \not\models \gamma$ whenever $\alpha \not\models \gamma$. If α and β are query-equivalent, they are also *equisatisfiable*, i.e. both formulae are either satisfiable or unsatisfiable at the same time.

A propositional formula in **C**onjunctive **N**ormal **F**orm (**CNF**) is a conjunction of one or more disjunctions, where the disjunctions's disjuncts are literals. Such disjunctions are commonly referred to as *clauses*. We denote the set of clauses of a CNF formula β by $C(\beta)$.

Every propositional formula can be converted to CNF. The naive approach is to repeatedly apply the usual boolean transformation rules on the initial non-CNF formula until a CNF formula is obtained. Despite the guaranteed success of this approach, its downside is that it can result in an exponential increase in the number of clauses. Hence, more efficient methods have been proposed in the literature. One of them is the Tseitin transformation [Tse68], which results in a CNF formula that is equisatisfiable to the initial formula with an only linear increase in the number of clauses (linear in the size of the original formula). The Tseitin transformation approach creates new auxiliary variables, which are part of the resulting CNF formula. The number of these newly introduced variables is also linear in the size of the original formula.

In this paper, a *belief* is represented by a propositional formula and an arbitrary set of beliefs, also called a *belief base*, conforms to the logical conjunction of its members.

In contrast to belief bases, a *belief set* or *belief theory* A in a propositional logic

language L is a deductively closed set of beliefs, i.e. $A = Cn(A)$, where $Cn()$ is a consequence operator defined in the following way: $Cn(X) = \{\alpha \in L \mid X \models \alpha\}$

2.2 Belief change

Belief change is concerned with the incorporation of new beliefs or information into a rational agent's existing belief state. While originally studied from a normative point of view in the philosophical community, the subject has been addressed by researchers from several distinct academic fields. Throughout the last decades belief change has received particular attention from researchers in the field of computer science. Examples of contexts, where insights gained from the study of belief change are of high relevance to computer scientists, are database updating and cognitive robotics [DLRT05].

Perhaps the most famous contribution to the subject is the AGM framework established in several papers by Carlos Alchourrón, Peter Gärdenfors and David Makinson [AGM85] [Gä88] [GM88]. The AGM framework, which was named after its authors, is based on the concept of belief sets and defines three distinct change operations executable on such sets - expansion, revision and contraction. It should be noted, that the framework assumes any kind of Tarskian logic language and was hence not intended specifically for the propositional logic case. However, since this paper is targeting propositional logic, the following description of the framework's concepts is based on the assumption of a propositional context.

Expansion is defined as the addition of a new belief to the existing belief set, which is then closed under logical consequence.

Definition 2.1. *Let K be a consistent belief set and μ a satisfiable proposition. If $K + \mu$ denotes the expansion of K by μ, then $K + \mu = Cn(K \cup \{\mu\})$ [AGM85].*

Contraction, on the other hand, is the removal of a specific belief from the belief set. It corresponds to switching from believing in a certain proposition to taking a neutral stance on the same - neither believing it nor its negation.

Definition 2.2. *A contraction operator $\dot{-}$ is a function mapping a set of beliefs K to a subset of K, that no longer entails the belief μ, that is to be removed [Han91]: $(K \dot{-} \mu) \subseteq K and (K \dot{-} \mu) \nvDash \mu$*

Revision $\dot{+}$ extends expansion $+$ by requiring the updated belief set to be logically consistent and can be defined in terms of contraction via the *Levi Identity*.

Definition 2.3. *Let K be a consistent belief set and μ a satisfiable proposition. If $K \dot{+} \mu$ denotes the revision of K by μ, then $K \dot{+} \mu = Cn((K \dot{-} \neg\mu) \cup \{\mu\})$ [AGM85].*

Along with definitions, the AGM framework also provides an axiomatic characterization for each proposed type of operation, one of the guiding principles being the notion of minimal change, which describes the attempt to keep as many of the previous beliefs as possible. The AGM postulates for belief revision are as follows, where K denotes a belief set and μ and ϕ are propositions [AGM85]:

($\dot{+}1$) $K \dot{+} \mu$ is a belief set (closure)
($\dot{+}2$) $\mu \in K \dot{+} \mu$ (success)
($\dot{+}3$) $K \dot{+} \mu \subseteq K + \mu$ (inclusion)
($\dot{+}4$) If $\neg\mu \notin K$, then $K \dot{+} \mu = K + \mu$ (vacuity)
($\dot{+}5$) $K \dot{+} \mu$ is inconsistent only if μ is inconsistent (consistency)
($\dot{+}6$) If $\mu \equiv \phi$, then $K \dot{+} \mu = K \dot{+} \phi$ (preservation)
($\dot{+}7$) $K \dot{+} (\mu \wedge \phi) \subseteq (K \dot{+} \mu) + \phi$
($\dot{+}8$) If $\neg\phi \notin K \dot{+} \mu$, then $(K \dot{+} \mu) + \phi \subseteq K \dot{+} (\mu \wedge \phi)$

For belief contraction, the AGM postulates are the following [AGM85]:

($\dot{-}1$) $K \dot{-} \mu$ is a belief set (closure)
($\dot{-}2$) $K \dot{-} \mu \subseteq K$ (inclusion)
($\dot{-}3$) If $\mu \notin K$, then $K \dot{-} \mu = K$ (vacuity)
($\dot{-}4$) If $\not\models \mu$, then $\mu \notin K \dot{-} \mu$ (success)
($\dot{-}5$) If $\mu \equiv \phi$, then $K \dot{-} \mu = K \dot{-} \phi$ (preservation)
($\dot{-}6$) If $\mu \in K$, then $K \subseteq (K \dot{-} \mu) + \mu$ (recovery)
($\dot{-}7$) $(K \dot{-} \mu) \cap (K \dot{-} \phi) \subseteq K \dot{-} (\mu \wedge \phi)$
($\dot{-}8$) If $\mu \notin K \dot{-} (\mu \wedge \phi)$, then $K \dot{-} (\mu \wedge \phi) \subseteq K \dot{-} \mu$

($\dot{+}1$) - ($\dot{+}6$) and ($\dot{-}1$) - ($\dot{-}6$) are called *basic revision postulates* and *basic contraction postulates*, respectively, whereas ($\dot{+}7$) and ($\dot{+}8$), as well as ($\dot{-}7$) and ($\dot{-}8$), are *supplementary postulates*, regulating the behaviour of belief change operators with regards to conjunctions [AGM85]. By defining postulates for belief change operations, the AGM framework has essentially laid out the benchmarks for evaluating change operators.

In addition to the AGM framework, many alternate approaches to belief change - some of them as a direct consequence of criticism towards AGM concepts - have been proposed, including alternate representations of belief states (belief bases), postulates for operations on those [KM91] [CKM17], and new types of change operations, such as update [KM92] or iterated change [DP97].

2.3 Belief bases

A common criticism of the AGM framework is its reliance on belief sets and the corresponding assumption, that the agent is a perfect logical reasoner. Belief sets do not distinguish between actually held beliefs and those that were merely derived [Neb89]. If an agent has the beliefs '*Earth is spherical*' and '*If earth is spherical, the equator is hotter than the poles*', then, due to the consequence operator, it also believes that '*The equator is hotter than the poles*'. If at a later point, the agent becomes undecided with regards to the shape of the earth and thus relinquishes the former belief that '*Earth is spherical*' - corresponding to the operation of contraction - the derived belief '*The equator is hotter than the poles*' might nevertheless remain in the belief set. Intuitively, this implicit belief should have no standing of its own and should be discarded once the belief it was

derived from is removed.

Apart from failing to make the important distinction between explicit and implicit beliefs, belief sets are a mathematical idealisation useful to the theoretical study of belief change, but turn out to be a hindrance in computational approaches. The reason for this lies in the difficulty of computing the logical closure of a set of propositions, combined with time and memory limitations of real-world agents.

Part of the criticism of belief sets was the introduction of belief bases. Belief bases contain an agent's explicit beliefs and are not closed under logical consequence. While the agent is still logically committed to all propositions that can be derived from its explicit beliefs, change operations are conducted on the belief base. As a consequence, implicit beliefs are relinquished as soon as they lose their support in the belief base [Han99]. A further advantage of representing beliefs as belief bases, especially finite ones, is that they allow for a computational implementation of belief change by no longer requiring the computation of logical closure.

Because of the AGM postulates' restriction to operations on belief sets, Katsuno and Mendelzon [KM91] have elaborated postulates for belief base revision and described, how these can be translated into the corresponding AGM postulates. Let κ and λ be consistent belief bases (note that κ and λ each correspond to a conjunction of their set members) and μ and ϕ be satisfiable propositional formulae representing normal statements. The belief base revision postulates proposed by Katsuno and Mendelzon can then be characterized as:

(R1) $\kappa \dot{+} \mu$ implies μ
(R2) If $\kappa \wedge \mu$ is satisfiable, then $\kappa \dot{+} \mu \equiv \kappa \wedge \mu$
(R3) If μ is satisfiable, then $\kappa \dot{+} \mu$ is also satisfiable
(R4) If $\kappa \equiv \lambda$ and $\mu \equiv \phi$, then $\kappa \dot{+} \mu \equiv \lambda \dot{+} \phi$
(R5) $(\kappa \dot{+} \mu) \wedge \phi$ implies $\kappa \dot{+} (\mu \wedge \phi)$
(R6) If $(\kappa \dot{+} \mu) \wedge \phi$ is satisfiable, then $\kappa \dot{+} (\mu \wedge \phi)$ implies $(\kappa \dot{+} \mu) \wedge \phi$

The authors have shown that, whenever a revision operator satisfies postulates (R1) - (R4) on belief bases, then its corresponding revision operator on belief sets satisfies the AGM postulates ($\dot{+}1$) - ($\dot{+}6$), with the condition being both necessary and sufficient. Furthermore, (R5) and (R6) correspond to ($\dot{+}7$) and ($\dot{+}8$) respectively [KM91].

Caridroit et al. [CKM17] have further extended the work of Katsuno and Mendelzon by proposing the following postulates for belief base contraction:

(C1) $\kappa \models \kappa \dot{-} \mu$
(C2) If $\kappa \not\models \mu$, then $\kappa \dot{-} \mu \models \kappa$
(C3) If $\kappa \dot{-} \mu \models \mu$, then μ is a tautology
(C4) $(\kappa \dot{-} \mu) \wedge \mu \models \kappa$
(C5) If $\kappa \equiv \lambda$ and $\mu \equiv \phi$, then $\kappa \dot{-} \mu \equiv \lambda \dot{-} \phi$
(C6) $\kappa \dot{-} (\mu \wedge \phi) \models (\kappa \dot{-} \mu) \vee (\kappa \dot{-} \phi)$
(C7) If $\kappa \dot{-} (\mu \wedge \phi) \not\models \mu$, then $\kappa \dot{-} \mu \models \kappa \dot{-} (\mu \wedge \phi)$

In their work they have demonstrated that the satisfaction of (C1) - (C7) by a contraction operator on belief bases is a necessary and sufficient condition for the satisfaction of AGM postulates ($\dot{-}1$) - ($\dot{-}8$) by its corresponding contraction operator on belief sets [CKM17].

2.4 Syntax-based vs. model-theoretic change operators

When analyzing belief base oriented change operators proposed in the literature, two main categories can be identified: syntax-based and model-theoretic operators. Whereas the former operate directly on the belief base's formulae, the latter consider its possible worlds, i.e. its models [HV91a]. Specific syntax-based operators were, among others, proposed by Fagin et al. [FUV83], Nebel [Neb89] and Ginsberg [Gin86]. Common to all of them is that each describes some kind of selection mechanism for choosing those sentences of the original belief base, that should be discarded, in order to obtain a consistent belief change result. An often-emphasized downside of syntax-based operators is the fact that conducting the same belief change operation on logically equivalent, yet syntactically different, belief bases does not necessarily lead to logically equivalent results. This phenomenon is commonly referred to as syntax-dependence and regarded as an undesirable property by several researchers in the belief change field, such as Dalal [Dal88], Winslett [Win88] and Katsuno and Mendelzon [KM89]. By contrast, model-theoretic operators examine the models of the belief base and of the new information and suggest a method for selecting those models, that will ultimately represent the updated belief state. They were proposed as part of an initiative to follow Dalal's principle of 'Irrelevance of Syntax', which states that, given two belief bases κ and λ, that are logically equivalent ($\kappa \equiv \lambda$), and given two logically equivalent incoming pieces of information α and β ($\alpha \equiv \beta$), the following has to hold: $\kappa \dot{+} \alpha \equiv \lambda \dot{+} \beta$ [Dal88].

Let $\kappa = \{p, q\}$ and $\lambda = \{p, p \longleftrightarrow q\}$ be two consistent belief bases. Even though syntactically distinct, they have the same logical meaning since $Mod(\kappa) = Mod(\lambda)$. Let us assume that a new belief $\alpha = \neg p$ has to be incorporated into both κ and λ, while maintaining consistency, which corresponds to the operation of revision. After applying a syntax-based revision operator, the resulting belief bases $\kappa \dot{+} \alpha$ and $\lambda \dot{+} \alpha$ might be distinct, as will be shown in the following. Consider an arbitrary syntax-based revision operator, that selects beliefs that are contradictory to the incoming belief and that should hence be given up, aiming to keep as many old beliefs as possible. Applying this operator results in the revised belief bases $\kappa \dot{+} \alpha = \{\neg p, q\}$ and $\lambda \dot{+} \alpha = \{\neg p, p \longleftrightarrow q\}$ with $Mod(\kappa \dot{+} \alpha) \neq Mod(\lambda \dot{+} \alpha)$. On the other hand, applying any kind of model-theoretic operator results in $Mod(\kappa \dot{+} \alpha) = Mod(\lambda \dot{+} \alpha)$ due to $Mod(\kappa) = Mod(\lambda)$. In the subsequent two sections, two specific model-theoretic approaches - the revision operators proposed by Dalal [Dal88] and Satoh [Sat88] - are introduced.

2.5 Dalal's revision operator

One of the most well-known revision operators for belief bases is the model-based operator proposed by Dalal in his 1988 paper 'Investigations Into a Theory of Knowledge Base Revision' [Dal88]. Apart from following the principle of *Irrelevance of Syntax*, Dalal's operator also abides by his principle of *Persistence of Prior Knowledge*, which conforms to the idea of minimal change in the old knowledge base. In order to quantify change, the Hamming distance, which is a metric for comparing bit vectors of equal size, is used. According to this distance, the less bits differ between two vectors, the closer the vectors are to each other. Dalal uses the Hamming metric for quantifying distance between propositional interpretations, which can be represented by bit vectors as already stated in Section 2.1.

Definition 2.4. *Given two interpretations X and Y for a given propositional signature, the number of atoms, that differ in terms of their truth assignments in X and Y, is called the Dalal distance between X and Y and is denoted by $d_D(X, Y)$ [Dal88].*

As an example, given a signature $\{p, q\}$, the models of the belief base $\kappa = \{p \vee q\}$ can be represented by the bit vectors 10, 01 and 11, hence $Mod(\kappa) = \{10, 01, 11\}$. The Dalal distance between models 10 and 01 is then 2, whereas the models 01 and 11 have a Dalal distance of 1. Analogously, the Dalal distance between two identical interpretations is 0.

The revision operator proposed by Dalal first determines the Dalal distance between each model of the belief base formula and each model of the revision formula. The smallest of all determined Dalal distance values is then called *minimum Dalal distance* and is denoted by $d_{min}(\alpha, \beta)$ for a belief base formula α and a revision formula β. Dalal's operator finally selects exactly those models of the revision formula as the models of the new belief base, that have the determined minimum Dalal distance to at least one model of the belief base formula. Definition 2.5 shows a formal characterization of the operator.

Definition 2.5. *Given a belief base κ and a revision formula μ, Dalal's revision operator $\dot{+}_D$ selects the models of the belief base obtained from a revision operation as follows [Dal88]*

$$Mod(\kappa \dot{+}_D \mu) = \{M \mid M \in Mod(\mu),\ \exists\, M_\kappa \in Mod(\kappa) \\ s.t.\ d_D(M_\kappa, M) = d_{min}(\kappa, \mu)\}$$

where $d_{min}(\kappa, \mu) = min\{d_D(M_\kappa, M_\mu) \mid M_\kappa \in Mod(\kappa)\ and\ M_\mu \in Mod(\mu)\}$.

The operator satisfies all of Katsuno and Mendelzon's belief base revision postulates and accordingly all AGM revision postulates [KM91]. The below example shows its execution on specific belief base and revision formulae.

Example 2.1. *Let $\kappa = a \wedge (a \vee d) \wedge b \wedge (\neg b \vee c)$ be a belief base formula and $\mu = \neg a \wedge \neg c$ a revision formula. We obtain $Mod(\kappa) = \{1110, 1111\}$ and $Mod(\mu) = \{0100, 0101, 0000, 0001\}$ for signature $\{a, b, c, d\}$. After comparing each model of κ to each model of μ we get the following set of*

Dalal distance values: $\{2, 3, 4\}$ *Since* $min\{2, 3, 4\} = 2$*, we obtain* $d_{min}(\kappa, \mu) = 2$*. Finally, as the only model comparisons with a Dalal distance of* 2 *are* $d_D(1110, 0100)$ *and* $d_D(1111, 0101)$*, we obtain* $Mod(\kappa \dot{+}_D \mu) = \{0100, 0101\}$*.*

In the trivial case, where κ and μ are consistent ($Mod(\kappa) \cap Mod(\mu) \neq \emptyset$), the minimum Dalal distance $d_{min}(\kappa, \mu)$ is of value 0 and as a consequence $Mod(\kappa \dot{+}_D \mu) = Mod(\kappa) \cap Mod(\mu)$. For a syntactic representation of the resulting belief base, one can choose any arbitrary set of formulae, whose models are identical to the ones obtained through Dalal's operator.

Due to Dalal's interpretation of minimal change as the closeness between models in terms of the Hamming distance, his operator can be classified under the category of cardinality-based operators. Another - quite similar - belief change operator of this category was also suggested by Forbus [For89].

2.6 Satoh's revision operator

In contrast to Dalal's cardinality-based revision operator, Satoh [Sat88] uses set inclusion as a means to quantify distance. His proposed revision operator, which he calls *Minimal Belief Revision*, can therefore be categorized as a set-inclusion- or set-containment-based operator. The operator suggested by Winslett in [Win88] can also be allocated to this category. Whereas Satoh originally defined his revision operator for the first-order case, it is approached from within a propositional context in the following. Before defining Satoh's operator, we first need to introduce the terms *difference set* and *minimal set* via the below formal definitions and examples.

Definition 2.6. *Given two interpretations X and Y of a signature, the set of propositional atoms, that have distinct truth assignments in X and Y, is called the difference set of X and Y, denoted by* $d_S(X, Y)$ *[Sat88].*

Example 2.2. *Let* V *be a signature with* $V = \{p, q\}$ *and* A*,* B *and* C *interpretations of* V *with* $A = 10$*,* $B = 00$ *and* $C = 11$*. Then* $d_S(A, B) = \{p\}$*,* $d_S(B, C) = \{p, q\}$ *and* $d_S(B, B) = \emptyset$*.*

Definition 2.7. *A member set* m *of a set* A *of sets is called a minimal set, if* A *contains no proper subset of* m*. The set containing all minimal sets of* A *is denoted by* $min_S(A)$*.*

Note that a set B is a proper subset of a set A, if A and B are not equal and B does not contain any elements, that are not in A. The empty set $\emptyset$ is a proper subset of every non-empty set.

Example 2.3. *Let* $A = \{\{p\}, \{q\}, \{p, q, r\}, \{q, r\}\}$ *be a set of sets, then* $min_S(A) = \{\{p\}, \{q\}\}$*. Further, let* $B = \{\{p\}, \{q\}, \{p, q, r\}, \{q, r\}, \emptyset\}$*, then* $min_S(B) = \{\emptyset\}$*.*

For Satoh the models of a revised belief base are precisely those models of the revision formula, for which there exists a model of the old belief base, such that the difference set between both models is a minimal set of the set of all difference sets. Formally, his operator can be defined as shown below.

Definition 2.8. *Given a belief base κ and a revision formula μ, Satoh's revision operator $\dot{+}_S$ selects the models of the belief base obtained from a revision operation as follows [Sat88] [KM91]*

$$Mod(\kappa \dot{+}_S \mu) = \{M \mid M \in Mod(\mu),\ \exists\, M_\kappa \in Mod(\kappa) \\ s.t.\ d_S(M_\kappa, M) \in min_S(D_S)\}$$

where $D_S = \{d_S(M_\kappa, M_\mu) \mid M_\kappa \in Mod(\kappa),\ M_\mu \in Mod(\mu)\}$.

Satoh's revision operator satisfies the belief base revision postulates (R1) - (R5), but not (R6) [KM91]. Accordingly, it's corresponding belief set revision operator satisfies AGM postulates ($\dot{+}1$) - ($\dot{+}7$), but not ($\dot{+}8$). Example 2.4 demonstrates the application of Satoh's revision operator on a specific belief revision instance.

Example 2.4. *Let $\kappa = a \wedge b \wedge (\neg b \vee c) \wedge (c \vee d) \wedge c$ be a belief base formula and $\mu = \neg c$ a revision formula. We obtain $Mod(\kappa) = \{1110, 1111\}$ and $Mod(\mu) = \{1100, 1101, 1000, 1001, 0100, 0101, 0000, 0001\}$ for signature $\{a, b, c, d\}$. Further, the set of difference sets obtained from comparing each model of κ to each model of μ is $D_S = \{\{c\}, \{c, d\}, \{b, c\}, \{b, c, d\}, \{a, c\}, \{a, c, d\}, \{a, b, c\}, \{a, b, c, d\}\}$ and $min_S(D_S) = \{\{c\}\}$. Since $d_S(1110, 1100) = \{c\}$ and $d_S(1111, 1101) = \{c\}$ are the only difference sets $\{c\}$, we obtain $Mod(\kappa \dot{+}_S \mu) = \{1100, 1101\}$.*

It follows from Definition 2.5 and Definition 2.8, that both Dalal and Satoh select models of the change formula, that are closest to models of the belief base, with the difference between the two approaches being their interpretation of distance. Note that in the case that κ and μ are consistent, i.e. $Mod(\kappa) \cap Mod(\mu) \neq \emptyset$, Dalal's and Satoh's revision operators lead to the same result. This is because in the case of consistency, the minimum Dalal distance is 0, hence $Mod(\kappa \dot{+}_D \mu) = Mod(\kappa) \cap Mod(\mu)$, and Satoh's set of minimal sets contains only the empty set $\emptyset$, thus $Mod(\kappa \dot{+}_S \mu) = Mod(\kappa) \cap Mod(\mu)$.

2.7 Harper Identity

Analogous to the *Levi Identity*, which was suggested by Levi [Lev77] in 1977 and which defines belief revision in terms of belief contraction, Harper [Har76] proposed a construction of contraction in terms of revision, referred to as *Harper Identity*. According to Harper, a belief q should be a member of a contraction $B \dot{-} p$ (B denoting a set of beliefs and p a belief) only if $q \in B$ and $q \in B \dot{+} \neg p$ hold.

Definition 2.9. *For each revision operator $\dot{+}$, there is an associated contraction operator $\dot{-}$, defined as follows*

$$B \dot{-} p = B \cap B \dot{+} \neg p$$

where B a set of beliefs and p a belief [Har76].

Applying the Harper Identity to the propositional logic case and belief bases, the subsequent relation between contraction and revision is obtained

$$\kappa \dot{-} \mu = \kappa \vee (\kappa \dot{+} \neg \mu) \tag{1}$$

where κ is a belief base (a conjunction of its member formulae) and μ a propositional formula. In terms of models Equation 1 can be translated to

$$Mod(\kappa \dot{-} \mu) = Mod(\kappa) \cup Mod(\kappa \dot{+} \neg\mu) \tag{2}$$

From equations 1 and 2 the nature of contractions becomes evident, namely the adoption of a neutral position towards a previously believed proposition.

The Harper Identity allows for the definition of the belief base contraction operators $\dot{-}_D$ and $\dot{-}_S$ from Dalal's and Satoh's revision operators.

Definition 2.10. *The contraction operator $\dot{-}_D$ obtained from Dalal's revision operator via the Harper identity is called Dalal's contraction operator and is defined as*

$$\kappa \dot{-}_D \mu = \kappa \vee (\kappa \dot{+}_D \neg\mu)$$

and

$$Mod(\kappa \dot{-}_D \mu) = Mod(\kappa) \cup Mod(\kappa \dot{+}_D \neg\mu)$$

Definition 2.11. *The contraction operator $\dot{-}_S$ obtained from Satoh's revision operator via the Harper Identity is called Satoh's contraction operator and is defined as*

$$\kappa \dot{-}_S \mu = \kappa \vee (\kappa \dot{+}_S \neg\mu)$$

and

$$Mod(\kappa \dot{-}_S \mu) = Mod(\kappa) \cup Mod(\kappa \dot{+}_S \neg\mu)$$

Caridroit et al. [CKM17] have elaborated, that a contraction operator, that was constructed from a revision operator via the Harper Identity, satisfies postulates (C1)-(C5) if the underlying revision operator satisfies postulates (R1)-(R4); and further, that (C6) is satisfied if (R5) is and (C7) is satisfied if (R6) is. Correspondingly, $\dot{-}_D$ fulfills contraction postulates (C1)-(C7) since (R1)-(R6) hold for $\dot{+}_D$ and $\dot{-}_S$ satisfies (C1)-(C6) since $\dot{+}_S$ abides by (R1)-(R5).

2.8 Technologies

Each of the subsequent three sections provides an introduction to one of the following technologies: SAT, ILP, and ASP.

2.8.1 Boolean satisfiability solving (SAT)

A SAT solver is a program, that is able to solve instances of the NP-complete *Boolean satisfiability problem*. [BHvMW09] Given a propositional formula, the *Boolean satisfiability problem* consists of determining whether there is a truth assignment to the contained variables (a model), so that the formula evaluates to true. Most SAT solvers do not only output, whether a formula is satisfiable, but also provide a possible solution, i.e. a model of the formula, in the case of satisfiability. A common input format to SAT

solvers is the DIMACS format, where variables are represented by consecutive natural numbers and the formula is in CNF (Conjunctive Normal Form). In each clause of the formula the logical connective $\vee$ between literals is removed and a '0' is attached to the end to mark the end of the clause. Usually, each line contains exactly one clause. Negation is represented by $-$. Further, comment lines begin with a 'c' and there is a so-called *problem line* of the form

p cnf *variables clauses*

at the top of each DIMACS instance, where *variables* is the number of distinct variables (i.e. the highest variable) occurring within the encoding and *clauses* specifies the total number of clauses.

Example 2.5. *The DIMACS instance of formula* $(\neg\alpha \vee \beta) \wedge (\beta \vee \neg\gamma)$ *looks as follows:*

p cnf 3 2
-1 2 0
2 -3 0

The given SAT instance is satisfiable, with one solution being the model '-1 -2 -3'.

Apart from regular SAT solvers, as described above, there are a few special kinds of SAT solvers, such as MaxSAT and partial MaxSAT solvers. A MaxSAT solver solves instances of the *Maximum Satisfiability Problem*, which is a generalization of the *Boolean satisfiability problem*. It corresponds to finding the maximum number of clauses, that can be satisfied by any truth assignment, given a CNF formula. By contrast, a *Partial Maximum Satisfiability Problem*, solvable by a partial MaxSAT solver, makes a distinction between *hard* clauses, that need to be satisfied always, and *soft* clauses, whose satisfiability is optional and which are thus the target of maximization. A common format used for such encodings is the WDIMACS format. The format is similar to the DIMACS format, but the first number of each line indicates the clause's weight, followed by the actual clause and - as usual - a closing 0. To distinguish hard clauses from soft clauses, all hard clauses are assigned the same weight, which must be greater than the sum of all soft clauses' weights. The problem line in WDIMACS is of the form

p wcnf *variables clauses maxWeight*

with *variables* and *clauses* defined as in the DIMACS format and *maxWeight* being the weight assigned to the hard clauses.

Example 2.6. *Assuming that the first conjunct is a hard clause and the second one a soft clause, the WDIMACS instance of* $(\neg\alpha \vee \beta) \wedge (\beta \vee \neg\gamma)$ *looks as follows:*

p wcnf 3 2 2
2 -1 2 0
1 2 -3 0

An optimal solution is '-1 -2 -3', as it satisfies both the soft and the hard clause.

Further information on the WDIMACS format can be found e.g. in the MaxHS solver's online documentation [1].

2.8.2 Integer linear programming (ILP)

Integer Linear Programming (ILP) deals with the solving of *integer linear programs* - programs where all unknowns are integers and which consist of linear constraints and an (optional) linear objective function. [Sch86] A solution to an ILP program is a value assignment to all its variables. While the exact syntax of an ILP program depends on the used *ILP solver*, a simple example of such a program is

$$\begin{aligned} \text{minimize} \quad & x + y + z \\ \text{subject to} \quad & x, y, z \in \mathbb{N}_0 \\ & x + z \geq 5 \end{aligned}$$

with one solution being $x = 5$, $y = 0$ and $z = 0$. The first line is a linear objective function, aiming at minimizing the sum of variables, whereas the second line defines the type of variables and the third one constitutes a linear constraint. The special case, where all unknowns are binary, is referred to as *0-1 Integer Linear Programming*. All ILP encodings suggested in this work are 0-1 ILP problems. Throughout this paper we denote the definition of ILP binary variables by the expression $def_b(x)$, where x is either a variable or set of variables.

Every propositional CNF formula α can be translated into a corresponding ILP program by applying the following rules [LZD04]:

- For every atom a_i in α, with $1 \leq i \leq |Var(\alpha)|$, a corresponding ILP binary variable b_i is defined. Values 0 and 1 represent the truth values *false* and *true*, respectively.
- Every clause γ of α is translated into an ILP constraint. Translation happens by creating an inequality equation of the form $\geq$ where the left side is a sum and the right side is 1. The sum on the equation's left is the sum of all variables b_i whose corresponding variables a_i occur in γ as positive literals and the expressions $(1 - b_i)$ for those variables b_i whose corresponding variables a_i occur in γ as negative literals. In case the clause consists of exactly one literal, an equality equation can be used instead, setting the corresponding variable b_i either equal to 0 (in case of a negative literal) or equal to 1 (in case of a positive literal). We denote the set of ILP constraints representing a CNF formula α by $con(\alpha, X)$, where X is the set of ILP binary variables, that represent the atoms in α.

Example 2.7. *The CNF formula $\mu = (a \vee \neg b \vee c) \wedge b \wedge (\neg b \vee \neg c)$ can be translated into below ILP encoding, after creating the corresponding ILP binary variables b_1, b_2 and b_3:*

constraint 1:	b_1 + *(1 -* b_2*)* + $b_3 \geq 1$
constraint 2:	b_2 = *1*
constraint 3:	*(1 -* b_2*)* + *(1 -* b_3*)* ≥ 1

[1] `http://www.maxhs.org/docs/wdimacs.html`

After applying the usual transformation rules for inequality equations, we alternatively obtain:

constraint 1:	b_1 - b_2 + $b_3 \geq 0$
constraint 2:	b_2 = *1*
constraint 3:	-b_2 - $b_3 \geq -1$

The set of solutions to a formula's ILP program corresponds to the formula's set of models.

2.8.3 Answer set programming (ASP)

Answer Set Programming (ASP) addresses the solution of complex combinatorial problems, taking a declarative approach to problem solving. [Lif19] Instead of writing an algorithm for finding correct solutions (imperative programming), solutions to a problem instance are described by an ASP program, using a declarative modeling language. The basic building blocks of ASP logic programs are rules, facts and integrity constraints. *Rules* are of the form

$$head :- body.$$

and always end with a dot. The head consists of an atom, whereas the body is of the form $l_1, ...l_n$, where each l_i ($1 \leq i \leq n$) is a literal and commas are read as *and*. While ASP programs do support classical negation ($-$), they also support the concept of default negation, which is expressed by placing a not in front of an atom. The symbol $:-$ can be interpreted as an *if*. Accordingly, a rule states that if the body is fulfilled, then the head must also be true; one also says that the head *is derived*. As an example, consider the following rule, which states that atom a_3 must be fulfilled, if atoms a_0 and a_2 are fulfilled and a_1 is not.

$$a_3 :- a_0,\ not\ a_1,\ a_2.$$

A rule without a body is called a *fact*, where the symbol $:-$ is omitted. As the name suggests, a fact describes something that must always be true. The below fact states that atom a_4 must be fulfilled.

$$a_4.$$

In contrast, a rule without a head is an *integrity constraint*, as it describes a situation that cannot occur. Below integrity constraint expresses that it cannot be the case that atom a_6 is fulfilled and a_5 is not.

$$:- not\ a_5,\ a_6.$$

In this work we make use of first-order ASP concepts and all our atoms are therefore predicate atoms. A *predicate atom* is of the form $p(t_1, ..., t_n)$ where p is a string starting with a lowercase letter and is essentially the name of the predicate. Inside the brackets, there are one or more terms as arguments. A *term* can be e.g. a constant (starting with lowercase letter), an integer, a string (enclosed in double quotes), a variable (starting with uppercase letter) or another predicate atom. Whereas constants, integers

and strings represent themselves, variables are placeholders for all variable-free terms within the logic program. As an example, consider below logic program.

$$bird(tweety).$$
$$dog(snoopy).$$
$$fly(X) :- bird(X).$$

The program consists of two facts and a rule. There are three distinct predicates, namely $bird/1$, $dog/1$ and $fly/1$ (/ followed by a number indicates the arity of a predicate), as well as two constants $tweety$ and $snoopy$ and the variable X. As mentioned above, a variable is a placeholder for all variable-free terms within the program. Thus, for this specific example, X is a placeholder for $tweety$, $snoopy$, $bird(tweety)$ and $dog(snoopy)$. Since the body of the rule is only fulfilled for $X = tweety$, the solution to the program contains the atom $fly(tweety)$ (in addition to $bird(tweety)$ and $dog(snoopy)$), but not $fly(snoopy)$, $fly(bird(tweety))$ or $fly(dog(snoopy))$.

Computer programs that are able to solve ASP logic programs are called *answer set solvers*. An answer set solver returns one or more so-called *answer sets* (also referred to as *stable models*) to a given logic program, which are essentially sets of atoms, whose satisfaction can be acyclically derived from the programs's rules, facts and integrity constraints. An atom is thus considered to be *true* if it is contained in the answer set and *false* otherwise. A logic program containing any kind of contradiction, e.g. derivation that an atom is satisfied and unsatisfied at the same time, does not have any answer sets.

Apart from the so far defined concepts, there are further expressions that can be used in logic programs. The list of all syntax constructs supported within an ASP program varies with different answer set solvers and their supported ASP dialects. In the following we limit ourselves to introducing those additional constructs, that are used within the ASP encodings proposed in Chapter 3. *Choice rules* are of the form

$$\{l_1; ...; l_n\}.$$

and tell an answer set solver to choose arbitrarily, which of the contained literals should be fulfilled by an answer set. Such a rule allows for a solution that satisfies none, some or all of the literals. As an example, a program consisting solely of the choice rule

$$\{bird(tweety);\ dog(snoopy)\}.$$

has four different answer sets: the empty set, the set containing only $bird(tweety)$, the set containing only $dog(snoopy)$ and the set containing both literals. Choice rules can also be turned into *cardinality constraints*, that set a lower (l) or upper (u) limit to the number of satisfied literals:

$$l\ \{l_1; ...; l_n\}\ u.$$

Lower and upper limits can either be used in combination or alone. An example for a cardinality constraint, that uses both, is

$$1\{bird(tweety);\ dog(snoopy);\ dog(daisy)\}2.$$

which states that at least 1 and at most 2 of the literals within the curly brackets are allowed to be satisfied at the same time. *Interval operators* (..) can be used to create several instances of a predicate in just one line. E.g. the line

$$isInteger(1..3).$$

is automatically expanded by an answer set solver to

$$isInteger(1).$$
$$isInteger(2).$$
$$isInteger(3).$$

Further, the aggregate #*count* can be used to count the number of elements in a given set. The aggregate is of the form

$$\#count\{t_1, ..., t_n \ : \ l_1, ..., l_n\}$$

where t_i ($1 \leq i \leq n$) are terms and l_j ($1 \leq j \leq n$) literals. The operator : can be interpreted as | in the context of set definitions, i.e. the set to be counted contains those terms t_i, for which the literals l_j are satisfied. Aggregates are often used along with *comparisons* ($=$, $!=$, $<$, $>$, $\leq$, $\geq$). Consider below logic program

$$\{bird(tweety); bird(charlie)\}.$$
$$dog(snoopy).$$
$$:- \ \#count\{X : bird(X)\} = 1.$$

with answer sets $\{dog(snoopy)\}$ and $\{dog(snoopy), bird(tweety), bird(charlie)\}$. The first line states that the solver can choose arbitrarily which of $bird(tweety)$ and $bird(charlie)$ are in the answer set. Line 2 is a fact and line 3 an integrity constraint, which states that the number of terms, for which $bird(X)$ is satisfied cannot be 1. Accordingly, the only two options are to either conclude that neither $tweety$ nor $charlie$ are birds or to conclude that both are birds. Similarly to ILP programs, there are optimization constructs, such as $\#minimize$, that tell the answer set solver to look for an *optimal* answer set. In our ASP encodings we only make use of minimization, which has the format

$$\#minimize\{w, t_1, ..., t_n \ : \ l_1, ..., l_n\}.$$

where w is a numerical value (a weight) assigned to each pair $t_1, ..., t_n$, for which the literals $l_1, ..., l_n$ hold. This statement makes the answer set solver determine an answer set, in which the sum of all weights is smallest. As can be seen, the minimization construct is very similar to the aggregate construct explained above, with the only difference being the additional weight specification. Finally, we introduce the $\#show$ construct. Oftentimes, only a subset of the atoms of an answer set are relevant as a solution to a given logic problem, while the remaining ones can be ignored. By using a $\#show$ construct an answer set solver's output can be restricted to only a specific type of atoms to

improve readability. When adding the expression $\#show\ bird/1.$ to the logic program used above

$$\{bird(tweety); bird(charlie)\}.$$
$$dog(snoopy).$$
$$:-\ \#count\{X : bird(X)\} = 1.$$
$$\#show\ bird/1.$$

the printed answer sets are now $\{\}$ and $\{bird(tweety), bird(charlie)\}$. Note that $\#show$ only influences which atoms are printed by the answer set solver, not the actual content of the answer sets.

Now that we have defined all ASP constructs, that are required for the later introduced ASP encodings, we have a short look at how a model of a propositional CNF formula α can be determined via an ASP logic program. Let us assume that $\alpha = (a \vee \neg b) \wedge a$. Firstly, we need to establish how to represent the atoms a and b in the logic program. While there are many different options, we represent them by integers. Hence, let a be represented by 1 and let b be represented by 2. Next, we define a predicate $t/1$, which denotes that the predicate's argument (here always an integer representing a propositional atom) is *true*. The first line of our logic program then looks as follows:

$$\{t(1..2)\}.$$

This choice rule, that uses the interval operator, tells the program that either none, one or all of the predicate atoms $t(1)$ and $t(2)$ can be in the answer set. We can then express the first clause of α by negating all literals and translating them into an integrity constraint:

$$:-\ not\ t(1),\ t(2).$$

This constraint ensures that it cannot be the case that $t(2)$ is in the answer set (i.e. b is *true*) whereas $t(1)$ is not (i.e. a is *false*), which is exactly what the clause $a \vee \neg b$ states. Analogously, we can translate the second clause of α into

$$:-\ not\ t(1).$$

The final logic program then has the answer sets $\{t(1)\}$ and $\{t(1), t(2)\}$, which correspond to the models of α.

2.9 Related work

As has become evident in the preceding sections, belief change has been studied widely from a theoretical perspective, including the proposition of several specific belief change operators and their axiomatic analysis. By contrast, there exists only a quite limited number of contributions targeting actual implementations of such operators. This might be, to a great extent, due to the computational intractability of the proposed belief change operators. In 1992, Eiter and Gottlob [EG92] conducted a study of the

computational costs of several distinct revision and update operators in the context of propositional logic, concluding that complexion-wise most of the operators are located on the second level of the polynomial hierarchy. Precisely, the authors analyzed the complexity of conducting inference checks, which correspond to the problem of deciding whether $K \dot{+} \mu \models \phi$ holds, with K being a belief base and μ and ϕ being propositional sentences. It was determined, that an inference check for Dalal's revision operator is complete for the complexity class $\Delta_2^P[log\ n] = P^{NP[O(log\ n)]}$, which is a subclass of $\Delta_2^P = P^{NP}$, and can hence be executed in polynomial time by a logarithmic number of calls to an NP-oracle. By contrast, an inference check using Satoh's revision operator is computationally even more complex. It is Π_2^P-complete and therefore located on the second level of the polynomial hierarchy [EG92]. With regards to model checking, i.e. deciding whether $M \models K \dot{+} \mu$ holds for an interpretation M, it was shown by Liberatore and Schaerf [LS01] that Dalal's revision operator is of complexity $P^{NP[O(log\ n)]}$-complete, whereas Satoh's operator is Σ_2^P-complete.

To address the high computational complexity of inference checks for the family of so-called topic-decomposable distance-based revision operators, with Dalal's revision operator $\dot{+}_D$ being a member of the same, Konieczny et al. [KLM17] suggested a compilation-based approach to belief revision. The concept of compilation was first proposed and analyzed by Liberatore [Lib98] and consists in dividing intractable problems into two parts, of which one is known beforehand (fixed part) and the other at execution time (varying part). As part of the compilation process, the fixed part is then preprocessed and compiled into a data structure, that can later be leveraged for the solving of the varying part. As an example, let the examination of whether $K \dot{+} \mu \models \alpha$ and $K \dot{+} \mu \models \beta$ hold, be two intractable problems. Each problem can be divided into a fixed part - the revision - and a varying part - the inference check. Applying the idea of compilation, the revision can be preprocessed and then compiled into an encoding, that facilitates the subsequent inference check. Since both problems have a common fixed part ($K \dot{+} \mu$), the compilation needs to be carried out only once. Konieczny et al.'s proposal is based on first determining the minimum Dalal distance (preprocessing), using a partial MaxSAT optimization encoding and a call to a partial MaxSAT solver, and thereafter generating a SAT encoding of the new belief base, incorporating the obtained minimum Dalal distance, that is query-equivalent to the revised belief base and thus reduces the complexity of inference checks to coNP-completeness. Their main motivation lies in leveraging the power of SAT solvers, which have in recent years become a powerful tool for solving feasibility problems. Whereas the authors conducted a thorough performance analysis of the compilation process, the biggest downside of their work is the lack of an investigation into the performance of the suggested encodings with regards to inference and model checks. Furthermore, Konieczny et al. do neither address belief contraction in their work, nor do they consider alternate kinds of operators, such as e.g. set-containment-based ones.

While there are further contributions, that leverage SAT and ASP technologies for implementing belief change operators, examples being the works by Aravanis [Ara22], Delgrande et al. [DLST07], Grégoire et al. [GLM14], Hunter and Agapeyev [HA19],

and Sérayet et al. [SDP09], to the best of our knowledge, none of them applies the idea of compilation. The work by Hunter and Agapeyev can be further critizised because of its reliance on parallelization to speed up an essentially naive implementation of belief revision. Further, existing implementations usually focus on either belief revision or contraction (see e.g. [Ara22], [GLM14], [HA19], [KLM17], [SDP09]), or they consider operators of only one specific family (see e.g. [Ara22], [DLST07], [HA19], [KLM17]), with AGM and cardinality-based operators, such as Dalal's, being the most popular ones. Besides, there seems to be, as of now, no work targeting an implementation of Satoh's operator.

Taking into account the described research situation, our thesis constitutes an important contribution to the field of practical belief change in two main ways. The first one is regarding the application of the concept of compilation to belief change implementations, which is a so far largely neglected approach. Considering that the only currently existing compilation-based approach to belief change, namely the one by Konieczny et al., focuses entirely on the encoding process as such, our application, which also implements inference and model checks on the generated encodings, is the first complete, compilation-founded implementation of belief change. Moreover, our work complements the one by Konieczny et al. by conducting a thorough evaluation of the belief bases' encodings in inference and model checks, allowing us to collect some insights into how efficient the proposed compilation-based approach actually is. The second one concerns the fact that both belief revision and contraction, as well as operators from two distinct families (cardinality- and set-containment-based operators) are considered. The last point is important in that a belief change application, suitable for real-world problems, should support as many different kinds of operations as possible. Even though our application targets only 4 specific operators ($\dot{+}_D$, $\dot{+}_S$, $\dot{-}_D$, $\dot{-}_S$), rather than an entire family of operators, as some existing contributions do (e.g. [Ara22], [KLM17] etc.), the fact that the operators are from distinct categories (cardinality- and set-containment-based) and for distinct kinds of operations (revision and contraction), is crucial. It is evident that extending an existing application by a new operator, that is of the same family as an already implemented operator, will be in most - if not all - cases less demanding than adding support for an operator of an entirely distinct family.

3 Implementation

In this chapter we provide a thorough description of the implementation details of our compilation-based belief change application. In particular we describe the application's architecture, its main algorithm schemes and propose SAT, ILP, and ASP encoding schemes for the four supported belief change operators. Note that in the course of the current and following chapters, we sometimes simplify the discussion by referring to our proposed encoding schemes as 'encodings'.

Throughout this chapter let κ be a CNF formula representing a consistent belief base, let μ be a satisfiable CNF formula representing a belief that is to be added to κ and let ϕ be a non-tautological CNF formula representing a belief that is to be contracted from κ. Further, let $Var(\kappa) \cup Var(\mu) = \{x_1, ..., x_n\} = X$ for a given change operation involving μ or $Var(\kappa) \cup Var(\phi) = \{x_1, ..., x_n\} = X$ for a given change operation involving ϕ. Consequently, let n be the total number of atoms subject to the given change operation. Moreover, let γ be a satisfiable, non-tautological CNF formula with $Var(\gamma) \subseteq Var(\kappa) \cup Var(\mu)$ for a given change operation involving μ or $Var(\gamma) \subseteq Var(\kappa) \cup Var(\phi)$ for a given change operation involving ϕ. Finally, let N be an interpretation of $Var(\kappa) \cup Var(\mu)$ for a given change operation involving μ or an interpretation of $Var(\kappa) \cup Var(\phi)$ for a given change operation involving ϕ.

3.1 Application architecture

Our application for compilation-based belief change is implemented in Java 17 and calls the partial MaxSat solver *MaxHS*, the SAT solver *CaDiCal*, the ILP solver *glpsol*, which is the LP/MIP stand-alone solver contained in the *GLPK* (GNU Linear Programming Kit) package, and the ASP solver *clingo*. The installation of these solvers is a pre-requisite for the application to run on a given system. For conducting the Tseitin transformation, the LogicNG Java library [2] is used. The application's source code is publicly available on GitHub [3]. Links to the download pages of the required solvers can be found in the repository's README file.

The application is started with a set of mandatory and optional input parameters and supports three distinct run modes: a **compilation mode** (flag `-C`), in which the belief change encoding for a provided revision or contraction instance is generated, and **inference** and **model check modes**, for executing inference (flag `-I`) and model (flag `-M`) checks on a previously generated encoding.

Input for the compilation mode consists of the following:

- A file containing a finite belief base CNF formula and a change CNF formula in an application-specific format specified via `-f <file path>`. The application-specific format is based on the well-known DIMACS format, with the only difference being an additional line of the form `n ---`, that works as a separator between belief base clauses (above the line) and change formula clauses (below the line).

[2] `https://github.com/logic-ng/LogicNG`

[3] `https://github.com/aig-hagen/msc_2023_julia_hayat`

- Specification of the intended operation: `-o <revision;contraction>`
- Specification of the intended operator: `-d <dalal;satoh>`
- The desired preprocessing algorithm: `-a <maxsat;asp;ilp>`
- The desired type of encoding: `-t <sat;asp;ilp>`
- Optional flag to skip the input validation step: `-s`

The inference and model check modes require below inputs:

- A previously generated encoding, provided via `-e <file path>`
- A file containing a formula for an inference check or an interpretation for a model check, provided via `-f <file path>`. The inference formula must be in DIMACS format without a problem line and must contain only variables of the original belief change instance's signature. The interpretation, on the other hand, must be an interpretation of the original belief change instance's signature. Since the original belief change instance was specified in DIMACS format, its signature consists of consecutive natural numbers, starting from 1. The interpretation must be a one-liner, containing all these numbers in ascending order with a single space as delimiter and specifying negativity of atoms via the minus symbol.
- Optional flag to skip the input validation step: `-s`

Figure 3.1 shows an architectural overview of the application. In the subsequent two sections each run mode is described in more detail.

3.1.1 Compilation mode

The compilation mode consists of an input validation step, followed by the preprocessing, that makes one or more calls to an external solver, depending on the selected change operator, and finally the generation of the belief change encoding, which is then written to a file. The validation step ensures that the provided belief base formula is satisfiable and that the change formula is both satisfiable and non-tautological. Moreover, in the case of revision it is ensured that the change formula is not yet believed in the belief base, whereas in the case of contraction it is ensured that the change formula is believed in the belief base. If any of the mentioned checks fails, the application is aborted immediately with a dedicated message. Note that using the flag `-s` allows for skipping the validation step, even though this should only be used when the correctness of a belief change instance has been ensured by other means, since otherwise the correctness of the application output cannot be guaranteed. Furthermore, the application always carries out a short file format validation, which cannot be skipped, not even by the flag `-s`.

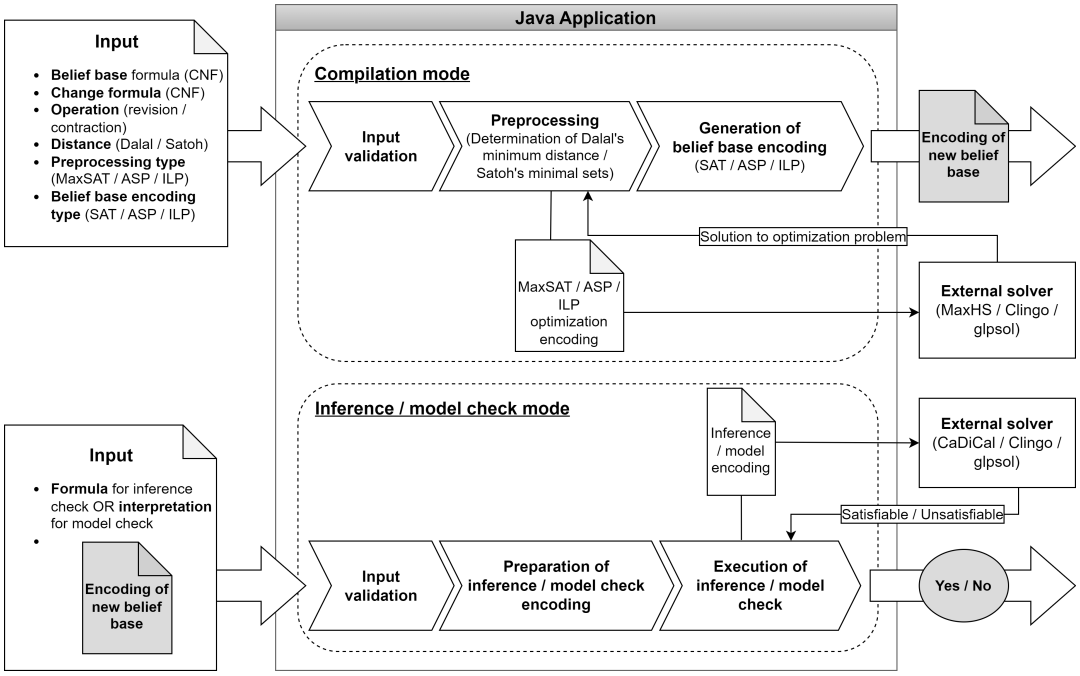


Figure 3.1: Application architecture

3.1.2 Inference and model check modes

The inference and model check modes both consist of an input validation step, the preparation of an encoding for the subsequent inference or model check and finally, the execution of the specified check by making a single call to an external solver and delivering a Yes or No answer based on the solver's output. The validation step ensures that the provided encoding is satisfiable and that the inference formula is both satisfiable and non-tautological. Again, the described validation can be skipped by using the flag `-s`, whereas the file format validation cannot.

3.2 Algorithms

In this section let $\alpha = \mu$ in case of a revision operation and $\alpha = \phi$ in case of a contraction operation. In the following the algorithm schemes used within our belief change application are introduced.

3.2.1 Compilation algorithm schemes

Algorithms 3.1 and 3.2 show our compilation algorithm schemes for Dalal's and Satoh's operators, respectively. The algorithm scheme for Dalal's operators is based on the approach first suggested by Konieczny et al. [KLM17], which consists of creating an optimization encoding (in their case partial MaxSAT) for determining the minimum Dalal

distance (lines 3 and 5) , followed by a single call to a corresponding solver (line 7), the extraction of the minimum Dalal distance value from the obtained solution (line 8), and finally the creation of an encoding (in their case SAT) of the new belief base that incorporates the determined minimum Dalal distance (lines 10 and 12). Note that whereas Konieczny et al. limited their work to belief revision, our algorithm scheme also addresses belief contraction. Further, Algorithm 3.1 depicts an algorithm scheme rather than a proper algorithm because, depending on the selected encoding type, every mentioned encoding function can return three different encodings. We denote the encoding generated by function `createOptimizationEncodingRevision()` by $S^O_{+_D}(\kappa, \alpha)$, $I^O_{+_D}(\kappa, \alpha)$, or $A^O_{+_D}(\kappa, \alpha)$ for partial MaxSAT, ILP, and ASP, respectively. Analogously, function `createOptimizationEncodingContraction()` generates encoding $S^O_{-_D}(\kappa, \alpha)$, $I^O_{-_D}(\kappa, \alpha)$, or $A^O_{-_D}(\kappa, \alpha)$. Further, `createEncodingRevision()` generates encoding $S_{+_D}(\kappa, \alpha)$, $I_{+_D}(\kappa, \alpha)$, or $A_{+_D}(\kappa, \alpha)$ for SAT, ILP, and ASP, respectively, and `createEncodingContraction()` encoding $S_{\dot{-}_D}(\kappa, \alpha)$, $I_{\dot{-}_D}(\kappa, \alpha)$, or $A_{\dot{-}_D}(\kappa, \alpha)$.

Algorithm 3.1 Compilation algorithm scheme - Dalal

```
1:  function COMPILEDALAL(κ, α, operation)
2:      if operation == revision then
3:          o ← CREATEOPTIMIZATIONENCODINGREVISION(κ, α)
4:      else if operation == contraction then
5:          o ← CREATEOPTIMIZATIONENCODINGCONTRACTION(κ, α)
6:      end if
7:      solutionModel ← CALLSOLVER(o)
8:      minDistance ← EXTRACTMINDISTANCE(solutionModel)
9:      if operation == revision then
10:         return CREATEENCODINGREVISION(κ, α, minDistance)
11:     else if operation == contraction then
12:         return CREATEENCODINGCONTRACTION(κ, α, minDistance)
13:     end if
14: end function
```

For Satoh's encodings it is sufficient to identify only those minimal sets of the set of all difference sets, that are at the same time proper subsets of some difference set. This is described in more detail in the upcoming sections and for now it is enough to point out that when referring to the determination of minimal sets, we actually refer to the determination of this special kind of minimal sets. Similar to the scheme for Dalal's operators, the algorithm scheme for Satoh's operators starts with the generation of an optimization encoding to determine the minimal sets (lines 3 and 5). However, whereas in Dalal's case only one solver call is required, in Satoh's there is a loop consisting of solver calls (line 16), extraction of the minimal set information from the solver's output (line 10) and continuous adjustments of the optimization encoding (line 15). The loop is only aborted once the encoding has turned unsatisfiable, i.e. all min-

imal sets have been identified. The final step is similar to Dalal's in that it consists of the generation of an encoding of the new belief base that incorporates the identified minimal sets (lines 19 and 21). Two special cases are conceivable. The first one occurs when the generated optimization encoding is unsatisfiable right away, i.e. there exists no minimal set. In this case the empty set is passed to the function, that generates the final encoding. The second special case is that the minimal set extracted from the solver output is the empty set. This can occur only in the first iteration of the loop since the empty set being a minimal set entails that there can be no other minimal set. In this case the loop is terminated immediately (line 13). We denote the encoding generated in function `createOptimizationEncodingRevision()` by $S^O_{+_S}(\kappa, \alpha)$, $I^O_{+_S}(\kappa, \alpha)$, or $A^O_{+_S}(\kappa, \alpha)$ for partial MaxSAT, ILP, and ASP, respectively. Analogously, function `createOptimizationEncodingContraction()` generates the encoding $S^O_{-_S}(\kappa, \alpha)$, $I^O_{-_S}(\kappa, \alpha)$, or $A^O_{-_S}(\kappa, \alpha)$. Further, `createEncodingRevision()` generates encoding $S_{+_S}(\kappa, \alpha)$, $I_{+_S}(\kappa, \alpha)$, or $A_{+_S}(\kappa, \alpha)$ for SAT, ILP, and ASP, respectively, and `createEncodingContraction()` outputs encoding $S_{\dot{-}_S}(\kappa, \alpha)$, $I_{\dot{-}_S}(\kappa, \alpha)$, or $A_{\dot{-}_S}(\kappa, \alpha)$.

Algorithm 3.2 Compilation algorithm scheme - Satoh

```
1:  function COMPILESATOH(κ, α, operation)
2:      if operation == revision then
3:          o ← CREATEOPTIMIZATIONENCODINGREVISION(κ, α)
4:      else if operation == contraction then
5:          o ← CREATEOPTIMIZATIONENCODINGCONTRACTION(κ, α)
6:      end if
7:      solutionModel ← CALLSOLVER(o)
8:      minSets ← empty set
9:      while solutionModel is not empty do
10:         minSet ← EXTRACTMINIMALSET(solutionModel)
11:         minSets ← minSets ∪ minSet
12:         if minSet is the empty set then
13:             break
14:         end if
15:         o ← ADJUSTOPTIMIZATIONENCODING(o, minSet)
16:         solutionModel ← CALLSOLVER(o)
17:     end while
18:     if operation == revision then
19:         return CREATEENCODINGREVISION(κ, α, minSets)
20:     else if operation == contraction then
21:         return CREATEENCODINGCONTRACTION(κ, α, minSets)
22:     end if
23: end function
```

From the above descriptions we obtain that our application implements $3 * 3 = 9$

different instances of each of the two algorithm schemes by supporting the creation of partial MaxSAT, ASP, and ILP optimization encodings for the preprocessing, as well as the compilation of the new belief state into SAT, ASP, and ILP encodings. Note that using the same technology for both the preprocessing and the actual encoding generation has the slight advantage that certain parts of the optimization encoding can be reused within the final encoding, allowing to skip some generation steps. The subsequent three sections 3.3, 3.4, and 3.5 each address one technology (SAT, ILP, ASP) and contain subsections for each of the four belief change operators, where the encoding schemes are defined and their correctness is proven.

3.2.2 Inference and model check algorithm schemes

Algorithms 3.3 and 3.4 demonstrate how the application carries out inference (model) checks, when provided a previously generated encoding and a formula for an inference check (interpretation for a model check). $S_I(x, y)$, $I_I(x, y)$, and $A_I(x, y)$ are inference check encoding schemes, that are unsatisfiable if and only if a given formula can be inferred from the belief base represented by the provided encoding. Encoding schemes $S_M(x, y)$, $I_M(x, y)$, and $A_M(x, y)$, on the other hand, are model check encodings, that are satisfiable if and only if a given interpretation is a model of the belief base represented by the provided encoding. The inference check encoding schemes are defined and proven in sections 3.3.6, 3.4.5 and 3.5.5, whereas the definitions of the model check encoding schemes, along with their proofs, can be found in sections 3.3.7, 3.4.6 and 3.5.6.

Algorithm 3.3 Algorithm scheme for inference checks

```
1:  function checkForInference(encoding, γ)
2:      if encoding is SAT encoding then
3:          inferenceEncoding ← S_I(encoding, γ)
4:          solutionModel ← callCaDiCal(inferenceEncoding)
5:      else if encoding is ILP encoding then
6:          inferenceEncoding ← I_I(encoding, γ)
7:          solutionModel ← callGlpsol(inferenceEncoding)
8:      else if encoding is ASP encoding then
9:          inferenceEncoding ← A_I(encoding, γ)
10:         solutionModel ← callClingo(inferenceEncoding)
11:     end if
12:     if solutionModel is null then
13:         return true
14:     else
15:         return false
16:     end if
17: end function
```

Algorithm 3.4 Algorithm scheme for model checks

```
1: function CHECKFORMODEL(encoding, N)
2:     if encoding is SAT encoding then
3:         modelEncoding ← S_M(encoding, N)
4:         solutionModel ← CALLCADICAL(modelEncoding)
5:     else if encoding is ILP encoding then
6:         modelEncoding ← I_M(encoding, N)
7:         solutionModel ← CALLGLPSOL(modelEncoding)
8:     else if encoding is ASP encoding then
9:         modelEncoding ← A_M(encoding, N)
10:        solutionModel ← CALLCLINGO(modelEncoding)
11:    end if
12:    if solutionModel is null then
13:        return false
14:    else
15:        return true
16:    end if
17: end function
```

3.3 SAT encodings

In this section we propose the SAT encoding schemes $S^O_{+_D}(\kappa, \mu)$, $S_{\dot{+}_D}(\kappa, \mu)$, $S^O_{-_D}(\kappa, \phi)$, $S_{\dot{-}_D}(\kappa, \phi)$, $S^O_{+_S}(\kappa, \mu)$, $S_{\dot{+}_S}(\kappa, \mu)$, $S^O_{-_S}(\kappa, \phi)$, and $S_{\dot{-}_S}(\kappa, \phi)$ generated and used by algorithms 3.1 and 3.2 as well as the SAT inference and model check encodings $S_I(x, y)$ and $S_M(x, y)$ generated and used by algorithms 3.3 and 3.4. The proofs of all lemata and theorems established in this section are provided in Appendix A.

3.3.1 Boolean cardinality constraint encoding

For the belief change SAT encodings that are proposed in the upcoming sections, we require a scheme to encode boolean cardinality constraints as SAT programs. A *boolean cardinality constraint* is a constraint stating that out of a set of propositional atoms, less than, more than, at least, at most or exactly k atoms are allowed to be *true* at the same time [ANORC13]. Formally, such a constraint can be described by

$$\sum_{i=1}^{n} value(V_i) \ \# \ k$$

where $k \in \mathbb{N}_0$ and $\#$ one of {<,>,≤, ≥, =}. Further, $value(V_i)$ represents the truth value of a boolean variable $V_i \in V$ and $V = \{V_1, ..., V_n\}$ denotes a set of boolean variables subject to the cardinality constraint. In the following, we limit ourselves to only one type of boolean cardinality constraint, namely that where $\#$ equals '=', and refer to constraints of this kind as *exactly-k constraints*.

In order to create a SAT encoding for a given exactly-k constraint, we differentiate between the cases $k = 0$ and $k > 0$. In the first case, the SAT encoding is quite simple, consisting of only one clause per variable of the set of variables, that is to be counted, with each clause requiring the corresponding variable to take the value 0. Regarding the second case, the main idea is to first create a binary counter CNF formula, that counts the number c of true variables of the given set of variables subject to the cardinality constraint. This formula contains several new variables, among them dedicated variables, that each represent a bit of the binary representation of c, and is then expanded by adding a clause for each such bit variable, setting the variable equal to 0 or 1, such that for all of its models $c = k$ holds. Following the work by Konieczny et al. we use a binary counter CNF encoding, that was suggested by Sinz in [Sin05] and that is based on a parallel counter circuit designed by Muller and Preparata [MP75].

For describing the functioning of the parallel counter circuit by Muller and Preparata, we assume that there is a set of propositional variables $\{x_1, ..., x_n\}$, wherein the number of true variables is to be counted. As can be seen in Figure 3.2, the set of variables is recursively split into two halves $x_1...x_{2^m-1}$ and $x_{2^m}...x_{n-1}$, as well as a separate variable x_n, by setting $m = \log_2(n)$. The two halves are each handed to sub-counters, where they are again split into halves and so on. Depending on the value of n, the second half can be smaller than the first, or even entirely empty, and the second sub-counter can hence have less inputs than the first sub-counter. The results of both sub-counters, represented by two binary numbers, consisting of bits $y_{m-1}, ..., y_0$ and $z_{m-1}, ..., z_0$ (in the case that the first and second half are of equal length), respectively, are then added using an m-bit adder. For the m-bit adder m 1-bit adders are required, which can be either full-adders (adding three bits) or half-adders (adding two bits). Whenever the first and second half input bits are of equal number, which is the case when n is equal to $2^{m+1}-1$, all 1-bit adders of the m-bit adder are full-adders. This is the case in Figure 3.2. In the opposite case, some of them will be half-adders as the binary number resulting from the second sub-counter will have less bits than the binary number resulting from the first sub-counter. As stated by Sinz, a parallel binary counter implemented in the described way, requires $n - \log_2(n) - 1$ full-adders and at most $\log_2(n)$ half-adders.

Each full- and half-adder outputs a two-bit binary number. In the following, the right bit will be called *sum*, denoted by s_{out}, and the left bit *carry*, denoted by c_{out}.

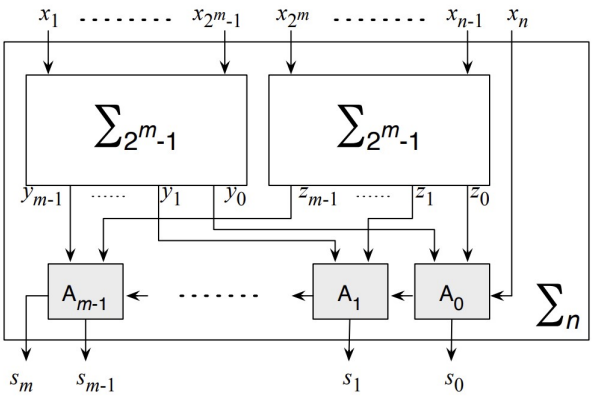
Definition 3.1. *A half-adder encoding for adding two binary variables a and b can be implemented by the following CNF formula [Sin05]:*

$$A_H(a, b, c_{out}, s_{out}) = (a \vee \neg b \vee s_{out}) \wedge (\neg a \vee b \vee s_{out}) \wedge (\neg a \vee \neg b \vee c_{out})$$

Definition 3.2. *A full-adder encoding for adding three binary variables a, b and c can be implemented by the following CNF formula [Sin05]:*

$$\begin{aligned} A_F(a, b, c, c_{out}, s_{out}) = &(a \vee b \vee \neg c \vee s_{out}) \wedge (a \vee \neg b \vee c \vee s_{out}) \wedge (\neg a \vee b \vee c \vee s_{out}) \\ &\wedge(\neg a \vee \neg b \vee \neg c \vee s_{out}) \wedge (\neg a \vee \neg b \vee c_{out}) \wedge (\neg a \vee \neg c \vee c_{out}) \wedge (\neg b \vee \neg c \vee c_{out}) \end{aligned}$$

Note that in both presented encodings (Definition 3.1 and 3.2) c_{out} and s_{out} do not necessarily correspond to the binary representation of the sum of a and b (a, b and c), since



Note. From [Sin05], page 830. Reproduced with permission from Springer Nature.

Figure 3.2: Parallel counter circuit designed by Muller and Preparata [MP75]

both variables are only forced to take the value 1, but not the value 0, i.e. if the sum is 2 (binary representation: 10) c_{out} is forced to have the value 1, whereas s_{out} can still be either 0 or 1. Accordingly, the binary number represented by bits c_{out} and s_{out} is *at least as great* as the sum of variables, that are to be counted. However, for our purposes this inaccuracy can be neglected, as we intend to use the proposed counter SAT encoding for the implementation of an exactly-*k* constraint where *k* equals Dalal's minimum distance d_{min}, i.e. for a case where we are certain that the number of true variables of a given set of variables cannot be smaller than k.

Example 3.1. *Let variables a and b be 1. Then the addition of a and b should equal 2, whose binary representation is 10. Accordingly, the following should hold: $s_{out} = 0$ and $c_{out} = 1$. Looking at the formula $(a \vee \neg b \vee s_{out}) \wedge (\neg a \vee b \vee s_{out}) \wedge (\neg a \vee \neg b \vee c_{out})$ we can see, that indeed for the given truth assignments the formula is satisfied only, when $c_{out} = 1$.*

Using the parallel counter circuit depicted in Figure 3.2 and the introduced formulae for full- and half-adders, a binary counter encoding, that counts the number of variables set to true in a set of binary variables, can be created. Algorithm 3.5 shows how such an encoding is generated by our belief change application. It takes as input a list v of propositional variables denoted by natural numbers and an initially empty set $clauses$. During the execution of the algorithm, the set is filled with the clauses of the binary counter encoding. The algorithm finally returns a list $resultBits$ of those new variables, that represent - in reversed order - the bits of the binary number, that indicates the amount of $true$ variables in v.

Algorithm 3.6 eventually shows the generation of the final exactly-*k* constraint encoding. As mentioned above, the case $k = 0$ is taken care of by adding one clause per variable, consisting of the negation of that variable. In contrast, the more complex case $k > 0$ is addressed by first calling Algorithm 3.5 and then adding additional clauses, that set the binary number bits in such a way, that the binary number is equal to k. We denote the exactly-*k* constraint encoding obtained by calling Algorithm 3.6 on a set v of

Algorithm 3.5 Creation of binary counter encoding for list of binary variables v

function COUNT(v, $clauses$)
  if $|v| \leq 1$ **then**
    return v
  end if
  $n \leftarrow |v|$
  $m \leftarrow$ (int) $\log_2(n)$
  sort v in ascending order
  $numMax \leftarrow$ last element of v
  $h1 \leftarrow$ sublist of v from index 0 to index $2^m - 2$ (incl.)
  if $n == 2^m$ **then**
    $h2 \leftarrow$ empty list
  else
    $h2 \leftarrow$ sublist of v from index $2^m - 1$ to index $n - 2$ (incl.)
  end if
  $result1 \leftarrow$ COUNT$(h1, clauses)$
  $result2 \leftarrow$ COUNT$(h2, clauses)$
  $carry \leftarrow$last element of v
  $resultBits \leftarrow$ empty list
  for $i = 0$ to $i =$ (last index of $result1$) **do**
    $bit1 \leftarrow$ element of $result1$ at index i
    $newCarry \leftarrow numMax + 1$
    $newSum \leftarrow numMax + 2$
    $numMax \leftarrow newSum$
    if $result2$ has index i **then**
      $bit2 \leftarrow$ element of $result2$ at index i
      $clauses \leftarrow clauses \cup A_F(bit1, bit2, carry, newCarry, newSum)$
    else
      $clauses \leftarrow clauses \cup A_H(bit1, carry, newCarry, newSum)$
    end if
    $resultBits \leftarrow resultBits \cup newSum$
    $carry \leftarrow newCarry$
  end for
  $resultBits \leftarrow resultBits \cup carry$
  return $resultBits$
end function

propositional variables and $k \in \mathbb{N}_0$ by $E(v, k)$. The binary counter encoding requires at most $7n-4(\log_2(n))-7$ clauses and at most $2*(n-1)$ new variables [Sin05], whereas the second part of the encoding consists of $m+1$ clauses (one per bit of the binary number output). Consequently, in total, our exactly-*k* constraint encoding consists of at most $7n - 4(\log_2(n)) - 7 + (m+1) = 7n - 3(\log_2(n)) - 6$ clauses and at most $2n - 2$ new variables. Note that if $k = 0$, no new variables are needed and the number of clauses corresponds exactly to n.

Algorithm 3.6 Creation of exactly-*k* constraint encoding for list v and $k \in \mathbb{N}_0$

```
1:  function CREATECONSTRAINTENCODING(v,k)
2:      clauses ← empty set
3:      if k == 0 then
4:          for var in v do
5:              newClause ← '¬counterBit'
6:              clauses ← clauses ∪ newClause
7:          end for
8:          return clauses
9:      end if
10:     counterBits ← COUNT(v, clauses)
11:     kBinary ← binary representation of k
12:     kBinary ← reverse sequence of kBinary
13:     for i = 0 to i = (last index of counterBits) do
14:         counterBit ← element of counterBits at index i
15:         if kBinary has index i then
16:             kBit ← element of kBinary at index i
17:             if kBit == 0 then
18:                 newClause ← '¬counterBit'
19:                 clauses ← clauses ∪ newClause
20:             else
21:                 newClause ← 'counterBit'
22:                 clauses ← clauses ∪ newClause
23:             end if
24:         else
25:             newClause ← '¬counterBit'
26:             clauses ← clauses ∪ newClause
27:         end if
28:     end for
29:     return clauses
30: end function
```

3.3.2 Dalal's revision (SAT)

In the following, we first present the partial MaxSAT optimization encoding $S^O_{\dot{+}_D}(\kappa, \mu)$, whose solution indicates the minimum Dalal distance $d_{min}(\kappa, \mu)$, followed by the introduction of the SAT encoding $S_{\dot{+}_D}(\kappa, \mu)$, that is query-equivalent to the belief base resulting from the execution of Dalal's revision on κ and μ.

The first part of $S^O_{\dot{+}_D}(\kappa, \mu)$ consists of the belief base formula κ, wherein we replace each variable $x_j \in X$ $(1 \leq j \leq n)$ with a variable $y_j \in Y$, with $Y = \{y_1, ..., y_n\}$ being a set of newly created variables. We denote the adjusted belief base formula by κ^y. To this first part we then add the unchanged revision formula μ. As a result we obtain

$$\kappa^y \wedge \mu \tag{3}$$

Lemma 1. *For each model M of $\kappa^y \wedge \mu$ the following holds: $proj(M, \{y_1, ..., y_n\}) = M_\kappa$, with $M_\kappa \in Mod(\kappa)$, and $proj(M, \{x_1, ..., x_n\}) = M_\mu$, with $M_\mu \in Mod(\mu)$. Further, each possible combination of members of $Mod(\kappa)$ and $Mod(\mu)$ is covered by a model of $\kappa^y \wedge \mu$.*

Next, in order to determine the Dalal distance between a model M_κ and a model M_μ we need to count the number of atoms with differing truth assignments in M_κ and M_μ. For this purpose we introduce for every x_j a so-called discrepancy variable $d_j \in \{d_1, ..., d_n\}$ and extend Formula 3 by adding the following formula for each x_j:

$$(d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j) \tag{4}$$

Lemma 2. *Formula 4 ensures that $d_j = 1$, whenever $x_j \neq y_j$.*

Thus far, our optimization encoding is identical to the one suggested by Konieczny et al. [KLM17]. However, the remaining part of our encoding deviates from theirs. Whereas they use a parallel binary counter formula and soft clauses with differing weights for obtaining $d_{min}(\kappa, \mu)$, we apply a simpler approach, using only soft clauses with identical weights and not requiring any further hard clauses. The reasons for this deviation are twofold. First, Konieczny et al. presented a rather vague description of this particular part of their optimization encoding, therefore not allowing for an exact reproduction. Second, their work targets the entire family of topic-decomposable distance-based revision operators, rather than Dalal's revision operator in particular. When considering solely Dalal's revision operator, a simpler encoding is sufficient, which is described in Definition 3.3.

Definition 3.3. *The partial MaxSAT optimization encoding $S^O_{\dot{+}_D}(\kappa, \mu)$ is defined as*

$$S^O_{\dot{+}_D}(\kappa, \mu) = \Big(\kappa^y \wedge \mu \wedge \bigwedge_{1 \leq j \leq n} \Big((d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j)\Big)\Big)_{w=n+1} \wedge \Big(\bigwedge_{1 \leq j \leq n} \Big(\neg d_j\Big)\Big)_{w=1}$$

where $\big(\big)_{w=x}$ indicates that all clauses contained within are assigned a weight of x.

All clauses of $S^O_{\dot{+}_D}(\kappa, \mu)$ with a weight of $n + 1$ are hard clauses, thus mandatory to be satisfied, whereas the clauses with a weight of 1 are soft clauses. Theorem 1 establishes how $d_{min}(\kappa, \mu)$ can be computed from the solution to the introduced encoding.

Theorem 1. *From the optimal solution S to the partial MaxSAT encoding $S^O_{\dot{+}_D}(\kappa, \mu)$ the minimum Dalal distance $d_{min}(\kappa, \mu)$ can be obtained by $d_{min}(\kappa, \mu) = \sum_{j=1}^{n} value(S, d_j)$.*

The following example demonstrates the encoding $S^O_{\dot{+}_D}(\kappa, \mu)$ for a specific belief revision instance.

Example 3.2. *Let $\kappa = x_1 \wedge (\neg x_1 \vee x_2)$ and $\mu = \neg x_2$. Then*

$$S^O_{\dot{+}_D}(\kappa, \mu) = \Big(y_1 \wedge (\neg y_1 \vee y_2) \ \wedge \ \neg x_2 \ \wedge \ (d_1 \vee \neg x_1 \vee y_1) \wedge (d_1 \vee x_1 \vee \neg y_1) \\ \wedge (d_2 \vee \neg x_2 \vee y_2) \wedge (d_2 \vee x_2 \vee \neg y_2)\Big)_{w=3} \wedge \ \Big(\neg d_1 \wedge \neg d_2\Big)_{w=1}$$

When passing the encoding to a partial MaxSAT solver, the identified solution satisfying the highest possible number of soft clauses satisfies exactly one soft clause, thus $d_{min}(\kappa, \mu) = 1$.

After determining $d_{min}(\kappa, \mu)$, the SAT encoding $S_{\dot{+}_D}(\kappa, \mu)$ of the belief base resulting from Dalal's revision can be generated. This encoding is defined in Definition 3.4.

Definition 3.4. *The SAT encoding $S_{\dot{+}_D}(\kappa, \mu)$ is defined as*

$$S_{\dot{+}_D}(\kappa, \mu) = \kappa^y \ \wedge \ \mu \ \wedge \bigwedge_{1 \leq j \leq n} \Big((d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j)\Big) \\ \wedge E(\{d_1, ..., d_n\}, d_{min}(\kappa, \mu))$$

The first part of the encoding is identical to the hard clauses of the partial MaxSAT encoding proposed above, whereas the second part consists of the exactly-*k* constraint encoding described in Section 3.3.1. This constraint ensures that the number of *true* discrepancy variables $\{d_1, ..., d_n\}$ is equal to $d_{min}(\kappa, \mu)$.

Theorem 2. *For the SAT encoding $S_{\dot{+}_D}(\kappa, \mu)$ following relation holds:* $proj(Mod(S_{\dot{+}_D}(\kappa, \mu)), \{x_1, ..., x_n\}) = Mod(\kappa \dot{+}_D \mu)$

From Theorem 2 we learn that our encoding scheme $S_{\dot{+}_D}(\kappa, \mu)$ is query-equivalent to the belief base resulting from Dalal's revision and can thus be leveraged for inference and model checks on the same. The exact nature of such checks is the subject of sections 3.3.6 and 3.3.7. We assume that our encoding scheme is identical or at least highly similar to the belief change SAT encoding scheme proposed by Konieczny et al [KLM17], as we followed their suggestion of implementing the exactly-k constraint with a parallel binary counter encoding. Due to the vague nature of their encoding description, the exact degree of similarity between the two encoding schemes cannot be stated.

3.3.3 Dalal's contraction (SAT)

In this section we first present a partial MaxSAT optimization encoding $S^O_{\dot{-}_D}(\kappa, \phi)$, whose solution indicates the minimum Dalal distance $d_{min}(\kappa, \neg\phi)$, followed by the introduction of the SAT encoding $S_{\dot{-}_D}(\kappa, \phi)$, that is query-equivalent to the belief base resulting from the execution of Dalal's contraction on κ and ϕ.

We recall from definition 2.10 that Dalal's contraction operator $\dot{-}_D$ is defined in terms of the revision operator $\dot{+}_D$ as follows:

$$Mod(\kappa \dot{-}_D \phi) = Mod(\kappa) \cup Mod(\kappa \dot{+}_D \neg\phi)$$

The models of the contraction result are hence the union of the belief base models and the models of the corresponding revision result after negating the contraction formula ϕ. Accordingly, the first step towards our aim of defining a SAT encoding for Dalal's contraction operator, is to determine the minimum Dalal distance $d_{min}(\kappa, \neg\phi)$.
Intuitively, one might immediately think of using the encoding $S^O_{\dot{+}_D}(\kappa, \mu)$ of Definition 3.3 for identifying $d_{min}(\kappa, \neg\phi)$. However, Definition 3.3 restricts μ to be a CNF formula, which $\neg\phi$ is not. To be precise, $\neg\phi$ is the negation of a CNF formula. To resolve this, we need to convert $\neg\phi$ into CNF. As already stated in Section 2.1, every propositional formula can be transformed to CNF, e.g. by applying the Tseitin transformation. In the following we denote the CNF formula obtained by applying the Tseitin transformation on a non-CNF formula α by $t(\alpha)$ and the set of auxiliary variables, that is newly created in the Tseitin transformation process, by $V^t_\alpha = Var(t(\alpha)) \setminus Var(\alpha)$. Definition 3.5 and Theorem 3 introduce the partial MaxSAT encoding for determining $d_{min}(\kappa, \neg\phi)$.

Definition 3.5. *We define the partial MaxSAT optimization encoding $S^O_{\dot{-}_D}(\kappa, \phi)$, based on definition 3.3, as:*

$$S^O_{\dot{-}_D}(\kappa, \phi) = \Big(\kappa^y \ \wedge\ t(\neg\phi)\ \wedge \bigwedge_{1\leq j\leq n} \Big((d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j)\Big)\Big)_{w=n+1} \wedge \Big(\bigwedge_{1\leq j\leq n} \Big(\neg d_j\Big)\Big)_{w=1}$$

Theorem 3. *From the optimal solution S to the partial MaxSAT encoding $S^O_{\dot{-}_D}(\kappa, \phi)$ the minimum Dalal distance $d_{min}(\kappa, \neg\phi)$ can be obtained by $d_{min}(\kappa, \neg\phi) = \sum_{j=1}^{n} value(S, d_j)$.*

The following example demonstrates the encoding for a specific belief contraction instance.

Example 3.3. *Let $\kappa = x_1 \wedge (\neg x_1 \vee x_2)$ and $\phi = (\neg x_1 \vee x_2) \wedge x_2$. Then*

$$S^O_{\dot{-}_D}(\kappa, \phi) = \Big(y_1 \wedge (\neg y_1 \vee y_2)\ \wedge\ ((\neg a_1 \vee x_1) \wedge (\neg a_1 \vee \neg x_2) \wedge (a_1 \vee \neg x_1 \vee x_2) \wedge (a_1 \vee \neg x_2))\ \wedge\ (d_1 \vee \neg x_1 \vee y_1) \wedge (d_1 \vee x_1 \vee \neg y_1) \wedge (d_2 \vee \neg x_2 \vee y_2) \wedge (d_2 \vee x_2 \vee \neg y_2)\Big)_{w=3} \wedge\ \Big(\neg d_1 \wedge \neg d_2\Big)_{w=1}$$

where a_1 is an auxiliary variable introduced by the Tseitin transformation. Note that the Tseitin transformation often does not pay off for small formulae such as ϕ in this example - without the Tseitin transformation, we could have turned $\neg\phi = \neg((\neg x_1 \vee x_2) \wedge x_2)$ into the simple CNF formula $\neg x_2$ by applying the usual boolean transformation rules. When passing the encoding in WDIMACS format to a partial MaxSAT solver, the solution satisfying the highest possible number of soft clauses satisfies exactly one soft clause, indicating $d_{min}(\kappa, \neg\phi) = 1$.

Again recalling that $Mod(\kappa \dot{-}_D \phi) = Mod(\kappa) \cup Mod(\kappa \dot{+} \neg\phi)$ and now given $d_{min}(\kappa, \neg\phi)$, we proceed with defining the belief change SAT encoding $S_{\dot{-}_D}(\kappa, \phi)$. For this purpose we first establish the following lemma.

Lemma 3. *For the SAT encoding*

$$D = \kappa^y \wedge t(\neg\phi) \wedge \bigwedge_{1 \leq j \leq n} \Big((d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j) \Big) \wedge E(\{d_1, ..., d_n\}, d_{min}(\kappa, \neg\phi))$$

the following holds:

$$proj(Mod(D), \{x_1, ..., x_n\}) = Mod(\kappa \dot{+} \neg\phi)$$

For the encoding $S_{\dot{-}_D}(\kappa, \phi)$ to represent the belief base resulting from Dalal's contraction the relation $proj(Mod(S_{\dot{-}_D}(\kappa, \phi)), \{x_1, ..., x_n\}) = Mod(\kappa \dot{-}_D \phi) = Mod(\kappa) \cup Mod(\kappa \dot{+}_D \neg\phi)$ must hold. This means that in $S_{\dot{-}_D}(\kappa, \phi)$ the variables $\{x_1, ..., x_n\}$ need to represent the models of κ as well as the models of $\kappa \dot{+}_D \neg\phi$. To this end we undertake the below steps:

- We create two sets of new variables $W = \{w_1, ..., w_n\}$ and $Z = \{z_1, ..., z_n\}$.
- We take encoding D from Lemma 3 and replace all variables $\{x_1, ..., x_n\}$ with the corresponding new variable in Z, resulting in

  $$D' = \kappa^y \wedge t(\neg\phi)^z \wedge \bigwedge_{1 \leq j \leq n} \Big((d_j \vee \neg z_j \vee y_j) \wedge (d_j \vee z_j \vee \neg y_j) \Big) \wedge E(\{d_1, ..., d_n\}, d_{min}(\kappa, \neg\phi))$$

  Note that $t(\neg\phi)^z$ denotes $t(\neg\phi)$ after replacing all $x_j \in \{x_1, ..., x_n\}$ with $z_j \in Z$, whereas the auxiliary variables within $t(\neg\phi)$ remain unchanged.

- To D' we add another copy (κ^w) of the belief base CNF formula κ, in which we replace each variable $x_j \in \{x_1, ..., x_n\}$ with the corresponding variable $w_j \in W$:

  $$D'' = D' \wedge \kappa^w = \kappa^y \wedge t(\neg\phi)^z \wedge \bigwedge_{1 \leq j \leq n} \Big((d_j \vee \neg z_j \vee y_j) \wedge (d_j \vee z_j \vee \neg y_j) \Big) \wedge E(\{d_1, ..., d_n\}, d_{min}(\kappa, \neg\phi)) \wedge \kappa^w$$

- Finally, we add another constraint to D'', that reintroduces the original variables $\{x_1, ..., x_n\}$ and ensures that they either adopt the same truth assignments as variables $\{w_1, ..., w_n\}$ or as variables $\{z_1, ..., z_n\}$. We denote this constraint by $F(X, Z, W, n)$ and it is the subject of Lemma 4.

$$F(X,Z,W,n) = \bigwedge_{1\leq j\leq n} \Big(\bigwedge_{\substack{1\leq i\leq n \\ i\neq j}} \Big((x_i \vee \neg z_i \vee x_j \vee \neg w_j) \wedge (\neg x_i \vee z_i \vee x_j \vee \neg w_j) \\ \wedge (x_i \vee \neg z_i \vee \neg x_j \vee w_j) \wedge (\neg x_i \vee z_i \vee \neg x_j \vee w_j) \Big) \\ \wedge (x_j \vee \neg z_j \vee \neg w_j) \wedge (\neg x_j \vee z_j \vee w_j) \Big)$$

Lemma 4. *For the SAT encoding $F(X, Z, W, n)$ the following holds:*

$$proj(Mod(F(X,Z,W,n)), X) \;=\; proj(Mod(F(X,Z,W,n)), Z) \\ \cup\; proj(Mod(F(X,Z,W,n)), W)$$

By combining the above introduced formulae we can now establish the formal definition of $S^{\cdot}_{-_D}(\kappa, \phi)$.

Definition 3.6. *The SAT encoding $S^{\cdot}_{-_D}(\kappa, \phi)$ is defined as*

$$S^{\cdot}_{-_D}(\kappa,\phi) = \kappa^y \;\wedge\; t(\neg\phi)^z \;\wedge \bigwedge_{1\leq j\leq n} \Big((d_j \vee \neg z_j \vee y_j) \wedge (d_j \vee z_j \vee \neg y_j) \Big) \\ \wedge\; E(\{d_1, ..., d_n\}, d_{min}(\kappa, \neg\phi)) \wedge \kappa^w \wedge F(X, Z, W, n)$$

Theorem 4 finally establishes, that $S^{\cdot}_{-_D}(\kappa, \phi)$ can be used for inference and model checks on belief bases resulting from the Dalal contraction operator.

Theorem 4. *For the SAT encoding $S^{\cdot}_{-_D}(\kappa, \phi)$ following relation holds:* $proj(Mod(S^{\cdot}_{-_D}(\kappa, \phi)), \{x_1, ..., x_n\}) = Mod(\kappa \dot{-}_D \phi)$

3.3.4 Satoh's revision (SAT)

In the following, we first present a partial MaxSAT optimization encoding $S^{O}_{+_S}(\kappa, \mu)$, followed by the introduction of the SAT encoding $S_{+_S}(\kappa, \mu)$, that is query-equivalent to the belief base resulting from the execution of Satoh's revision on κ and μ.

Firstly, let us recall from Section 2.6 that Satoh's revision operator is based on determining the minimal sets of the set of all difference sets between all models of κ and all models of μ. The models of Satoh's revision result are those models of μ, for which there exists at least one model of κ, such that the difference set of these two models is a minimal set. Analogous to the encoding generation for Dalal's operators, which required the determination of the minimum Dalal distance, in order to compile Satoh's revision into an encoding we need information on the minimal sets. As will be demonstrated below it is enough to identify those difference sets, that are minimal sets AND

proper subsets of at least one difference set - note that being a minimal set does not entail being a proper subset of some set. To identify precisely those sets, we propose the partial MaxSAT encoding $S^O_{+_S}(\kappa, \mu)$.

For the upcoming definitions we require an alternative representation of difference sets. Definition 2.6 defined a difference set between two interpretations X and Y of a signature as the set containing those atoms, whose truth assignments differ between the two models. Apart from depicting a difference set as a set containing atoms, we can alternatively depict it as a bit vector as in the following example:

Example 3.4. *Let $\{p, q, r\}$ be a signature and let $X = \{111\}$ and $Y = \{010\}$ be interpretations of the signature. The difference set is then $d_s(X, Y) = \{p, r\}$ and can be alternatively depicted by a bit vector: $d^v_s(X, Y) = \{101\}$.*

From now on we denote the bit vector representation of the difference set of interpretations X and Y by $d^v_s(X, Y)$.

The first step in building the encoding $S^O_{+_S}(\kappa, \mu)$ consists in creating a formula, that allows for obtaining all difference sets between $Mod(\kappa)$ and $Mod(\mu)$. Lemma 5 describes such a formula.

Lemma 5. *For the formula*

$$G = \kappa^y \wedge \mu \wedge \bigwedge_{1 \leq j \leq n} \Big((d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j) \\ \wedge (\neg d_j \vee \neg x_j \vee \neg y_j) \wedge (\neg d_j \vee x_j \vee y_j) \Big) \tag{5}$$

where κ^y is a copy of κ, wherein each variable $x_j \in \{x_1, ..., x_n\}$ is replaced by a new variable $y_j \in \{y_1, ..., y_n\}$ and $\{d_1, ..., d_n\}$ are discrepancy variables, the following holds:

$$proj(Mod(G), \{d_1, ..., d_n\}) = D^v_S(\kappa, \mu)$$

where $D^v_S(\kappa, \mu)$ is the set of bit vector representations of all difference sets between $Mod(\kappa)$ and $Mod(\mu)$.

For determining a difference set, that is a proper subset of another difference set, we require a way to compare all difference sets to each other. For this purpose we add to the above defined formula G of Equation 5 a formula G', defined as

$$G' = \kappa^z \wedge \mu^w \wedge \bigwedge_{1 \leq j \leq n} \Big((d'_j \vee \neg w_j \vee z_j) \wedge (d'_j \vee w_j \vee \neg z_j) \\ \wedge (\neg d'_j \vee \neg w_j \vee \neg z_j) \wedge (\neg d'_j \vee w_j \vee z_j) \Big) \tag{6}$$

where κ^z is a copy of κ, wherein each variable $x_j \in \{x_1, ..., x_n\}$ is replaced by a new variable $z_j \in \{z_1, ..., z_n\}$, μ^w is a copy of μ, wherein each variable $x_j \in \{x_1, ..., x_n\}$ is replaced by a new variable $w_j \in \{w_1, ..., w_n\}$ and $\{d'_1, ..., d'_n\}$ are discrepancy variables analogous to the discrepancy variables $\{d_1, ..., d_n\}$ in Equation 5 above.

Lemma 6. *Each model M of the formula $G \wedge G'$, with G and G' as in equations 5 and 6, contains bit vector representations of two difference sets: $proj(M, \{d_1, ..., d_n\}) \in D_S^v(\kappa, \mu)$ and $proj(M, \{d'_1, ..., d'_n\}) \in D_S^v(\kappa, \mu)$. Furthermore, each possible combination of two difference sets of $D_S^v(\kappa, \mu)$ is addressed by a model of $G \wedge G'$.*

From Lemma 6 we learn that formula $G \wedge G'$ can be leveraged for comparing the difference sets in $D_S^v(\kappa, \mu)$ to each other. However, we do not want to compare two identical difference sets, since a set cannot be a proper subset of itself. As a consequence, we need to add a constraint to $G \wedge G'$, which ensures that the difference sets $proj(M, \{d_1, ..., d_n\})$ and $proj(M, \{d'_1, ..., d'_n\})$ are distinct. This can be accomplished by introducing another set of discrepancy variables $\{d''_1, ..., d''_n\}$ with $d''_j = 0$ whenever $d_j = d'_j$ and $d''_j = 1$ whenever $d_j \neq d'_j$, and ensuring that at least one of these new discrepancy variables has the value 1. The corresponding formula and its characteristics is addressed by Lemma 7.

Lemma 7. *Each model M of the formula*

$$\begin{aligned} H = G \wedge G' \wedge \bigwedge_{1 \leq j \leq n} \Big(& (d''_j \vee \neg d_j \vee d'_j) \wedge (d''_j \vee d_j \vee \neg d'_j) \\ & \wedge (\neg d''_j \vee \neg d_j \vee \neg d'_j) \wedge (\neg d''_j \vee d_j \vee d'_j) \Big) \wedge \bigvee_{1 \leq j \leq n} d''_j \end{aligned} \tag{7}$$

contains bit vector representations of two difference sets: $proj(M, \{d_1, ..., d_n\}) \in D_S^v(\kappa, \mu)$ and $proj(M, \{d'_1, ..., d'_n\}) \in D_S^v(\kappa, \mu)$ with $proj(M, \{d_1, ..., d_n\}) \neq proj(M, \{d'_1, ..., d'_n\})$. Furthermore, each possible combination of two distinct difference sets of $D_S^v(\kappa, \mu)$ is addressed by a model of H.

Next, we adjust formula H in such a way that, given a model M of the adjusted formula, the difference set represented by $proj(M, \{d'_1, ..., d'_n\})$ is a proper subset of the difference set represented by $proj(M, \{d_1, ..., d_n\})$. The implementation of this requirement is based on the following observation: the difference set represented by $proj(M, \{d'_1, ..., d'_n\})$ can only be a proper subset of the difference set represented by $proj(M, \{d_1, ..., d_n\})$ if

1. both are distinct, which is already taken care of in formula H, and
2. all elements contained in the difference set represented by $proj(M, \{d'_1, ..., d'_n\})$ are also contained in the difference set represented by $proj(M, \{d_1, ..., d_n\})$.

The last requirement can be addressed by adding to H a constraint $d_j \vee \neg d'_j$ for every $d_j \in \{d_1, ..., d_n\}$, as is established in Lemma 8.

Lemma 8. *Each model M of the formula*

$$I = H \wedge \bigwedge_{1 \leq j \leq n} \Big(d_j \vee \neg d'_j \Big) \tag{8}$$

contains bit vector representations of two difference sets $proj(M, \{d_1, ..., d_n\}) \in D_S^v(\kappa, \mu)$ and $proj(M, \{d'_1, ..., d'_n\}) \in D_S^v(\kappa, \mu)$ with the second one being a proper subset of the first one.

Furthermore, the set $proj(Mod(I), \{d'_1, ..., d'_n\})$ corresponds to the complete set of difference sets in $D^v_S(\kappa, \mu)$, that are proper subsets of some difference set.

Now that we have defined a formula I, whose models allow for determining all sets in $D^v_S(\kappa, \mu)$, that are proper subsets, we create the partial MaxSAT encoding $S^O_{+_S}(\kappa, \mu)$ by turning all clauses belonging to I into hard clauses and adding for each $d'_j \in \{d'_1, ..., d'_n\}$ a soft clause, that requires d'_j to be *false*.

Definition 3.7. *The partial MaxSAT optimization encoding $S^O_{+_S}(\kappa, \mu)$ is defined as*

$$
\begin{aligned}
S^O_{+_S}(\kappa, \mu) &= \Big(I\Big)_{w=n+1} \wedge \Big(\bigwedge_{1\leq j\leq n} \Big(\neg d'_j\Big)\Big)_{w=1} \\
&= \Big(\kappa^y \ \wedge \ \mu \ \wedge \bigwedge_{1\leq j\leq n} \Big((d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j) \\
&\quad \wedge \ (\neg d_j \vee \neg x_j \vee \neg y_j) \wedge (\neg d_j \vee x_j \vee y_j)\Big) \\
&\quad \wedge \kappa^z \ \wedge \ \mu^w \ \wedge \bigwedge_{1\leq j\leq n} \Big((d'_j \vee \neg w_j \vee z_j) \wedge (d'_j \vee w_j \vee \neg z_j) \\
&\quad \wedge \ (\neg d'_j \vee \neg w_j \vee \neg z_j) \wedge (\neg d'_j \vee w_j \vee z_j)\Big) \\
&\quad \wedge \bigwedge_{1\leq j\leq n} \Big((d''_j \vee \neg d_j \vee d'_j) \wedge (d''_j \vee d_j \vee \neg d'_j) \\
&\quad \wedge \ (\neg d''_j \vee \neg d_j \vee \neg d'_j) \wedge (\neg d''_j \vee d_j \vee d'_j)\Big) \wedge \bigvee_{1\leq j\leq n} d''_j \\
&\quad \wedge \bigwedge_{1\leq j\leq n} \Big(d_j \vee \neg d'_j\Big)\Big)_{w=n+1} \wedge \Big(\bigwedge_{1\leq j\leq n} \Big(\neg d'_j\Big)\Big)_{w=1}
\end{aligned}
$$

where $\Big)_{w=x}$ indicate that all clauses contained within are assigned a weight of x.

Theorem 5 finally establishes how a minimal set, that is a proper subset of some difference set, can be determined with the help of encoding $S^O_{+_S}(\kappa, \mu)$.

Theorem 5. *For the optimal solution S to the partial MaxSat encoding $S^O_{+_S}(\kappa, \mu)$ the bit vector $proj(S, \{d'_1, ..., d'_n\})$ corresponds to the bit vector representation of a minimal set of the set of all difference sets $D_S(\kappa, \mu)$, that is also a proper subset of some set in $D_S(\kappa, \mu)$.*

Example 3.5 shows the just described partial MaxSAT encoding for a specific belief revision instance.

Example 3.5. *Let $\kappa = x_1 \wedge (\neg x_1 \vee x_2)$ and $\mu = \neg x_2$. Then*

$$
\begin{aligned}
S^O_{+_S}(\kappa, \mu) = \Big(&y_1 \wedge (\neg y_1 \vee y_2) \ \wedge \ \neg x_2 \ \wedge \ (d_1 \vee \neg x_1 \vee y_1) \wedge (d_1 \vee x_1 \vee \neg y_1) \\
&\wedge \ (\neg d_1 \vee \neg x_1 \vee \neg y_1) \wedge (\neg d_1 \vee x_1 \vee y_1) \wedge (d_2 \vee \neg x_2 \vee y_2)
\end{aligned}
$$

$$
\begin{aligned}
&\wedge (d_2 \vee x_2 \vee \neg y_2) \wedge\ (\neg d_2 \vee \neg x_2 \vee \neg y_2) \wedge (\neg d_2 \vee x_2 \vee y_2) \\
&\wedge z_1 \wedge (\neg z_1 \vee z_2)\ \wedge\ \neg w_2\ \wedge\ (d'_1 \vee \neg w_1 \vee z_1) \wedge (d'_1 \vee w_1 \vee \neg z_1) \\
&\wedge\ (\neg d'_1 \vee \neg w_1 \vee \neg z_1) \wedge (\neg d'_1 \vee w_1 \vee z_1) \wedge (d'_2 \vee \neg w_2 \vee z_2) \\
&\wedge (d'_2 \vee w_2 \vee \neg z_2) \wedge\ (\neg d'_2 \vee \neg w_2 \vee \neg z_2) \wedge (\neg d'_2 \vee w_2 \vee z_2) \\
&\wedge (d''_1 \vee \neg d_1 \vee d'_1) \wedge (d''_1 \vee d_1 \vee \neg d'_1) \wedge\ (\neg d''_1 \vee \neg d_1 \vee \neg d'_1) \\
&\wedge (\neg d''_1 \vee d_1 \vee d'_1) \wedge (d''_2 \vee \neg d_2 \vee d'_2) \wedge (d''_2 \vee d_2 \vee \neg d'_2) \\
&\wedge\ (\neg d''_2 \vee \neg d_2 \vee \neg d'_2) \wedge (\neg d''_2 \vee d_2 \vee d'_2)\ \wedge (d''_1 \vee d''_2) \\
&\wedge (d_1 \vee \neg d'_1) \wedge (d_2 \vee \neg d'_2)\Big)_{w=3} \wedge \Big(\neg d'_1 \wedge \neg d'_2\Big)_{w=1}
\end{aligned}
$$

When passing the encoding in WDIMACS format to a partial MaxSAT solver, the solution satisfying the highest possible number of soft clauses is $x_1 = 0$, $x_2 = 0$, $y_1 = 1$, $y_2 = 1$, $d_1 = 1$, $d_2 = 1$, $z_1 = 1$, $z_2 = 1$, $w_1 = 1$, $w_2 = 0$, $d'_1 = 0$, $d'_2 = 1$, $d''_1 = 1$ *and* $d''_2 = 0$, *which indicates that the set* $\{x_2\}$ *is a minimal set (due to bit vector* 01 *represented by variables* d'_1 *and* d'_2*).*

As described by Algorithm 3.2, the optimization encoding is adjusted after every successful solver call. Denoting the obtained solution model by S, the adjustment consists of adding a new hard clause, that is a disjunction of negated atoms d'_j, for which $value(S, d'_j) = 1$. The goal of this new clause is to ensure that the proper subset p to be detected by the subsequent solver call is also a minimal set. It does so by excluding all those sets, that are proper supersets of the already determined minimal set and therefore cannot be minimal sets, as well as excluding the determined minimal set itself. Set p is then another minimal set, because there exists only one more set (the already determined one), that is a proper subset of some set and that potentially has fewer elements than p, and that set cannot be a proper subset of p due to the newly added hard clause.

Example 3.6. *In Example 3.5 we have determined the minimal set* $\{x_2\}$*, represented by* $proj(S, \{d'_1, d'_2\}) = 01$ *after making one call to a partial MaxSat solver. Following Algorithm 3.2, the next step consists of adjusting the encoding by adding the new hard clause*

$$\neg\, d'_2$$

The next solver call returns no solution model because the adjusted encoding is unsatisfiable. Accordingly, we can conclude that $D_S(\kappa, \mu)$ *contains exactly one set, that is a minimal set and proper subset at the same time, and that is precisely the set* $\{x_2\}$*.*

For creating the belief change encoding $S_{\dot{+}_S}(\kappa, \mu)$, we require the previously determined minimal sets of $D_S(\kappa, \mu)$. In the following we denote this set of minimal sets by $min^p_S(\kappa, \mu)$. Let us first consider the two simpler cases: If $min^p_S(\kappa, \mu)$

1. contains only one element and that element is the empty set, then the empty set is the only minimal set of $D_S(\kappa, \mu)$. In this case, the encoding $S_{\dot{+}_S}(\kappa, \mu)$ is identical to the encoding $S_{\dot{+}_D}(\kappa, \mu)$ of Dalal's revision, with a minimum Dalal distance of 0.

2. is the empty set, then there exists no set in $D_S(\kappa, \mu)$, that is a proper subset of some other set in $D_S(\kappa, \mu)$ and therefore all sets in $D_S(\kappa, \mu)$ are minimal sets. Recalling Definition 2.8 of Satoh's revision operator, this entails that the models of the revised belief base are precisely the models of the revision formula μ. Thus, $S_{+_S}(\kappa, \mu) = \mu$

Let us now address the more complex case, that $min_S^p(\kappa, \mu)$ contains at least one minimal set, that is not the empty set. We start off with the formula G of Equation 5 above.

$$G = \kappa^y \wedge \mu \wedge \bigwedge_{1 \leq j \leq n} \Big((d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j) \\ \wedge (\neg d_j \vee \neg x_j \vee \neg y_j) \wedge (\neg d_j \vee x_j \vee y_j) \Big)$$

From Lemma 5 we know that $proj(Mod(G), \{d_1, ..., d_n\}) = D_S^v(\kappa, \mu)$. The next step consists in ensuring that for each model M, the difference set represented by $proj(M, \{d_1, ..., d_n\})$ is a minimal set of $D_S(\kappa, \mu)$. Note that it is not conceivable to simply add a constraint to G stating that $proj(M, \{d_1, ..., d_n\})$ must be equal to one of the minimal sets in $min_S^p(\kappa, \mu)$. The reason for this lies in the fact that $min_S^p(\kappa, \mu)$ might not be a complete set of all minimal sets of $D_S(\kappa, \mu)$ as it contains only those minimal sets, that are also proper subsets. Hence, we apply a different approach, based on the following idea: the set of non-minimal sets of $D_S(\kappa, \mu)$ contains exactly those sets of $D_S(\kappa, \mu)$, that contain all elements of a set, that is a proper subset and a minimal set, and contain more elements than that set. Accordingly, we can ensure that for each model M, the difference set represented by $proj(M, \{d_1, ..., d_n\})$ is a minimal set of $D_S(\kappa, \mu)$ by adding a constraint $L(\{d_1, ..., d_n\}, min_S^p(\kappa, \mu))$ to G, that ensures that for every determined minimal set $m \in min_S^p(\kappa, \mu)$, the difference set represented by variables $\{d_1, ..., d_n\}$ does not contain all elements of m AND more elements than m at the same time. The last part is important since otherwise the difference set cannot be the set m. Algorithm 3.7 demonstrates the creation of constraint $L(\{d_1, ..., d_n\}, min_S^p(\kappa, \mu))$.
In order to count the number of elements in the difference set $proj(M, \{d_1, ..., d_n\})$, Algorithm 3.5 of Section 3.3.1 is called in line 3, which creates a set of clauses, that implement a binary counter. Subsequently, for every minimal set $m \in min_S^p(\kappa, \mu)$ a formula is created. The formula states that it cannot be the case that a difference set, represented by the variables $\{d_1, ..., d_n\}$, contains all elements of m AND a different number of elements than m. Once the formula is complete, it is stored in a set called $conjunction$. After generating formulae for every determined minimal set, in line 33 the set $conjunction$ is passed to a function, that creates a conjunction of the set's elements and transforms it into CNF. Finally, the union of the obtained clauses and the already determined binary counter clauses is returned. Note that the returned clauses contain new auxiliary variables due to the binary counter encoding and the Tseitin transformation.
After having discussed all necessary parts of the final encoding, we can now establish a formal definition of encoding $S_{+_S}(\kappa, \mu)$.

Algorithm 3.7 Creation of minimal set constraint $L(\{d_1, ..., d_n\}, min_S^p(\kappa, \mu))$

```
1: function CREATECONSTRAINT({d_1, ..., d_n},minimalSets)
2:     clauses ← empty set
3:     counterBits ← COUNT({d_1, ..., d_n}, clauses)
4:     for minimalSet in minimalSets do
5:         e ← |minimalSet|
6:         formula ← "¬(¬("
7:         eBinary ← binary representation of e
8:         eBinary ← reverse sequence of eBinary
9:         conjunction ← empty set
10:        for i = 0 to i = (last index of counterBits) do
11:            counterBit ← element of counterBits at index i
12:            if i != 0 then
13:                formula ← formula + "∧"
14:            end if
15:            if eBinary has index i then
16:                eBit ← element of eBinary at index i
17:                if eBit == 0 then
18:                    formula ← formula + "¬counterBit"
19:                else
20:                    formula ← formula + "counterBit"
21:                end if
22:            else
23:                formula ← formula + "¬counterBit"
24:            end if
25:        end for
26:        formula ← ")"
27:        for m_j in minimalSet do
28:            formula ← formula + "∧ d_j"
29:        end for
30:        formula ← formula + ")"
31:        conjunction ← conjunction ∪ formula
32:    end for
33:    return clauses ∪ TSEITINTRANSFORMATION(conjunction)
34: end function
```

Definition 3.8. *The SAT encoding $S_{\dot{+}_S}(\kappa, \mu)$ is defined as follows:*

for $min_S^p(\kappa, \mu) = \emptyset$:

$$S_{\dot{+}_S}(\kappa, \mu) = \mu$$

for $min_S^p(\kappa, \mu) = \{\emptyset\}$:

$$S_{\dot{+}_S}(\kappa, \mu) = S_{\dot{+}_D}(\kappa, \mu) \quad with \quad d_{min}(\kappa, \mu) = 0$$

for all other cases:

$$S_{\dot{+}_S}(\kappa, \mu) = \kappa^y \wedge \mu \wedge \bigwedge_{1 \leq j \leq n} \Big((d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j) \wedge (\neg d_j \vee \neg x_j \vee \neg y_j) \\ \wedge (\neg d_j \vee x_j \vee y_j) \Big) \wedge L(\{d_1, ..., d_n\}, min_S^p(\kappa, \mu))$$

Theorem 6 establishes that our presented encoding is query-equivalent to the belief base resulting from Satoh's revision.

Theorem 6. *For the SAT encoding $S_{\dot{+}_S}(\kappa, \mu)$ following relation holds: $proj(Mod(S_{\dot{+}_S}(\kappa, \mu)), \{x_1, ..., x_n\}) = Mod(\kappa \dot{+}_S \mu)$*

Example 3.7 shows our proposed encoding for a specific belief revision instance.

Example 3.7. *Considering again the belief revision instance of examples 3.5 and 3.6, for which we have determined $min_S^p(\kappa, \mu) = \{\{x_2\}\}$, the encoding $S_{\dot{+}_S}(\kappa, \mu)$ then looks as follows:*

$$\begin{aligned} S_{\dot{+}_S}(\kappa, \mu) = & y_1 \wedge (\neg y_1 \vee y_2) \wedge \neg x_2 \wedge (d_1 \vee \neg x_1 \vee y_1) \wedge (d_1 \vee x_1 \vee \neg y_1) \\ & \wedge (\neg d_1 \vee \neg x_1 \vee \neg y_1) \wedge (\neg d_1 \vee x_1 \vee y_1) \wedge (d_2 \vee \neg x_2 \vee y_2) \\ & \wedge (d_2 \vee x_2 \vee \neg y_2) \wedge (\neg d_2 \vee \neg x_2 \vee \neg y_2) \wedge (\neg d_2 \vee x_2 \vee y_2) \\ & \wedge (\neg d_1 \vee \neg d_2 \vee a_1) \wedge (d_1 \vee \neg d_2 \vee a_2) \wedge (\neg d_1 \vee d_2 \vee a_2) \\ & \wedge (\neg a_3 \vee a_2) \wedge (\neg a_3 \vee \neg a_1) \wedge (a_3 \vee \neg a_2 \vee a_1) \wedge (\neg d_2 \vee a_3) \end{aligned}$$

where variables a_1 and a_2 are new auxiliary variables part of the binary counter and a_3 is a new auxiliary variable introduced by the Tseitin transformation.

3.3.5 Satoh's contraction (SAT)

In this section we first present the partial MaxSAT optimization encoding $S^O_{\dot{-}_S}(\kappa, \phi)$ for determining the minimal sets $min_S^p(\kappa, \neg\phi)$, followed by the introduction of the SAT encoding $S_{\dot{-}_S}(\kappa, \phi)$, that is query-equivalent to the belief base resulting from the execution of Satoh's contraction on κ and ϕ.

Definition 2.11 defined Satoh's contraction result models as the union of the belief base models and the models of the revision of the belief base with the negation of the contraction formula, i.e.

$$Mod(\kappa \dot{-}_S \phi) = Mod(\kappa) \cup Mod(\kappa \dot{+}_S \neg\phi)$$

Accordingly, in order to create the encoding $S_{\dot{-}_S}(\kappa, \phi)$ of Satoh's contraction, we first need to determine the minimal sets $min_S^p(\kappa, \neg\phi)$. For the identification of $min_S^p(\kappa, \neg\phi)$ we can use the optimization encoding $S^O_{\dot{-}_S}(\kappa, \phi)$ defined in Definition 3.9 below, which is based on our optimization encoding $S^O_{+_S}(\kappa, \phi)$ for Satoh's revision operator.

Definition 3.9. *The partial MaxSAT encoding $S^O_{\dot{-}_S}(\kappa, \phi)$ is defined as*

$$
\begin{aligned}
S^O_{\dot{-}_S}(\kappa, \phi) &= \Big(I'\Big)_{w=n+1} \wedge \Big(\bigwedge_{1\leq j\leq n}\Big(\neg d'_j\Big)\Big)_{w=1} \\
&= \Big(\kappa^y \ \wedge\ t(\neg\phi)\ \wedge \bigwedge_{1\leq j\leq n}\Big((d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j) \\
&\quad \wedge\ (\neg d_j \vee \neg x_j \vee \neg y_j) \wedge (\neg d_j \vee x_j \vee y_j)\Big) \\
&\quad \wedge \kappa^z \ \wedge\ t(\neg\phi)^w \ \wedge \bigwedge_{1\leq j\leq n}\Big((d'_j \vee \neg w_j \vee z_j) \wedge (d'_j \vee w_j \vee \neg z_j) \\
&\quad \wedge\ (\neg d'_j \vee \neg w_j \vee \neg z_j) \wedge (\neg d'_j \vee w_j \vee z_j)\Big) \\
&\quad \wedge \bigwedge_{1\leq j\leq n}\Big((d''_j \vee \neg d_j \vee d'_j) \wedge (d''_j \vee d_j \vee \neg d'_j) \\
&\quad \wedge\ (\neg d''_j \vee \neg d_j \vee \neg d'_j) \wedge (\neg d''_j \vee d_j \vee d'_j)\Big) \wedge \bigvee_{1\leq j\leq n} d''_j \\
&\quad \wedge \bigwedge_{1\leq j\leq n}\Big(d_j \vee \neg d'_j\Big)\Big)_{w=n+1} \wedge \Big(\bigwedge_{1\leq j\leq n}\Big(\neg d'_j\Big)\Big)_{w=1}
\end{aligned}
$$

where I' like I defined in Lemma 8 with the adjustment that each occurrence of formula μ is replaced by $t(\neg\phi)$ (the Tseitin transformation of formula $\neg\phi$) and $\Big(\Big)_{w=x}$ as usual indicate that all clauses contained within are assigned a weight of x.

Analogous to Theorem 5 for Satoh's revision Theorem 7 establishes how $S^O_{\dot{-}_S}(\kappa, \phi)$ can be used for identifying the minimal sets $min_S^p(\kappa, \neg\phi)$.

Theorem 7. *For the optimal solution S to the partial MaxSat optimization encoding $S^O_{\dot{-}_S}(\kappa, \phi)$, the bit vector $proj(S, \{d'_1, ..., d'_n\})$ corresponds to the bit vector representation of a minimal set of the set of all difference sets $D_S(\kappa, \neg\phi)$, that is a proper subset of some set in $D_S(\kappa, \neg\phi)$.*

The adjustment of the optimization encoding after every solver call (see Algorithm 3.2) is identical to the one described in Section 3.3.4 on Satoh's revision operator.

Next, we propose the encoding $S_{\dot{-}_S}(\kappa, \phi)$ of Satoh's contraction, which incorporates the minimal sets $min_S^p(\kappa, \neg\phi)$ determined via Algorithm 3.2, as well as a theorem (Theorem 8) constituting the foundation for using this encoding for inference and model checks on belief bases resulting from Satoh's contraction.

Definition 3.10. *The SAT encoding $S_{\dot{-}_S}(\kappa, \phi)$ is defined as follows:*

for $min_S^p(\kappa, \neg\phi) = \emptyset$:

$$S_{\dot{-}_S}(\kappa, \phi) = \kappa^y \wedge t(\neg\phi)^z \wedge F(X, Y, Z, n)$$

for $min_S^p(\kappa, \neg\phi) = \{\emptyset\}$:

$$S_{\dot{-}_S}(\kappa, \phi) = S_{\dot{-}_D}(\kappa, \phi) \quad with \quad d_{min}(\kappa, \neg\phi) = 0$$

for all other cases:

$$\begin{aligned} S_{\dot{-}_S}(\kappa, \phi) = \kappa^y \ \wedge \ t(\neg\phi)^z \ \wedge \ \bigwedge_{1 \leq j \leq n} \Big(& (d_j \vee \neg z_j \vee y_j) \wedge (d_j \vee z_j \vee \neg y_j) \\ & \wedge \ (\neg d_j \vee \neg z_j \vee \neg y_j) \ \wedge (\neg d_j \vee z_j \vee y_j) \Big) \wedge L(\{d_1, ..., d_n\}, min_S^p(\kappa, \neg\phi)) \\ & \wedge \kappa^w \wedge F(X, Z, W, n) \end{aligned}$$

Theorem 8. *For the SAT encoding $S_{\dot{-}_S}(\kappa, \phi)$ following relation holds: $proj(Mod(S_{\dot{-}_S}(\kappa, \phi)), \{x_1, ..., x_n\}) = Mod(\kappa \dot{-}_S \phi)$*

3.3.6 SAT inference checks

In this section we demonstrate how the above defined belief change SAT encodings can be leveraged to determine, whether the following expression evaluates to *true* for a given belief change operation B:

$$B \models \gamma$$

As already noted in Section 2.1, whenever $\alpha \models \beta$ holds, $Mod(\alpha) \subseteq Mod(\beta)$ holds and vice versa. The second expression can be interpreted as follows: *For every model of formula α, the formula β evaluates to true.* From this follows that if there is at least one model of α, for which $\neg\beta$ evaluates to *true*, then $\alpha \not\models \beta$. If there is no such model, then $\alpha \models \beta$. Based on this insight into the nature of inference, it is possible to create a SAT instance, that consists of one of the above defined encodings and a few additional clauses, whose satisfiability/unsatisfiability indicates whether formula γ can be inferred from the belief base resulting from a belief change operation. Definition 3.11 presents the scheme of such a SAT instance for the different belief change operators addressed in this paper and Theorem 9 establishes, how it can be used for inference checks.

Definition 3.11. *The inference check SAT encoding $S_I(B, \gamma)$ is defined as follows:*

$$S_I(B, \gamma) = S \wedge t(\neg\gamma)$$

where $t(\neg\gamma)$ denotes the CNF formula, that is obtained from carrying out a Tseitin transformation on $\neg\gamma$ and that contains a few new auxiliary variables $V_{\neg\gamma}^t = Var(t(\neg\gamma)) \setminus Var(\gamma)$ not already contained in the SAT encoding and

$$S = S_{\dot{+}_D}(\kappa, \mu) \quad for \quad B = (\kappa \dot{+}_D \mu)$$

$$S = S_{\dot{+}_S}(\kappa, \mu) \quad for \quad B = (\kappa \dot{+}_S \mu)$$
$$S = S_{\dot{-}_D}(\kappa, \phi) \quad for \quad B = (\kappa \dot{-}_D \phi)$$
$$S = S_{\dot{-}_S}(\kappa, \phi) \quad for \quad B = (\kappa \dot{-}_S \phi)$$

Theorem 9. *The inference check SAT encoding $S_I(B, \gamma)$ is unsatisfiable if and only if $B \models \gamma$.*

3.3.7 SAT model checks

The aim of this section is to define a SAT encoding scheme $S_M(B, N)$, similar to the scheme proposed in the previous section for inference checks, that can be used to decide whether the interpretation N is a model of the belief base represented by a belief change operation B. To be precise, the encoding scheme must be usable for determining whether below expression holds:

$$N \models B$$

In order to create such an encoding $S_M(B, N)$, we take one of the above defined SAT encodings, i.e. the one that corresponds to B, and add one additional clause per $x_j \in X$, that consists of only one literal, namely the atom x_j (if $value(N, x_j) = true$) or its negation (if $value(N, x_j) = false$). We denote the conjunction of such a set of additional clauses by $m(N, X)$. Using this expression, we can establish the following formal definition of $S_M(B, N)$ and its corresponding theorem.

Definition 3.12. *The model check SAT encoding $S_M(B, N)$ is defined as follows:*

$$S_M(B, N) = S \wedge m(N, X)$$

with S as in Definition 3.11.

Theorem 10. *The model check SAT encoding $S_M(B, N)$ is satisfiable if and only if $N \in Mod(B)$.*

3.4 ILP encodings

Analogously to Section 3.3, this section contains the definition of the ILP encoding schemes $I^O_{\dot{+}_D}(\kappa, \mu)$, $I_{\dot{+}_D}(\kappa, \mu)$, $I^O_{\dot{-}_D}(\kappa, \phi)$, $I_{\dot{-}_D}(\kappa, \phi)$, $I^O_{\dot{+}_S}(\kappa, \mu)$, $I_{\dot{+}_S}(\kappa, \mu)$, $I^O_{\dot{-}_S}(\kappa, \phi)$ and $I_{\dot{-}_S}(\kappa, \phi)$ generated and used by algorithms 3.1 and 3.2 as well as the ILP inference and model check encodings $I_I(x, y)$ and $I_M(x, y)$ generated and used by algorithms 3.3 and 3.4. All proofs for this section can be found in Appendix A.

3.4.1 Dalal's revision (ILP)

In the following we first present the ILP optimization encoding $I^O_{\dot{+}_D}(\kappa, \mu)$, whose solution indicates the minimum Dalal distance $d_{min}(\kappa, \mu)$, followed by the proposal of the ILP belief change encoding $I_{\dot{+}_D}(\kappa, \mu)$ for Dalal's revision.

Analogous to the partial MaxSAT encoding proposed in Section 3.3.2, three sets of variables are required for the encoding $I^O_{+_D}(\kappa, \mu)$. We denote the first set of ILP variables, that represent the original variables X by $X^I = \{x^I_1, ..., x^I_n\}$, the second one by $Y^I = \{y^I_1, ..., y^I_n\}$ and analogously the third set, the set of discrepancy variables, by $D^I = \{d^I_1, ..., d^I_n\}$. The first part of $I^O_{+_D}(\kappa, \mu)$ thus consists of definitions of these binary variables: $def_b(X^I)$, $def_b(Y^I)$ and $def_b(D^I)$. The variable definition is followed by constraints representing the belief base formula κ, wherein the variables Y^I represent variables X ($con(\kappa, Y^I)$) and by constraints, that implement the revision formula μ with the variables X^I ($con(\mu, X^I)$). The constraints are generated as described in Section 2.8.2 above. Further, we add two constraints per discrepancy variable $d^I_j \in D^I$, that ensure that the variable has a value of 1, whenever its corresponding variables x^I_j and y^I_j differ in their values. This last set of constraints is translated from the SAT encoding schemes $(d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j)$ of Section 3.3.2 as follows:

$$\begin{array}{ll} \text{constraint 1:} & d^I_j + (1 - x^I_j) + y^I_j \geq 1 \\ \text{constraint 2:} & d^I_j + x^I_j + (1 - y^I_j) \geq 1 \end{array}$$

After applying the usual transformation rules of inequality equations we obtain:

$$\begin{array}{ll} \text{constraint 1:} & x^I_j - y^I_j - d^I_j \leq 0 \\ \text{constraint 2:} & y^I_j - x^I_j - d^I_j \leq 0 \end{array}$$

We denote the complete set of these discrepancy constraints by $con_D(D^I, X^I, Y^I)$. Note that $con_D(D^I, X^I, Y^I) = con_D(D^I, Y^I, X^I)$. The final part of $I^O_{+_D}(\kappa, \mu)$ is an optimization constraint to minimize the sum of all discrepancy variables:

$$minimize \sum_{j=1}^{n} \left(d^I_j \right)$$

Now that we have addressed the individual parts of the ILP optimization encoding for Dalal's revision, we can establish a formal definition.

Definition 3.13. *The ILP encoding $I^O_{+_D}(\kappa, \mu)$ is defined as*

$def_b(X^I)$
$def_b(Y^I)$
$def_b(D^I)$
$con(\kappa, Y^I)$
$con(\mu, X^I)$
$con_D(D^I, X^I, Y^I)$
$minimize \sum_{j=1}^{n} \left(d^I_j \right)$

Note that $I^O_{+_D}(\kappa, \mu)$ is essentially a translation of the partial MaxSAT encoding $S^O_{+_D}(\kappa, \mu)$ of Section 3.3.2 into an ILP program. The encoding can be used to determine $d_{min}(\kappa, \mu)$ as described by Theorem 11 below.

Theorem 11. *The optimal value of the optimal solution to $I^O_{+_D}(\kappa, \mu)$ corresponds to the minimum Dalal distance $d_{min}(\kappa, \mu)$.*

Example 3.8 demonstrates the generation of $I^O_{+_D}(\kappa, \mu)$ for a specific belief revision instance, using the encoding language supported by the solver *glpsol*.

Example 3.8. *Let $\kappa = (a \vee \neg b) \wedge b$, $\mu = \neg a$ and thus $Var(\kappa) \cup Var(\mu) = \{a, b\}$. For determining $d_{min}(\kappa, \mu)$ using a glpsol encoding, we define the required binary ILP variables as follows: variables $x1$ and $x2$ represent variables a and b, respectively, in the belief revision formula constraints; $y1$ and $y2$ represent variables a and b, respectively, in the belief base formula constraints; and $d1$ and $d2$ are the discrepancy variables used for indicating whether $x1$ and $y1$ as well as $x2$ and $y2$, have differing truth assignments. The final glpsol encoding then looks as follows:*

```
var x1 binary;
var x2 binary;
var y1 binary;
var y2 binary;
var d1 binary;
var d2 binary;
s.t. baseConstraint1:
     y1 - y2 >= 0;
s.t. baseConstraint2:
     y2 = 1;
s.t. changeConstraint1:
     x1 = 0;
s.t. discrepancyVarsConstraint1:
     y1 - x1 - d1 <= 0;
s.t. discrepancyVarsConstraint2:
     x1 - y1 - d1 <= 0;
s.t. discrepancyVarsConstraint3:
     y2 - x2 - d2 <= 0;
s.t. discrepancyVarsConstraint4:
     x2 - y2 - d2 <= 0;
minimize distance: d1 + d2;
end;
```

When passing the encoding to the glpsol solver, we obtain 1 as optimal value for the minimization constraint, hence $d_{min}(\kappa, \mu) = 1$.

For the creation of the belief change ILP encoding $I_{+_D}(\kappa, \mu)$, we use encoding $I^O_{+_D}(\kappa, \mu)$ and replace the minimization objective with a new constraint, that requires the sum of all discrepancy variables D^I to be equal to the determined minimum distance $d_{min}(\kappa, \mu)$:

$$\sum_{j=1}^{n} \left(d^I_j \right) = d_{min}(\kappa, \mu)$$

This leads us to the below formal definition of $I_{\dot{+}_D}(\kappa, \mu)$ and its accompanying theorem (Theorem 12).

Definition 3.14. *The ILP encoding $I_{\dot{+}_D}(\kappa, \mu)$ is defined as*

$$
\begin{array}{|c|}
\hline
def_b(X^I) \\
def_b(Y^I) \\
def_b(D^I) \\
\hline
con(\kappa, Y^I) \\
con(\mu, X^I) \\
con_D(D^I, X^I, Y^I) \\
\sum_{j=1}^{n} \left(d_j^I\right) = d_{min}(\kappa, \mu) \\
\hline
\end{array}
$$

Again, encoding $I_{\dot{+}_D}(\kappa, \mu)$ is essentially an ILP translation of the SAT encoding $S_{\dot{+}_D}(\kappa, \mu)$ suggested in Section 3.3.2.

Theorem 12. *For the set S of bit vector representations of all solutions to $I_{\dot{+}_D}(\kappa, \mu)$ the following relation holds:*

$$proj(S, X^I) = Mod(\kappa \dot{+}_D \mu)$$

Example 3.9 shows how the belief change ILP encoding for the Dalal revision of Example 3.8 looks like.

Example 3.9. *Assuming the same belief revision instance as in Example 3.8 and having determined $d_{min}(\kappa, \mu) = 1$, the ILP encoding of the resulting belief base looks as follows:*

```
var x1 binary;
var x2 binary;
var y1 binary;
var y2 binary;
var d1 binary;
var d2 binary;
s.t. baseConstraint1:
     y1 - y2 >= 0;
s.t. baseConstraint2:
     y2 = 1;
s.t. changeConstraint1:
     x1 = 0;
s.t. discrepancyVarsConstraint1:
     y1 - x1 - d1 <= 0;
s.t. discrepancyVarsConstraint2:
     x1 - y1 - d1 <= 0;
s.t. discrepancyVarsConstraint3:
     y2 - x2 - d2 <= 0;
```

```
s.t. discrepancyVarsConstraint4:
     x2 - y2 - d2 <= 0;
s.t. distanceConstraint:
     d1 + d2 = 1;
end;
```

3.4.2 Dalal's contraction (ILP)

In this section we introduce the ILP optimization encoding scheme $I^O_{\dot{-}_D}(\kappa, \phi)$ for determining the minimum Dalal distance for Dalal's contraction, as well as the ILP belief change encoding scheme $I_{\dot{-}_D}(\kappa, \phi)$.

Again we point to the fact that the models of Dalal's contraction result are defined by $Mod(\kappa \dot{-}_D \phi) = Mod(\kappa) \cup Mod(\kappa \dot{+}_D \neg\phi)$. Consequently, the first step consists of determining $d_{min}(\kappa, \neg\phi)$ via the encoding $I^O_{\dot{-}_D}(\kappa, \phi)$. For creating this encoding we need a way to translate a negated CNF formula (in this case $\neg\phi$) into a set of ILP constraints. One possible solution is to do a Tseitin transformation on $\neg\phi$ and translate the resulting formula into ILP constraints. However, in order to avoid the call of a Tseitin transformation function, we suggest a different approach, that is based on the following idea: by definition, a CNF formula α is a conjunction of disjunctions, whose disjuncts are literals. For $\neg\alpha$ to evaluate to *true*, at least one of the disjunctions (clauses) in α must evaluate to *false*. Based on this idea, we propose the below approach for transforming the negation of a CNF formula α into a set of ILP constraints:

1. Firstly, we create a binary ILP variable for each variable in $Var(\alpha)$ as well as one auxiliary binary ILP variable per clause in $C(\alpha)$.

2. Per clause in $C(\alpha)$ we create an ILP constraint as follows: a sum of the clause's literals is created, wherein each positive literal is then replaced by the corresponding newly created ILP variable b_i and each negative literal is replaced by $(1 - b_i)$. The resulting expression is then enclosed in brackets. Finally, we add the expression $\div\, m \leq b_a$, where b_a denotes the newly created auxiliary ILP variable for the currently addressed clause and m the number of literals in the clause.

3. Finally, we add a constraint, that ensures that the sum of all auxiliary ILP variables is smaller than or equal to $|C(\alpha)|-1$.

Example 3.10 demonstrates the just described constraint generation process.

Example 3.10. *Let $\alpha = (a \vee \neg b) \wedge (\neg a \vee \neg b \vee c)$ and let a be represented by the ILP variable b_1, b by b_2 and c by b_3. Since α contains 2 clauses, we create the auxiliary ILP variables b_{a1} and b_{a2}. The set of ILP constraints representing $\neg\alpha$ is then as follows:*

$$\begin{array}{ll} \textit{constraint 1:} & (b_1 + (1 - b_2)) \div 2 \leq b_{a1} \\ \textit{constraint 2:} & ((1 - b_1) + (1 - b_2) + b_3) \div 3 \leq b_{a2} \\ \textit{constraint 3:} & b_{a1} + b_{a2} \leq 1 \end{array}$$

In the following we denote the set of constraints, that is obtained by applying the above transformation rules on a CNF formula α and that thus corresponds to the negation of α, by $con_N(\alpha, P, Q)$, where P is the set of binary ILP variables that represent the variables of α within the constraints and Q is the set of auxiliary ILP variables. The below lemma establishes the correctness of the suggested approach.

Lemma 9. *Given a CNF formula α and two sets of binary ILP variables P and Q with $|P|= |Var(\alpha)|$, $|Q|= |C(\alpha)|$ and $P \cap Q = \emptyset$, for the set S of bit vector representations of all solutions to $con_N(\alpha, X, A)$ the following holds: $proj(S, P) = Mod(\neg\alpha)$*

We can now combine the above results and those from Section 3.4.1 to define $I^O_{\dot{-}_D}(\kappa, \phi)$.

Definition 3.15. *The ILP encoding $I^O_{\dot{-}_D}(\kappa, \phi)$ is defined as*

$$
\begin{array}{|c|}
\hline
def_b(A^I) \\
def_b(Z^I) \\
def_b(Y^I) \\
def_b(D^I) \\
\hline
con(\kappa, Y^I) \\
con_N(\phi, Z^I, A^I) \\
con_D(D^I, Z^I, Y^I) \\
\hline
minimize \ \sum_{j=1}^{n} \left(d_j^I\right) \\
\hline
\end{array}
$$

Note that A^I denotes the set of auxiliary variables needed for the negation of ϕ and $Z^I = \{z_1^i, ..., z_n^i\}$ the set of variables, that represent the variables X in ϕ. Theorem 13 states that $d_{min}(\kappa, \neg\phi)$ can be determined from the optimal solution to $I^O_{\dot{-}_D}(\kappa, \phi)$ in the same way as $d_{min}(\kappa, \mu)$ can be determined from the revision optimization encoding $I^O_{+_D}(\kappa, \mu)$ of the previous section.

Theorem 13. *The optimal value of the optimal solution to $I^O_{\dot{-}_D}(\kappa, \phi)$ corresponds to $d_{min}(\kappa, \neg\phi)$.*

Example 3.11 shows the ILP program $I^O_{\dot{-}_D}(\kappa, \phi)$ for a specific belief contraction instance.

Example 3.11. *Let $\kappa = (a \vee \neg b) \wedge c \wedge (b \vee \neg c)$ and $\phi = (a \vee \neg b) \wedge (b \vee \neg c)$. The encoding $I^O_{\dot{-}_D}(\kappa, \phi)$ then looks as follows in the syntax of the glpsol solver*

```
var a1 binary;
var a2 binary;
var y1 binary;
var y2 binary;
var y3 binary;
var z1 binary;
```

```
var z2 binary;
var z3 binary;
var d1 binary;
var d2 binary;
var d3 binary;
s.t. baseConstraint1:
     y1 - y2 >= 0;
s.t. baseConstraint2:
     y3 = 1;
s.t. baseConstraint3:
     y2 - y3 >= 0;
s.t. changeConstraint1:
     (z1 + (1-z2)) / 2 <= a1;
s.t. changeConstraint2:
     (z2 + (1-z3)) / 2 <= a2;
s.t. changeConstraint3:
     a1 + a2 <= 1;
s.t. discrepancyVarsConstraint1:
     y1 - z1 - d1 <= 0;
s.t. discrepancyVarsConstraint2:
     z1 - y1 - d1 <= 0;
s.t. discrepancyVarsConstraint3:
     y2 - z2 - d2 <= 0;
s.t. discrepancyVarsConstraint4:
     z2 - y2 - d2 <= 0;
s.t. discrepancyVarsConstraint5:
     y3 - z3 - d3 <= 0;
s.t. discrepancyVarsConstraint6:
     z3 - y3 - d3 <= 0;
minimize distance: d1 + d2 + d3;
end;
```

Next, we introduce $I_{\dot{-}_D}(\kappa, \phi)$. For creating $I_{\dot{-}_D}(\kappa, \phi)$ we start off with $I^O_{\dot{-}_D}(\kappa, \phi)$, but replace the minimization constraint by a constraint ensuring that the sum of all discrepancy variables in D^I is equal to $d_{min}(\kappa, \neg\phi)$. This is analogous to how the final encoding $I_{\dot{+}_D}(\kappa, \phi)$ for Dalal's revision is created in the previous section. Since Dalal's contraction operator is defined by $Mod(\kappa \dot{-}_D \phi) = Mod(\kappa) \cup Mod(\kappa \dot{+}_D \neg\phi)$, we introduce additional binary ILP variables X^I, B^I, E^I and F^I with $|X^I| = |B^I| = |E^I| = |F^I| = |X|$, as well as two more binary ILP variables g^I_1 and g^I_2. To the existing constraints we add $con(\kappa, B^I)$, $con_D(E^I, X^I, B^I)$ and $con_D(F^I, X^I, Z^I)$, as well as below constraints:

$$\Big(\sum_{j=1}^{n}\Big(e^I_j\Big)\Big) \div n \;\leq g^I_1$$

$$\Big(\sum_{j=1}^{n}\Big(f_j^I\Big)\Big) \div n \ \leq g_2^I$$

$$g_1^I + g_2^I \leq 1$$

This leads to the below formal definition of $I_{\dot{-}_D}(\kappa, \phi)$ and its accompanying theorem.

Definition 3.16. *The ILP encoding $I_{\dot{-}_D}(\kappa, \phi)$ is defined as*

$$\begin{array}{c}
def_b(A^I) \\
def_b(Y^I) \\
def_b(Z^I) \\
def_b(D^I) \\
def_b(X^I) \\
def_b(B^I) \\
def_b(E^I) \\
def_b(F^I) \\
def_b(g_1^I) \\
def_b(g_2^I) \\
\hline
con(\kappa, Y^I) \\
con_N(\phi, Z^I, A^I) \\
con_D(D^I, Y^I, Z^I) \\
con(\kappa, B^I) \\
con_D(E^I, X^I, B^I) \\
con_D(F^I, X^I, Z^I) \\
\sum_{j=1}^{n}\Big(d_j^I\Big) = d_{min}(\kappa, \neg\phi) \\
\Big(\sum_{j=1}^{n}\Big(e_j^I\Big)\Big) \div n \ \leq g_1^I \\
\Big(\sum_{j=1}^{n}\Big(f_j^I\Big)\Big) \div n \ \leq g_2^I \\
g_1^I + g_2^I \leq 1
\end{array}$$

Theorem 14. *For the set S of bit vector representations of all solutions to $I_{\dot{-}_D}(\kappa, \phi)$ the following relation holds:*

$$proj(S, X^I) = Mod(\kappa \mathbin{\dot{-}_D} \phi)$$

3.4.3 Satoh's revision (ILP)

This section introduces the encoding schemes $I^O_{\dot{+}_S}(\kappa, \mu)$ and $I_{\dot{+}_S}(\kappa, \mu)$ for Satoh's revision, which are translations of the already defined SAT encoding schemes $S^O_{\dot{+}_S}(\kappa, \mu)$ and $S_{\dot{+}_S}(\kappa, \mu)$ into ILP. Before starting with the description of these encoding schemes, we firstly translate below SAT encoding scheme of Section 3.3.4

$$(d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j) \wedge \ (\neg d_j \vee \neg x_j \vee \neg y_j) \wedge (\neg d_j \vee x_j \vee y_j)$$

into ILP constraints. After following the translation rules introduced in Section 2.8.2 and applying the usual transformation rules of inequality equations we obtain:

constraint 1:	d_j^I - x_j^I + $y_j^I \geq 0$
constraint 2:	d_j^I + x_j^I - $y_j^I \geq 0$
constraint 3:	d_j^I + x_j^I + $y_j^I \leq 2$
constraint 4:	$-d_j^I$ + x_j^I + $y_j^I \geq 0$

These constraints ensure that $d_j^I = 0$ whenever $x_j^I = y_j^I$ and $d_j^I = 1$ whenever $x_j^I \neq y_j^I$. We denote the complete set of discrepancy constraints for all variables $d_j^I \in D^I$ by $con_D^E(D^I, X^I, Y^I)$.

$I_{+_S}^O(\kappa, \mu)$ starts with a definition of the following sets of binary ILP variables: X^I, Y^I, D^I, YS^I, ZS^I, DS^I and DDS^I with $|X^I|= |Y^I|= |D^I|= |YS^I|= |ZS^I|= |DS^I|= |DDS^I|= n$. The variable definition is followed by the constraints $con(\kappa, Y^I)$, $con(\mu, X^I)$ and $con_D^E(D^I, X^I, Y^I)$. Next, we add the constraints $con(\kappa, YS^I)$, $con(\mu, ZS^I)$ and $con_D^E(DS^I, ZS^I, YS^I)$. As a result variables D^I and DS^I each represent a difference set of $D_S(\kappa, \mu)$. These constraints are followed by $con_D^E(DDS^I, D^I, DS^I)$ and the constraint $\sum_{j=1}^{n} \left(dds_j^I \right) \geq 1$, which ensures that D^I and DS^I do not represent the same difference set. Analogously to the partial MaxSAT optimization encoding for Satoh's revision in Section 3.3.4 we further add for every discrepancy variable $d_j^I \in D^I$ a constraint of the form

$$d_j^I \; - \; ds_j^I \; \geq \; 0$$

and denote the obtained set of constraints by $con_{PS}(D^I, DS^I)$. These constraints ensure that the difference set represented by variables DS^I is a proper subset of the difference set that is represented by variables D^I. Finally, the optimization constraint

$$minimize \; \sum_{j=1}^{n} \left(ds_j^I \right)$$

is added, resulting in the below formal definition of $I_{+_S}^O(\kappa, \mu)$.

Definition 3.17. *The ILP encoding $I_{+_S}^O(\kappa, \mu)$ is defined as*

$$\boxed{\begin{array}{c} def_b(X^I) \\ def_b(Y^I) \\ def_b(D^I) \\ def_b(YS^I) \\ def_b(ZS^I) \\ def_b(DS^I) \\ def_b(DDS^I) \end{array}}$$

$$
\begin{array}{c}
con(\kappa, Y^I) \\
con(\mu, X^I) \\
con_D^E(D^I, X^I, Y^I) \\
con(\kappa, YS^I) \\
con(\mu, ZS^I) \\
con_D^E(DS^I, ZS^I, YS^I) \\
con_D^E(DDS^I, D^I, DS^I) \\
\sum_{j=1}^{n} \left(dds_j^I \right) \geq 1 \\
con_{PS}(D^I, DS^I) \\
\hline
minimize \sum_{j=1}^{n} \left(ds_j^I \right)
\end{array}
$$

Theorem 15. *Given the bit vector representation S of the optimal solution to $I_{+_S}^O(\kappa, \mu)$ the bit vector $proj(S, \{ds_1^I, ..., ds_n^I\})$ represents a minimal set of the set of all difference sets $D_S(\kappa, \mu)$, that is also a proper subset of some set in $D_S(\kappa, \mu)$.*

The adjustment, that is needed after every solver call to prepare the encoding for the subsequent call, consists in adding one more constraint to the ILP encoding, that excludes all proper supersets of the just determined minimal set as well as the minimal set itself. Let S be the optimal solution obtained by the last solver call and $m = proj(S, \{ds_1^I, ..., ds_n^I\})$ the bit vector representation of the determined minimal set. The new constraint is then an inequality equation stating that the sum of all $ds_j^I \in \{ds_1^I, ..., ds_n^I\}$ for which $value(m, ds_j^I) = 1$ must be smaller than or equal to $n - 1$. Note that this adjustment corresponds exactly to the adjustment to the SAT encoding scheme $S_{+_S}^O(\kappa, \mu)$ described in Section 3.3.4. We denote the final set of bit vector representations of the determined minimal sets by $min_S^p(\kappa, \mu)$.

Analogous to the definition of SAT encoding scheme $S_{+_S}(\kappa, \mu)$, for the definition of $I_{+_S}(\kappa, \mu)$ we distinguish between three distinct cases, depending on the content of $min_S^p(\kappa, \mu)$. If $min_S^p(\kappa, \mu)$ contains only the bit vector representing the empty set, then the empty set is the only minimal set of $D_S(\kappa, \mu)$ and $I_{+_S}(\kappa, \mu)$ is equal to $I_{+_D}(\kappa, \mu)$ with a minimum Dalal distance of 0. If $min_S^p(\kappa, \mu)$ is the empty set, then all sets in $D_S(\kappa, \mu)$ are minimal sets and the models of the revised belief base are the models of formula μ. In all other cases encoding $I_{+_S}(\kappa, \mu)$ is as described in the following. We require, apart from the variables X^I, Y^I and D^I with $|X^I| = |Y^I| = |D^I| = n$ two auxiliary ILP variables per $m \in min_S^p(\kappa, \mu)$ and denote the set of these variables by S^I. Further, we need the constraints $con(\kappa, Y^I)$, $con(\mu, X^I)$ and $con_D^E(D^I, X^I, Y^I)$. So far the variables D^I represent the difference sets between κ and μ. As already described in Section 3.3.4 when defining $S_{+_S}(\kappa, \mu)$, it is not sufficient to ensure that the difference set represented by variables D^I corresponds to a set in $min_S^p(\kappa, \mu)$. Instead, we need to ensure that for every $m \in min_S^p(\kappa, \mu)$ the difference set represented by variables D^I does not contain all elements of the set represented by m AND more elements than the set represented by m at the same time. This can be accomplished by adding the below three constraints

for every $m \in min_S^p(\kappa, \mu)$:

$$\text{constraint 1:} \quad \sum_{d_j^I \in D^I,\, value(m, d_j^I)=1} \left(d_j^I\right) \text{ - } (o \text{ - } 1) \leq s_1^I$$

$$\text{constraint 2:} \quad \left(\sum_{d_j^I \in D^I} \left(d_j^I\right) \text{ - } o\right) \div (n \text{ - } o) \leq s_2^I$$

$$\text{constraint 3:} \quad s_1^I + s_2^I \leq 1$$

where s_1^I and s_2^I the auxiliary variables and o the number of 1-bits in m. In the following we denote the set of such constraints for all $m \in min_S^p(\kappa, \mu)$ by $con_{MS}(min_S^p(\kappa, \mu), D^I, S^I)$.

Lemma 10. *Constraints $con_{MS}(min_S^p(\kappa, \mu), D^I, S^I)$ ensure that the set represented by variables D^I is not a proper superset of any of the sets represented by the bit vectors $min_S^p(\kappa, \mu)$.*

Now that all parts of $I_{\dot{+}_S}(\kappa, \mu)$ have been described, we continue with its formal definition and corresponding theorem.

Definition 3.18. *The ILP encoding $I_{\dot{+}_S}(\kappa, \mu)$ is defined as follows:*

for $min_S^p(\kappa, \mu) = \emptyset$:

$def_b(X^I)$
$con(\mu, X^I)$

for $min_S^p(\kappa, \mu) = \{\emptyset\}$:

$$I_{\dot{+}_D}(\kappa, \mu) \quad with \quad d_{min}(\kappa, \mu) = 0$$

for all other cases:

$def_b(X^I)$
$def_b(Y^I)$
$def_b(D^I)$
$def_b(S^I)$
$con(\kappa, Y^I)$
$con(\mu, X^I)$
$con_D^E(D^I, X^I, Y^I)$
$con_{MS}(min_S^p(\kappa, \mu), D^I, S^I)$

Theorem 16. *For the set S of bit vector representations of all solutions to $I_{\dot{+}_S}(\kappa, \mu)$ the following relation holds: $proj(S, X^I) = Mod(\kappa \dot{+}_S \mu)$*

3.4.4 Satoh's contraction (ILP)

The ILP encoding $I^O_{\dot{-}_S}(\kappa, \phi)$ is identical to encoding $I^O_{+_S}(\kappa, \mu)$ apart from the newly introduced sets of variables A^I and AS^I, the replacement of X^I by Z^I and the replacements of $con(\mu, X^I)$ and $con(\mu, ZS^I)$ by $con_N(\phi, Z^I, A^I)$ and $con_N(\phi, ZS^I, AS^I)$.

Definition 3.19. *The ILP encoding $I^O_{\dot{-}_S}(\kappa, \phi)$ is defined as*

$$
\begin{array}{c}
def_b(A^I) \\
def_b(Y^I) \\
def_b(Z^I) \\
def_b(D^I) \\
def_b(AS^I) \\
def_b(YS^I) \\
def_b(ZS^I) \\
def_b(DS^I) \\
def_b(DDS^I) \\
\hline
con(\kappa, Y^I) \\
con_N(\phi, Z^I, A^I) \\
con^E_D(D^I, Z^I, Y^I) \\
con(\kappa, YS^I) \\
con_N(\phi, ZS^I, AS^I) \\
con^E_D(DS^I, ZS^I, YS^I) \\
con^E_D(DDS^I, D^I, DS^I) \\
\sum_{j=1}^{n} \left(dds^I_j \right) \geq 1 \\
con_{PS}(D^I, DS^I) \\
\hline
minimize \sum_{j=1}^{n} \left(ds^I_j \right)
\end{array}
$$

Theorem 17. *For the bit vector representation S of the optimal solution to $I^O_{\dot{-}_S}(\kappa, \phi)$ the bit vector $proj(S, \{ds^I_1, ..., ds^I_n\})$ corresponds to the bit vector representation of a minimal set of the set of all difference sets $D_S(\kappa, \neg\phi)$, that is also a proper subset of some set in $D_S(\kappa, \neg\phi)$.*

The adjustment of $I^O_{\dot{-}_S}(\kappa, \phi)$ after every solver call is identical to the adjustment of $I^O_{+_S}(\kappa, \mu)$ described in the previous section. Definition 3.20 formally defines encoding $I_{\dot{-}_S}(\kappa, \phi)$ and Theorem 18 describes how the same can be used for determining $Mod(\kappa \dot{-}_S \phi)$.

Definition 3.20. *The ILP encoding $I_{\dot{-}_S}(\kappa, \phi)$ is defined as follows:*

for $min^p_S(\kappa, \neg\phi) = \emptyset$:

$def_b(Z^I)$
$def_b(A^I)$
$def_b(X^I)$
$def_b(B^I)$
$def_b(E^I)$
$def_b(F^I)$
$def_b(g_1^I)$
$def_b(g_2^I)$
$con_N(\phi, Z^I, A^I)$
$con(\kappa, B^I)$
$con_D(E^I, X^I, B^I)$
$con_D(F^I, X^I, Z^I)$
$\left(\sum_{j=1}^{n}\left(e_j^I\right)\right) \div n \ \leq g_1^I$
$\left(\sum_{j=1}^{n}\left(f_j^I\right)\right) \div n \ \leq g_2^I$
$g_1^I + g_2^I \leq 1$

for $min_S^p(\kappa, \neg\phi) = \{\emptyset\}$*:*

$$I_{\dot{-}_D}(\kappa, \phi) \quad with \quad d_{min}(\kappa, \neg\phi) = 0$$

for all other cases:

$def_b(A^I)$
$def_b(Y^I)$
$def_b(Z^I)$
$def_b(D^I)$
$def_b(S^I)$
$def_b(X^I)$
$def_b(B^I)$
$def_b(E^I)$
$def_b(F^I)$
$def_b(g_1^I)$
$def_b(g_2^I)$
$con(\kappa, Y^I)$
$con_N(\phi, Z^I, A^I)$
$con_D(D^I, Y^I, Z^I)$
$con(\kappa, B^I)$
$con_D(E^I, X^I, B^I)$
$con_D(F^I, X^I, Z^I)$
$con_{MS}(min_S^p(\kappa, \neg\phi), D^I, S^I)$

$$\begin{gathered}\left(\sum_{j=1}^{n}\left(e_j^I\right)\right) \div n \ \leq g_1^I \\ \left(\sum_{j=1}^{n}\left(f_j^I\right)\right) \div n \ \leq g_2^I \\ g_1^I + g_2^I \leq 1\end{gathered}$$

Theorem 18. *For the set S of bit vector representations of all solutions to $I_{\dot{-}_S}(\kappa, \phi)$ the following relation holds: $proj(S, X^I) = Mod(\kappa \dot{-}_S \phi)$*

3.4.5 ILP inference checks

This section aims to demonstrate how the ILP belief change encodings of the previous sections can be leveraged to determine whether for a given belief change operation B the below expression evaluates to *true*:

$$B \models \gamma$$

In order to do so, we start with the ILP belief change encoding, that corresponds to B, and combine it with constraints implementing the negation of formula γ, as can be seen in below formal definition.

Definition 3.21. *The inference check ILP encoding $I_I(B, \gamma)$ is defined as follows:*

I
$def_b(A_2^I)$
$con_N(\gamma, X^I, A_2^I)$

where A_2^I the set of auxiliary variables required for the negation of γ and

$$\begin{aligned} I = I_{\dot{+}_D}(\kappa, \mu) \quad & for \quad B = (\kappa \dot{+}_D \mu) \\ I = I_{\dot{+}_S}(\kappa, \mu) \quad & for \quad B = (\kappa \dot{+}_S \mu) \\ I = I_{\dot{-}_D}(\kappa, \phi) \quad & for \quad B = (\kappa \dot{-}_D \phi) \\ I = I_{\dot{-}_S}(\kappa, \phi) \quad & for \quad B = (\kappa \dot{-}_S \phi) \end{aligned}$$

As established by Theorem 19 the determination of whether γ can be inferred from the belief change result, then consists in making a single call to an ILP solver.

Theorem 19. *The inference check ILP encoding $I_I(B, \gamma)$ has no solution if and only if $B \models \gamma$.*

3.4.6 ILP model checks

In the following we propose the ILP encoding $I_M(B, N)$ for determining whether the interpretation N is a model of the belief base resulting from a belief change operation B, i.e. whether the below expression evaluates to *true*:

$$N \models B$$

To obtain the encoding $I_M(B, N)$ we start with the ILP belief change encoding, that corresponds to B, and add one additional constraint per $x_j \in X$, consisting of $x_j^I = 1$ (if $value(N, x_j) = true$) and $x_j^I = 0$ (if $value(N, x_j) = false$), where x_j^I is the ILP variable representing variable x_j in the ILP encoding. We denote the set of these constraints by $con_M(N, X^I)$.

Definition 3.22. *The model check ILP encoding $I_M(B, N)$ is defined as follows:*

I
$con_M(N, X^I)$

with I as in Definition 3.21.

Theorem 20. *The model check ILP encoding $I_M(B, N)$ has a solution if and only if $N \in Mod(B)$.*

3.5 ASP encodings

Analogously to sections 3.3 and 3.4, this section introduces the ASP encoding schemes $A^O_{\dot{+}_D}(\kappa, \mu)$, $A_{\dot{+}_D}(\kappa, \mu)$, $A^O_{\dot{-}_D}(\kappa, \phi)$, $A_{\dot{-}_D}(\kappa, \phi)$, $A^O_{\dot{+}_S}(\kappa, \mu)$, $A_{\dot{+}_S}(\kappa, \mu)$, $A^O_{\dot{-}_S}(\kappa, \phi)$ and $A_{\dot{-}_S}(\kappa, \phi)$ generated and used by algorithms 3.1 and 3.2 as well as the ASP inference and model check encodings $A_I(x, y)$ and $A_M(x, y)$ generated and used by algorithms 3.3 and 3.4. As usual, Appendix A contains all proofs of the following lemata and theorems.

Before proposing our ASP encodings we firstly define a few notations required for the subsequent sections. Given an answer set a, a set $I = \{i_1, ...i_i, ...i_n\}$ of positive integers and a predicate p with arity 1, we define the expression $interpretation(a, I, p)$ to denote the bit vector obtained from applying the following rule: for every $i_i \in I$, if $p(i_i) \in a$, we set the bit 1, otherwise bit 0. As an example, let $a' = \{t(1)\}$ and $a'' = \{t(1), t(2)\}$, then $interpretation(a', \{1, 2\}, t/1) = 10$ and $interpretation(a'', \{1, 2\}, t/1) = 11$. Further, let $A = \{a_1, ..., a_m\}$ be a set of answer sets, then $interpretation(A, I, p) = \{interpretation(a_1, I, p), ..., interpretation(a_m, I, p)\}$ (note that since $interpretation(A, I, p)$ is a set, it does not contain any duplicates). In case of $A = \emptyset$, we obtain $interpretation(A, I, p) = \emptyset$.

3.5.1 Dalal's revision (ASP)

In this section we define the ASP encoding $A^O_{\dot{+}_D}(\kappa, \mu)$, that can be used for determining the minimum Dalal distance $d_{min}(\kappa, \mu)$, and the belief change ASP encoding $A_{\dot{+}_D}(\kappa, \mu)$, which encodes the belief base emerging from the operation $\kappa \dot{+}_D \mu$.

Within the ASP logic program we use positive integers to represent propositional atoms. In the same fashion as in the already suggested SAT and ILP encodings, we use two distinct sets of integers when translating κ and μ into ASP language. Accordingly, we first allocate to each atom $x_i \in X$ an integer x of the range $1 \leq x \leq |X|$ such that each atom is represented by exactly one integer within the range and each integer within the

range represents exactly one propositional atom. We denote the integer representing the propositional atom x_i by x_i^I and the set of these integers by X^I. In the same fashion we allocate to each atom x_i an integer y of the range $(|X|+1) \leq y \leq (2*|X|)$ and denote the set of these integers by Y^I.
The first line of the encoding $A^O_{+_D}(\kappa, \mu)$ consists of a choice rule, that uses the interval operator:

$$\{t(1..intMax)\}.$$

$intMax$ is the highest allocated integer, i.e. $intMax = 2*|X|$, and, as already mentioned in Section 2.8.3, the predicate $t/1$ has the meaning that the propositional atom, that is represented by its integer argument, has the truth value *true*.
Next, we introduce a new predicate $r/2$, which states that its arguments represent the same propositional atom, and add for each pair (x_i^I, y_i^I) a fact of the form

$$r(x_i^I, y_i^I).$$

to the encoding. In the remaining part of the paper we denote the entire set of such representation facts by $rFacts[X^I, Y^I]$.
Now that the representation of propositional atoms within the logic program has been addressed, the formulae κ and μ can be translated into ASP constructs. Accordingly, the next part of the encoding consists of the set of integrity constraints, needed to represent CNF formula κ as described in Section 2.8.3. Within these integrity constraints, we use the integers Y^I to represent the propositional atoms X. We denote the resulting set of integrity constraints by $constraints[\kappa, Y^I]$. These are followed by the integrity constraints for formula μ, wherein we use the integers X^I and which are denoted by $constraints[\mu, X^I]$.
Next, we introduce the predicate $d/1$ by adding the below two static rules:

$$d(M) \ :- \ r(M,N),\ t(M),\ not\ t(N).$$
$$d(M) \ :- \ r(M,N),\ not\ t(M),\ t(N).$$

This predicate indicates that the truth assignment to M differs from the truth assignment to N, and that M and N represent the same propositional atom x.
Finally, a minimization construct is added, which tells the answer set solver to look for the answer set with the smallest amount of atoms of the predicate $d/1$:

$$\#minimize\{1, P \ : \ d(P)\}.$$

Definition 3.23 formally defines the just described ASP optimization encoding $A^O_{+_D}(\kappa, \mu)$.

Definition 3.23. *The ASP encoding $A^O_{+_D}(\kappa, \mu)$ is defined as*

$$
\begin{array}{|c|}
\hline
\{t(1..intMax)\}. \\
rFacts[X^I, Y^I] \\
constraints[\kappa, Y^I] \\
constraints[\mu, X^I] \\
\hline
d(M) \ :- \ r(M,N),\ t(M),\ not\ t(N). \\
d(M) \ :- \ r(M,N),\ not\ t(M),\ t(N). \\
\#minimize\{1, P \ : \ d(P)\}. \\
\hline
\end{array}
$$

where $intMax = 2 * |X|$.

Note that white marks the dynamic part of the encoding, that changes with different κ and μ, whereas grey marks the static part. Theorem 21 establishes that the encoding can be used to determine $d_{min}(\kappa, \mu)$.

Theorem 21. *The optimal value of the optimal answer set to the ASP logic program* $A^O_{+_D}(\kappa, \mu)$ *corresponds to the minimum Dalal distance* $d_{min}(\kappa, \mu)$.

Example 3.12 shows a sample $A^O_{+_D}(\kappa, \mu)$ encoding.

Example 3.12. *Let* $\kappa = (a \vee \neg b) \wedge b$ *and* $\mu = \neg a$. *Then* $X = Var(\kappa) \cup Var(\mu) = \{x_1, x_2\} = \{a, b\}$ *and we define* $x^I_1 = 1$, $x^I_2 = 2$, $y^I_1 = 3$ *and* $y^I_2 = 4$. *The encoding* $A^O_{+_D}(\kappa, \mu)$ *then looks as follows:*

$$
\begin{array}{l}
\{t(1..4)\}. \\
r(1,3). \\
r(2,4). \\
:- \ not\ t(3),\ t(4). \\
:- \ not\ t(4). \\
:- \ t(1). \\
d(M) \ :- \ r(M,N),\ t(M),\ not\ t(N). \\
d(M) \ :- \ r(M,N),\ not\ t(M),\ t(N). \\
\#minimize\{1, P \ : \ d(P)\}.
\end{array}
$$

When passing the encoding to an answer set solver, an optimal value of 1 *is determined, thus* $d_{min}(\kappa, \mu) = 1$.

For generating encoding $A_{+_D}(\kappa, \mu)$ we replace the minimization expression in $A^O_{+_D}(\kappa, \mu)$ by

$$:- \ \#count\{P : d(P)\} \ != \ d_{min}(\kappa, \mu).$$

Definition 3.24. *The ASP encoding $A_{\dot{+}_D}(\kappa, \mu)$ is defined as*

$\{t(1..intMax)\}.$ $rFacts[X^I, Y^I]$ $constraints[\kappa, Y^I]$ $constraints[\mu, X^I]$
$d(M) :- r(M,N),\ t(M),\ not\ t(N).$ $d(M) :- r(M,N),\ not\ t(M),\ t(N).$
$:- \#count\{P : d(P)\} != d_{min}(\kappa, \mu).$

where $intMax = 2 * |X|$.

As formally stated in Theorem 22, the encoding represents the new belief base emanating from $\kappa \dot{+}_D \mu$ in that $Mod(\kappa \dot{+}_D \mu)$ can be extracted from the encoding's answer sets.

Theorem 22. *For the set A of all answer sets to the ASP logic program $A_{\dot{+}_D}(\kappa, \mu)$ the following relation holds:*

$$interpretation(A, X^I, t/1) = Mod(\kappa \dot{+}_D \mu)$$

Example 3.13 shows a sample $A_{\dot{+}_D}(\kappa, \mu)$ encoding for a given belief revision instance.

Example 3.13. *Let κ and μ be as in example 3.12. From the same example we know that $d_{min}(\kappa, \mu) = 1$ and thus obtain the following $A_{\dot{+}_D}(\kappa, \mu)$ encoding:*

$$\begin{aligned}
&\{t(1..4)\}.\\
&r(1,3).\\
&r(2,4).\\
&:- not\ t(3),\ t(4).\\
&:- not\ t(4).\\
&:- t(1).\\
&d(M) :- r(M,N),\ t(M),\ not\ t(N).\\
&d(M) :- r(M,N),\ not\ t(M),\ t(N).\\
&:- \#count\{P : d(P)\} != 1.
\end{aligned}$$

3.5.2 Dalal's contraction (ASP)

In this section we define the ASP encodings $A^O_{\dot{-}_D}(\kappa, \phi)$ and $A_{\dot{-}_D}(\kappa, \phi)$ for Dalal's contraction operator.

As Dalal's contraction operator is defined by $Mod(\kappa \dot{-}_D \phi) = Mod(\kappa) \cup Mod(\kappa \dot{+}_D \neg\phi)$, the first step towards encoding Dalal's contraction into ASP is as usual to create an ASP encoding $A^O_{\dot{-}_D}(\kappa, \phi)$ for determining the minimum Dalal distance $d_{min}(\kappa, \neg\phi)$. To do so, we first present the following steps to translate a negated CNF formula $\neg\phi$ into ASP:

1. We translate every clause of ϕ into a cardinality constraint, with a lower limit of 1. As an example, assume $\phi = (a \vee \neg b) \wedge (\neg b \vee c)$, then we obtain the constraints

$$1\{t(1);\ not\ t(2)\}.$$

and

$$1\{not\ t(2);\ t(3)\}.$$

where a is represented by integer 1, b by 2 and c by 3. Each cardinality constraint corresponds to a disjunction, since it states that at least one of the literals need to be satisfied.

2. Next, we create an integrity constraint using the cardinality constraints of step 1, which ensures that not all of them can be satisfied at the same time. For this purpose we first concatenate the cardinality constraints of step 1, using a comma as separator, and denote the resulting concatenation by $cConstraints[\phi, I]$, where I represents the set of integers used as arguments for predicate t/1 to represent propositional atoms in ϕ. The integrity constraint then looks as follows:

$$:-\ cConstraints[\phi, I].$$

This implements the negation, since in the case that at least one cardinality constraint is not satisfied, ϕ is also not satisfied, hence $\neg\phi$. Assuming the same example as in step 1, we obtain as integrity constraint:

$$:-\ 1\{t(1);\ not\ t(2)\},\ 1\{not\ t(2);\ t(3)\}.$$

Encoding $A^{Q}_{-_D}(\kappa, \phi)$ is then obtained by replacing in encoding $A^{Q}_{+_D}(\kappa, \mu)$ the constraints $constraints[\mu, X^I]$ with the integrity constraint $:-\ cConstraints[\phi, X^I]$. as is summarized in Definition 3.25.

Definition 3.25. *The ASP encoding $A^{Q}_{-_D}(\kappa, \phi)$ is defined as*

$\{t(1..intMax)\}.$ $rFacts[X^I, Y^I]$ $constraints[\kappa, Y^I]$ $:-\ cConstraints[\phi, X^I].$
$d(M)\ :-\ r(M,N),\ t(M),\ not\ t(N).$ $d(M)\ :-\ r(M,N),\ not\ t(M),\ t(N).$ $\#minimize\{1, P\ :\ d(P)\}.$

where $intMax = 2 * |X|$.

Theorem 23 states that the determination of $d_{min}(\kappa, \neg\phi)$ from the optimal answer set of $A^{Q}_{-_D}(\kappa, \phi)$ is analogous to that of the corresponding revision optimization encoding $A^{Q}_{+_D}(\kappa, \phi)$ of Section 3.5.1.

Theorem 23. *The optimal value of the optimal answer set to the ASP logic program $A^{O}_{\dot{-}_D}(\kappa, \phi)$ corresponds to the minimum Dalal distance $d_{min}(\kappa, \neg\phi)$.*

Example 3.14 shows $A^{O}_{\dot{-}_D}(\kappa, \phi)$ for a given belief contraction instance.

Example 3.14. *Let $\kappa = (a \vee \neg b) \wedge b$ and $\phi = a \vee b$. Then $X = Var(\kappa) \cup Var(\phi) = \{x_1, x_2\} = \{a, b\}$ and we define $x^I_1 = 1$, $x^I_2 = 2$, $y^I_1 = 3$ and $y^I_2 = 4$. The encoding $A^{O}_{\dot{-}_D}(\kappa, \phi)$ then looks as follows*

$$
\begin{aligned}
&\{t(1..4)\}.\\
&r(1,3).\\
&r(2,4).\\
&:-\ not\ t(3),\ t(4).\\
&:-\ not\ t(4).\\
&:-\ 1\{t(1);\ t(2)\}.\\
&d(M)\ :-\ r(M,N),\ t(M),\ not\ t(N).\\
&d(M)\ :-\ r(M,N),\ not\ t(M),\ t(N).\\
&\#minimize\{1, P\ :\ d(P)\}.
\end{aligned}
$$

with the only answer set {r(1,3), r(2,4), d(1,3), d(2,4), t(3), t(4)}, thus $d_{min}(\kappa, \neg\phi) = 2$.

Next, we proceed with proposing encoding $A_{\dot{-}_D}(\kappa, \phi)$. Recalling once again, that $Mod(\kappa \dot{-}_D \phi) = Mod(\kappa) \cup Mod(\kappa \dot{+}_D \neg\phi)$ we introduce the constant *isRevisionModel* and turn the integrity constraint

$$:-\ cConstraints[\phi, X^I].$$

in encoding $A^{O}_{\dot{-}_D}(\kappa, \phi)$ into the following rule:

$$not\ isRevisionModel\ :-\ cConstraints[\phi, X^I].$$

Furthermore, we replace the minimization statement in encoding $A^{O}_{\dot{-}_D}(\kappa, \phi)$ by the following two rules:

$$isRevisionModel\ :-\ \#count\{P\ :\ d(P)\}\ !=\ 0.$$

$$not\ isRevisionModel\ :-\ \#count\{S\ :\ d(S)\}\ !=\ d_{min}(\kappa, \neg\phi).$$

The resulting encoding $A_{\dot{-}_D}(\kappa, \phi)$ is described in Definition 3.26 below. Theorem 24 further constitutes the contraction counterpart of the above Theorem 22 of Dalal's revision.

Definition 3.26. *The ASP encoding $A_{\dot{-}_D}(\kappa, \phi)$ is defined as*

$\{t(1..intMax)\}.$ $rFacts[X^I, Y^I]$ $constraints[\kappa, Y^I]$
$d(M) :- r(M,N),\ t(M),\ not\ t(N).$ $d(M) :- r(M,N),\ not\ t(M),\ t(N).$ $isRevisionModel :- \#count\{P : d(P)\} != 0.$
$not\ isRevisionModel :- cConstraints[\phi, X^I].$ $not\ isRevisionModel :- \#count\{S : d(S)\} != d_{min}(\kappa, \neg\phi).$

where $intMax = 2 * |X|$.

Theorem 24. *For the set A of all answer sets to the ASP logic program $A_{\dot{-}_D}(\kappa, \phi)$ the following relation holds:*

$$interpretation(A, X^I, t/1) = Mod(\kappa \dot{-}_D \phi)$$

.

Example 3.15 shows a sample $A_{\dot{-}_D}(\kappa, \phi)$ ASP encoding.

Example 3.15. *Let κ and ϕ be as in Example 3.14 with $d_{min}(\kappa, \neg\phi) = 2$, then $A_{\dot{-}_D}(\kappa, \phi)$ looks as follows:*

$$\begin{aligned}
&\{t(1..4)\}.\\
&r(1,3).\\
&r(2,4).\\
&:- not\ t(3),\ t(4).\\
&:- not\ t(4).\\
&d(M) :- r(M,N),\ t(M),\ not\ t(N).\\
&d(M) :- r(M,N),\ not\ t(M),\ t(N).\\
&isRevisionModel :- \#count\{P : d(P)\} != 0.\\
&not\ isRevisionModel :- 1\{t(1);\ t(2)\}.\\
&not\ isRevisionModel :- \#count\{S : d(S)\} != 2.
\end{aligned}$$

3.5.3 Satoh's revision (ASP)

This section introduces the ASP encodings $A^O_{\dot{+}_S}(\kappa, \mu)$ and $A_{\dot{+}_S}(\kappa, \mu)$ for Satoh's revision. In the same fashion as with the previously suggested SAT and ILP Satoh encodings, we first propose the encoding $A^O_{\dot{+}_S}(\kappa, \mu)$, that allows for identifying those difference sets of $D_S(\kappa, \mu)$, that are both minimal sets and proper subsets, followed by the introduction of the ASP encoding $A_{\dot{+}_S}(\kappa, \mu)$ of the belief base resulting from Satoh's revision on κ and μ.

For the encoding $A^O_{+_S}(\kappa, \mu)$ we once again require sets of integers, that represent the propositional atoms in $|X|$. We allocate to each atom $x_i \in X$ an integer x of the range $1 \leq x \leq |X|$ such that each atom is represented by exactly one integer and each integer represents exactly one atom. Again, we denote the integer representing the propositional atom x_i by x_i^I and the set of these integers by X^I. In the same way we allocate to each atom $x_i \in X$ an integer y of the range $(|X|+1) \leq y \leq (2 * |X|)$, an integer w of the range $(2 * |X|+1) \leq w \leq (3 * |X|)$ and an integer z of the range $(3 * |X|+1) \leq z \leq (4 * |X|)$, denoting the sets by Y^I, W^I and Z^I, respectively. The beginning of encoding $A^O_{+_S}(\kappa, \mu)$ is identical to the one of $A^O_{+_D}(\kappa, \mu)$

$$
\begin{aligned}
&\{t(1..intMax)\}.\\
&rFacts[X^I, Y^I]\\
&constraints[\kappa, Y^I]\\
&constraints[\mu, X^I]\\
&d(M) \ :- \ r(M,N),\ t(M),\ not\, t(N).\\
&d(M) \ :- \ r(M,N),\ not\, t(M),\ t(N).
\end{aligned}
$$

except for $intMax = 4 * |X|$. We introduce predicates $r2/2$ and $r3/2$, which work analogously to predicate $r/2$, i.e. they state that their integer arguments represent the same propositional atom. Further, we introduce predicates $d2/1$ and $d3/1$ and use them analogously to predicate $d/1$ as becomes evident in the second part of $A^O_{+_S}(\kappa, \mu)$:

$$
\begin{aligned}
&r2Facts[Z^I, W^I]\\
&constraints[\kappa, W^I]\\
&constraints[\mu, Z^I]\\
&d2(M) \ :- \ r2(M,N),\ t(M),\ not\, t(N).\\
&d2(M) \ :- \ r2(M,N),\ not\, t(M),\ t(N).\\
&r3Facts[X^I, Z^I]\\
&d3(A) \ :- \ r3(A,B),\ d(A),\ not\, d2(B).\\
&d3(A) \ :- \ r3(A,B),\ not\, d(A),\ d2(B).
\end{aligned}
$$

The so far introduced components of $A^O_{+_S}(\kappa, \mu)$ allow for comparing two difference sets in $D_S(\kappa, \mu)$ to each other, since the presence of $d/1$ and $d2/1$ predicate atoms in a given answer set each indicate one difference set in $D_S(\kappa, \mu)$. To ensure that the $d/1$ and $d2/1$ predicate atoms do not both indicate the same difference set, we add the following integrity constraint:

$$:- \ \#count\{A : d3(A)\} = 0.$$

Finally, to ensure that the difference set indicated by predicate atoms $d2/1$ is both a

proper subset of some other difference set and a minimal set of $D_S(\kappa, \mu)$, we add

$$
\begin{aligned}
&\mathit{not}\ d2(B)\ :-\ r3(A,B),\ \mathit{not}\ d(A).\\
&\#\mathit{minimize}\{1, P\ :\ d2(P)\}.\\
&\#\mathit{show}\ \ d2/1.
\end{aligned}
$$

The show construct allows for easier readability of the encoding's answer sets. Definition 3.27 formally summarizes the just described composition of $A^O_{+_S}(\kappa, \mu)$ and Theorem 25 establishes, how the encoding can be used to determine a minimal set of $D_S(\kappa, \mu)$.

Definition 3.27. *The ASP encoding $A^O_{+_S}(\kappa, \mu)$ is defined as*

$\{t(1..intMax)\}.$ $rFacts[X^I, Y^I]$ $constraints[\kappa, Y^I]$ $constraints[\mu, X^I]$
$d(M)\ :-\ r(M,N),\ t(M),\ \mathit{not}\ t(N).$ $d(M)\ :-\ r(M,N),\ \mathit{not}\ t(M),\ t(N).$
$r2Facts[Z^I, W^I]$ $constraints[\kappa, W^I]$ $constraints[\mu, Z^I]$
$d2(M)\ :-\ r2(M,N),\ t(M),\ \mathit{not}\ t(N).$ $d2(M)\ :-\ r2(M,N),\ \mathit{not}\ t(M),\ t(N).$
$r3Facts[X^I, Z^I]$
$d3(A)\ :-\ r3(A,B),\ d(A),\ \mathit{not}\ d2(B).$ $d3(A)\ :-\ r3(A,B),\ \mathit{not}\ d(A),\ d2(B).$ $:-\ \#\mathit{count}\{A : d3(A)\} = 0.$ $\mathit{not}\ d2(B)\ :-\ r3(A,B),\ \mathit{not}\ d(A).$ $\#\mathit{minimize}\{1, P\ :\ d2(P)\}.$ $\#\mathit{show}\ \ d2/1.$

where $intMax = 4 * |X|$.

Theorem 25. *For the optimal answer set a to the ASP logic program $A^O_{+_S}(\kappa, \mu)$, the bit vector $interpretation(a, Z^I, d2/1)$ corresponds to the bit vector representation of a minimal set of the set of all difference sets $D_S(\kappa, \mu)$, that is a proper subset of some set in $D_S(\kappa, \mu)$.*

The adjustment, that is carried out on $A^O_{+_S}(\kappa, \mu)$ after every solver call in preparation for the next one, consists of adding an integrity constraint, which states that all predicate atoms of $d2/1$, which were in the answer set a, cannot be contained all together in the new answer set. As an example, assuming that the answer set a of the first solver call contained the predicate atoms $d2(11)$ and $d2(13)$, then the integrity constraint that is added to the encoding $A^O_{+_S}(\kappa, \mu)$ for the subsequent solver call is $:-\ d2(11), d2(13)$. By doing so, it is ensured that the already determined minimal set and all of its proper

supersets are excluded from the subsequent solver calls. Accordingly, the proper subset p that is to be determined by the next solver call, is also a minimal set since the only other set, that is a proper subset of some set and that potentially contains fewer elements than p, is the already determined minimal set and that set cannot be a proper subset of p due to the newly added integrity constraint. The described adjustment is analogous to the adjustments to the corresponding SAT and ILP encodings of sections 3.3.4, 3.3.5, 3.4.3 and 3.4.4.

Again we denote the final set of determined minimal sets, that are proper subsets of some set, by $min^p_S(\kappa, \mu)$. With regards to the encoding $A_{\dot{+}_S}(\kappa, \mu)$ we distinguish between three different cases, as we did with the SAT and ILP Satoh encodings described above.

1. In case of $min^p_S(\kappa, \mu) = \emptyset$, no set in $D_S(\kappa, \mu)$ is a proper subset and hence all of them are minimal sets. Accordingly, the models of the revised belief base are exactly the models of the revision formula μ and $A_{\dot{+}_S}(\kappa, \mu)$ looks as follows

$$\{t(1..intMax)\}.$$
$$constraints[\mu, X^I]$$

   with $intMax = |X|$.

2. In case of $min^p_S(\kappa, \mu) = \{\emptyset\}$ on the other hand, encoding $A_{\dot{+}_S}(\kappa, \mu)$ is identical to encoding $A_{\dot{+}_D}(\kappa, \mu)$ with a minimum Dalal distance of 0.

3. In all other cases $A_{\dot{+}_S}(\kappa, \mu)$ is almost identical to $A_{\dot{+}_D}(\kappa, \mu)$ with the only difference being the replacement of the integrity constraint by a set of integrity constraints, which we denote by $minSetConstraints[min^p_S(\kappa, \mu), X^I]$ and explain in the following. The set $minSetConstraints[min^p_S(\kappa, \mu), X^I]$ of integrity constraints contains one integrity constraint per minimal set $m \in min^p_S(\kappa, \mu)$. Every integrity constraint consists of $d/1$ predicate atoms for every element in m, that take as argument the integer $x^I \in X^I$, that represents the corresponding propositional atom $x \in X$, combined with an aggregate $\#count\{P : d(P)\}!= |m|$. As an example, let $X = \{a, b, c\}$ be represented by integers $X^I = \{1, 2, 3\}$ in the given order. Given a minimal set $m = \{a, c\}$, the corresponding integrity constraint looks as follows:

$$:- \ d(1), \ d(3), \ \#count\{P : d(P)\} \ != \ 2.$$

Definition 3.28 constitutes the formal definition of encoding $A_{\dot{+}_S}(\kappa, \mu)$ and Theorem 26 describes how it can be leveraged for the determination of $Mod(\kappa \dot{+}_S \mu)$.

Definition 3.28. *The ASP encoding $A_{\dot{+}_S}(\kappa, \mu)$ is defined as follows:*

for $min^p_S(\kappa, \mu) = \emptyset$:

$\{t(1..intMax)\}.$ $constraints[\mu, X^I]$

where $intMax = |X|$.

for $min_S^p(\kappa, \mu) = \{\emptyset\}$*:*

$$A_{+_D}(\kappa, \mu) \quad with \quad d_{min}(\kappa, \mu) = 0$$

for all other cases:

$\{t(1..intMax)\}.$ $rFacts[X^I, Y^I]$ $constraints[\kappa, Y^I]$ $constraints[\mu, X^I]$
$d(M) :- r(M,N),\ t(M),\ not\ t(N).$ $d(M) :- r(M,N),\ not\ t(M),\ t(N).$
$minSetConstraints[min_S^p(\kappa, \mu), X^I]$

where $intMax = 2 * |X|$.

Theorem 26. *For the set A of all answer sets to the ASP logic program* $A_{\dot{+}_S}(\kappa, \mu)$*, the following relation holds:*

$$interpretation(A, X^I, t/1) = Mod(\kappa \dot{+}_S \mu)$$

3.5.4 Satoh's contraction (ASP)

This section closes the definition of our belief change encodings by defining and proving the last two remaining belief change encodings: $A_{\dot{-}_S}^O(\kappa, \phi)$ and $A_{\dot{-}_S}(\kappa, \phi)$

Encoding $A_{\dot{-}_S}^O(\kappa, \phi)$ is the same as $A_{+_S}^O(\kappa, \mu)$, except that $constraints[\mu, X^I]$ are replaced by $:- \ cConstraints[\phi, X^I].$ and $constraints[\mu, Z^I]$ replaced by $:- \ cConstraints[\phi, Z^I].$ as can be seen in Definition 3.29.

Definition 3.29. *The ASP encoding* $A_{\dot{-}_S}^O(\kappa, \phi)$ *is defined as*

$\{t(1..intMax)\}.$ $rFacts[X^I, Y^I]$ $constraints[\kappa, Y^I]$ $:- \ cConstraints[\phi, X^I].$
$d(M) :- r(M,N),\ t(M),\ not\ t(N).$ $d(M) :- r(M,N),\ not\ t(M),\ t(N).$

$$
\begin{array}{|c|}
\hline
r2Facts[Z^I, W^I] \\
constraints[\kappa, W^I] \\
:- \ cConstraints[\phi, Z^I]. \\
\hline
d2(M) \ :- \ r2(M,N), \ t(M), \ not \ t(N). \\
d2(M) \ :- \ r2(M,N), \ not \ t(M), \ t(N). \\
\hline
r3Facts[X^I, Z^I] \\
\hline
d3(A) \ :- \ r3(A,B), \ d(A), \ not \ d2(B). \\
d3(A) \ :- \ r3(A,B), \ not \ d(A), \ d2(B). \\
:- \ \#count\{A : d3(A)\} = 0. \\
not \ d2(B) \ :- \ r3(A,B), \ not \ d(A). \\
\#minimize\{1, P \ : \ d2(P)\}. \\
\#show \ d2/1. \\
\hline
\end{array}
$$

where $intMax = 4 * |X|$.

Theorem 27 further establishes that $A^O_{\dot{-}_S}(\kappa, \phi)$ can be used in the same way as $A^O_{+_S}(\kappa, \mu)$ for determining a minimal set of $D_S(\kappa, \neg\phi)$.

Theorem 27. *For the optimal answer set a to the ASP logic program $A^O_{\dot{-}_S}(\kappa, \phi)$, the bit vector $interpretation(a, Z^I, d2/1)$ corresponds to the bit vector representation of a minimal set of the set of all difference sets $D_S(\kappa, \neg\phi)$, that is a proper subset of some set in $D_S(\kappa, \neg\phi)$.*

The adjustment of $A^O_{\dot{-}_S}(\kappa, \phi)$ after every ASP solver call is identical to the one described in Section 3.5.3. After the successful determination of $min^p_S(\kappa, \neg\phi)$ the encoding $A_{\dot{-}_S}(\kappa, \phi)$ can be generated as described by Definition 3.30. Note that the definition is simply the contraction counterpart to that of $A_{+_S}(\kappa, \mu)$ (Definition 3.28). The set of rules denoted by $minSetRules[min^p_S(\kappa, \neg\phi), X^I]$ contains one rule per minimal set $m \in min^p_S(\kappa, \neg\phi)$. Every rule's right side consists of $d/1$ predicate atoms for every element in m, that take as argument the integer $x^I \in X^I$, that represents the corresponding propositional atom $x \in X$, combined with an aggregate $\#count\{P : d(P)\}! = |m|$. The rules' left side always contains the same expression: $not \ isRevisionModel$. As an example, let $X = \{a, b, c\}$ be represented by integers $X^I = \{1, 2, 3\}$ in the given order. Given a minimal set $m = \{a, c\}$, the corresponding rule looks as follows:

$$not \ isRevisionModel \ :- \ d(1), \ d(3), \ \#count\{P : d(P)\} \ != \ 2.$$

Definition 3.30. *The ASP encoding $A_{\dot{-}_S}(\kappa, \phi)$ is defined as follows:*

for $min^p_S(\kappa, \neg\phi) = \emptyset$:

$$
\begin{array}{|c|}
\hline
\{t(1..intMax)\}. \\
rFacts[X^I, Y^I] \\
\end{array}
$$

$constraints[\kappa, Y^I]$
$d(M) :- r(M,N),\ t(M),\ not\ t(N).$ $d(M) :- r(M,N),\ not\ t(M),\ t(N).$ $isRevisionModel :- \#count\{P : d(P)\} != 0.$
$not\ isRevisionModel :- cConstraints[\phi, X^I].$

where $intMax = 2 * |X|$.

for $min_S^p(\kappa, \neg\phi) = \{\emptyset\}$*:*

$$A_{\dot{-}_D}(\kappa, \phi) \quad with \quad d_{min}(\kappa, \neg\phi) = 0$$

for all other cases:

$\{t(1..intMax)\}.$ $rFacts[X^I, Y^I]$ $constraints[\kappa, Y^I]$
$d(M) :- r(M,N),\ t(M),\ not\ t(N).$ $d(M) :- r(M,N),\ not\ t(M),\ t(N).$ $isRevisionModel :- \#count\{P : d(P)\} != 0.$
$not\ isRevisionModel :- cConstraints[\phi, X^I].$ $minSetRules[min_S^p(\kappa, \neg\phi), X^I]$

where $intMax = 2 * |X|$.

Finally, Theorem 28 states how $Mod(\kappa \dot{-}_S \phi)$ can be determined from $A_{\dot{-}_S}(\kappa, \phi)$.

Theorem 28. *For the set A of all answer sets to the ASP logic program* $A_{\dot{-}_S}(\kappa, \phi)$*, the following relation holds:*

$$interpretation(A, X^I, t/1) = Mod(\kappa \dot{-}_S \phi)$$

3.5.5 ASP inference checks

In this section we demonstrate how to determine, using the ASP encodings defined in the previous sections of this chapter, whether the following expression evaluates to *true* for a given belief change operation B:

$$B \models \gamma$$

Analogous to the SAT inference checks of Section 3.3.6 and the ILP inference checks of Section 3.4.5, we create the inference check ASP encodings by combining the corresponding ASP encoding of the previous sections with an additional constraint, that implements the negation of formula γ.

Definition 3.31. *The inference check ASP encoding $A_I(B, \gamma)$ is defined as follows:*

A
$:-\ cConstraints[\gamma, X^I]$

where $cConstraints[...]$ as defined in Section 3.5.2, X^I as defined in Section 3.5.1, namely the set of integers used to represent the propositional atoms $Var(\kappa) \cup Var(\mu)$ (or $Var(\kappa) \cup Var(\phi)$) within the first part of each above encoding, and

$$\begin{aligned} A = A_{+_D}(\kappa, \mu) \quad & for \quad B = (\kappa \dot{+}_D \mu) \\ A = A_{+_S}(\kappa, \mu) \quad & for \quad B = (\kappa \dot{+}_S \mu) \\ A = A_{-_D}(\kappa, \phi) \quad & for \quad B = (\kappa \dot{-}_D \phi) \\ A = A_{-_S}(\kappa, \phi) \quad & for \quad B = (\kappa \dot{-}_S \phi) \end{aligned}$$

Theorem 29 states that once the encoding $A_I(B, \gamma)$ is generated, it can be determined with a single call to an answer set solver, whether γ can be inferred from the given compiled belief base.

Theorem 29. *For the set A of answer sets to the inference check ASP encoding $A_I(B, \gamma)$ $A = \emptyset$ holds if and only if $B \models \gamma$.*

3.5.6 ASP model checks

Analogous to the SAT and ILP model check encodings of Sections 3.3.7 and 3.4.6, in this section we suggest an ASP encoding $A_M(B, N)$ for determining whether the interpretation N is a model of the belief base resulting from the belief change operation B. Precisely, the encoding $A_M(B, N)$ is intended to be usable for deciding whether the following expression holds:

$$N \models B$$

For creating $A_M(B, N)$, we take from the above defined ASP encodings the encoding, that corresponds to the operation B, and add one fact per $x_i \in X$, stating either $t(x_i^I)$ (if $value(N, x_i) = true$) or $not\ t(x_i^I)$ (if $value(N, x_i) = false$) where $x_i^I \in X^I$ is the integer, that represents the propositional atom x_i in the above defined ASP encodings. The collection of these facts is denoted by $modelFacts[N, X^I]$. We can then establish below formal definition of $A_M(B, N)$.

Definition 3.32. *The model check ASP encoding $A_M(B, N)$ is defined as follows:*

A
$modelFacts[N, X^I]$

with A as in Definition 3.31.

Theorem 30 establishes that given the encoding $A_M(B, N)$ it is possible to determine with a single call to an answer set solver, whether the interpretation N is a model of the compiled belief base.

Theorem 30. *For the model check ASP encoding $A_M(B, N)$ the equation $A \neq \emptyset$, with A denoting the set of all answer sets to the encoding, holds if and only if $N \in Mod(B)$.*

4 Evaluation

The experimental evaluation of our belief change application targets both the compilation and the inference and model checks for each of the four supported change operators. The aim is to determine, by runtime comparisons, which of the implemented preprocessing algorithms and encoding schemes perform overall best with regards to a specific change operator. Moreover, we compare our compilation-based approaches to a naive baseline implementation for each supported operator. Even though we expect the naive approach to perform poorest for all operators, the analysis is worthwhile as the results allow us to better quantify the superiority of our compilation-based approaches. In Section 4.1 we describe our experimental setup, followed by a description of the naive baseline algorithms in 4.2. Finally, we present the results of our experimental analysis in Section 4.3.

4.1 Experimental setup

Due to the lack of official benchmark data for belief revision and contraction operations, we had to create our own data sets [4]. These include two distinct data sets per belief operation, i.e. data sets R1 and R2 for belief revision and data sets C1 and C2 for belief contraction. Every generated belief change instance originates from a SAT instance available in the GBD online database [5], which was developed by Iser and Sinz [IS19] and contains at the time of writing more than 30 000 SAT benchmark instances. In the following each data set is described in terms of its generation and composition.

For data set **C1** we firstly selected, using the GBD database's filtering function, all satisfiable SAT instances, that contain less than 5 000 variables and less than 5 000 clauses and are solvable by the MiniSat solver within one minute. Furthermore, we removed those instances, that were part of the 2022 and 2023 SAT competitions' main tracks as these instances are already included in data set C2, which is described below. Afterwards, there were 151 remaining SAT instances, which constituted our belief bases. In order to create belief contraction instances, we wrote a small program, that selected for each SAT instance (belief base) randomly (uniform distribution) 5%, 15%, 25% and 35% of the clauses. Each set of clauses then constituted a contraction formula. By selecting clauses of the belief base as contraction formula clauses, trivial cases where the contraction formula is not believed, were ruled out. We further ensured that each contraction formula was non-tautological, to avoid cases, in which a contraction is not possible. After applying this process to each selected SAT instance (belief base), we ended up with $151 * 4 = 604$ belief contraction instances within data set C1. In order to obtain formulae for inference checks we once more randomly selected 5% and 25% of each SAT instance's clauses, while ensuring that these were non-tautological. The conjunctions of these clauses then constituted our inference formulae, which are guaranteed to be satisfiable and non-tautological. Further, we randomly generated for each SAT

[4] Available at `https://fernuni-hagen.sciebo.de/s/Ni5FHqP1V2OaQ7k`

[5] `https://benchmark-database.de/`

instance two different interpretations of the instance's signature. Accordingly, we obtained $151 * 2 = 302$ inference formulae and $151 * 2 = 302$ interpretations for model checks.

For creating data set **C2** we selected those satisfiable SAT instances, that were part of the main track benchmark data of the official 2022 and 2023 SAT competitions [6]. Next, we further narrowed down the selected instances by eliminating all those instances, that were not solvable within $180s$ wall clock time by the partial MaxSAT solver MaxHS. The intention was to remove the most difficult SAT instances. The elimination resulted in 67 remaining, satisfiable SAT instances, which constituted our belief bases. The subsequent generation steps were identical to those of data set C1 and resulted in $67 * 4 = 268$ belief contraction instances, $67 * 2 = 134$ satisfiable and non-tautological inference formulae and $67 * 2 = 134$ interpretations for model checks within data set C2.

Data set **R1** was created in a similar way as data set C1. However, instead of selecting satisfiable SAT instances, we selected all unsatisfiable SAT instances, that contain less than 5 000 variables and less than 5 000 clauses and are solvable by the MiniSat solver within one minute. Next, we removed those instances, that were part of the MUS track benchmark data of the official 2011 SAT competition since these instances are already included in data set R2, which is described below. This resulted in 134 unsatisfiable SAT instances. In order to obtain belief revision instances from these SAT instances we randomly divided each instance into two parts, such that the size of one part was exactly 5%, 15%, 25% and 35% of the size of the other part. The larger set of clauses then constituted the belief base and the smaller set the revision formula. Again, this was carried out with the help of a program, which further ensured that the resulting belief base and revision formulae were both satisfiable. The major advantage of this approach is the fact that trivial cases, in which the minimum Dalal distance is 0 or where the empty set is the only minimal set within the set of difference sets, are ruled out as the conjunction of the belief base and revision formulae corresponds to the unsatisfiable SAT instance. Using the described method, we were able to generate revision instances for 132 of the 134 selected unsatisfiable SAT instances, thus resulting in $132 * 4 = 528$ belief revision instances within data set R1. Analogously to the generation of data sets C1 and C2 we further generated $132 * 2 = 264$ satisfiable, non-tautological inference formulae (this time explicitly checking for satisfiability as well) and $132 * 2 = 264$ interpretations for model checks.

Finally, data set **R2** was created by selecting those unsatisfiable SAT instances, that were part of the MUS track benchmark data of the official 2011 SAT competition. This data set was also used as a starting point by Konieczny et al. in their evaluation [KLM17]. After eliminating all those instances, that the partial MaxSAT solver MaxHS was not able to solve within $180\ s$ wall clock time, there were 241 remaining, unsatisfiable SAT instances. From these instances we selected at random 67 to obtain a data set, that is of equal size as its contraction counterpart C2. The generation of belief revision instances, inference formulae and interpretations from our 67 selected unsat-

[6] `http://www.satcompetition.org/`

isfiable SAT instances was analogous to that of data set R1. Eventually, we ended up with $67 * 4 = 268$ belief revision instances, $67 * 2 = 134$ satisfiable, non-tautological inference formulae and $67 * 2 = 134$ interpretations within data set R2.

Table 4.1 shows the composition of the SAT instances, that were used for the generation of our data sets. As can be seen, both the signature sizes, varying between 12 and more than a million, and the clause numbers, which vary between 24 and more than 3 million, are quite distributed. The highest average number of disjuncts per clause, on the other hand, is only 12, hence not exhibiting as much variation as the signature size and the number of clauses.

Data set	Min S	Max S	Avg S	Min C	Max C	Avg C	Min $\varnothing D$	Max $\varnothing D$	Avg $\varnothing D$
C1	15	1 885	389	24	4 983	2 154	2	12	3
C2	240	260 359	74 360	1 026	2 770 239	368 269	2	7	3
R1	12	1 800	245	32	4 200	905	2	6	3
R2	26	1 199 597	113 191	70	3 868 693	357 714	2	3	2

Table 4.1: Composition of SAT instances used for data set generation wrt. signature size (S), clause number (C) and average disjuncts per clause ($\varnothing D$)

Overall, we expect the belief change instances of data sets R2 and C2 to be harder to compile than those of data sets R1 and C1 for three different reasons. The first one is the restriction on the number of variables and clauses that we introduced by selecting SAT instances with less than 5 000 variables and less than 5 000 clauses as bases for data sets R1 and C1. The second reason is the fact that the SAT instances that served as a basis for data sets R2 and C2 were part of official SAT competitions and are therefore supposed to be non-trivial instances. The third and final reason for our assumption lies in the fact that we selected the SAT instances for data sets R1 and C1 based on them being solvable by the MiniSat solver within one minute, whereas the SAT instances of data sets R2 and C2 had to be solvable within three minutes by the solver MaxHS.

All tests mentioned in this paper were run on five identical Google Cloud Compute Engine virtual machines of the machine type *c2-standard-8*, hosting the operating system Ubuntu 22.04. Each machine had 32 Gigabyte RAM available and was based on the CPU platform *Intel Cascade Lake* with a clock frequency of 3.90 GHz. With regards to the external solvers, we used version 4.0.0 of MaxHS, version 1.7.3 of CaDiCal, version 5.0 of the GLPK package, containing glpsol, and version 5.6.2 of clingo. In each test execution of our program we set the maximum available heap space for the JVM to 10 Gigabyte. Furthermore, for each run we passed the flag `-s` to our application in order to skip the validation phase. The application's execution time on each given benchmark instance was measured in wall clock time and a timeout of 600 seconds was set. To reduce the impact of other processes on the measured runtime, the used virtual machines had no GUI and it was ensured that only system-relevant processes were running in parallel with the tests. Furthermore, execution time for each benchmark

instance was measured three times in a non-sequential manner to rule out possible influences of caching and each instance's result presented in Section 4.3 constitutes the mean value of all three executions. If for a given instance one or more of the three executions resulted in a timeout or memory error, no mean value is given and the instance is treated as a timeout or memory error, respectively.

4.2 Naive implementation

To gain a deeper understanding of how well our compilation-based approaches perform in comparison to a naive implementation of inference and model checks for the targeted belief change operators, we implemented the following naive algorithms:

- *Dalal's revision*: The algorithm first determines the belief base's and change formula's models through iterative calls to the CaDiCal solver. Subsequently, the algorithm compares each belief base model to each change formula model in order to determine the minimum Dalal distance. Finally, it returns all those models of the change formula, that have the determined minimum Dalal distance to at least one belief base model.

- *Dalal's contraction*: This algorithm is very similar to the algorithm for Dalal's revision, but the change formula is first negated, using the Tseitin transformation, and the list of determined result models is expanded by the models of the belief base.

- *Satoh's revision*: The algorithm starts with determining the belief base's and change formula's models using CaDiCal, as well as the difference sets between each of them. Next, it checks for every determined difference set, whether there is another difference set, that is a proper subset of it. That way, all minimal sets are obtained. Finally, it once more compares each belief base model to each change formula model and checks whether their difference set is one of the previously determined minimal sets. If yes, the corresponding change formula model is added to the list of result models, which is returned once all comparisons have been carried out.

- *Satoh's contraction*: This algorithm is very similar to the algorithm for Satoh's revision, but the change formula is first negated, using the Tseitin transformation, and the list of determined result models is expanded by the models of the belief base.

- *Inference check*: As the above naive belief change algorithms return the list of models of the new belief base, an inference check can be carried out by checking, whether the inference formula holds for every model of the new belief base. For this purpose, the algorithm carries out iterative calls to CaDiCal, with each call checking whether the inference formula holds for a specific model. If the inference formula holds in all models, it is believed in the new belief base.

- *Model check*: This algorithm simply consists of a check, whether the provided model is contained in the list of models of the new belief base, that was returned by one of the above naive belief change algorithms.

4.3 Results and discussion

In this section we present and discuss the results of our experimental analysis. We begin with the compilation results, followed by those of the inference and model checks. Afterwards, we describe the further investigations that were carried out as a response to the results of the previous sections. The section closes with a comparison of our compilation-based approach to the naive implementation.

For visualizing and comparing runtime results, we sort for every evaluated algorithm instance the runtimes measured for all belief change instances of a data set from low to high and then plot them. Accordingly, the highest x-value of an algorithm instance's graph indicates the number of belief change instances that were solved without timeout or error by the given algorithm instance. Such kind of plots are referred to as *cactus plots*. Note that, as mentioned in Section 4.1 above, all plotted runtimes constitute the mean value of three non-sequential runs.

4.3.1 Compilation

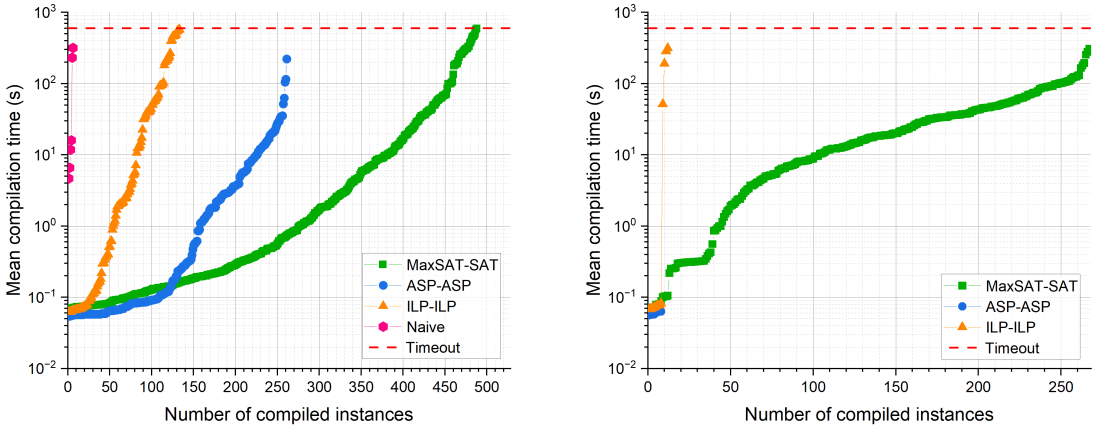
The first set of tests consisted of measuring for each data set and belief change operator the compilation runtimes of the following compilation algorithm instances:

- MaxSAT-SAT - using MaxSAT for the preprocessing and compiling the instance into a SAT encoding
- ASP-ASP - using ASP for the preprocessing and compiling the instance into an ASP encoding
- ILP-ILP - using ILP for the preprocessing and compiling the instance into an ILP encoding
- Naive - using the naive implementation, which generates the list of models of the belief base emerging from the belief change operation

The reason for starting off with algorithm instances that use the same technology for both preprocessing and compilation was the assumption that these instances might yield shorter runtimes than the mixed-technology implementations due to the re-use of parts of the preprocessing encoding in the final compilation result.

The instances of data sets R1 and C1 were tested for both Dalal's and Satoh's operators, whereas the instances of data sets R2 and C2 were tested for Dalal's operators only. Moreover, due to its poor performance in data sets R1 and C1, the naive implementation was no longer considered for data sets R2 and C2.

Figure 4.1 shows the mean compilation times for Dalal's revision operator. It becomes immediately apparent that the naive approach, which attempts to determine



(a) Data set R1

(b) Data set R2

Figure 4.1: Compilation runtime for Dalal's revision with a timeout of 600 s

the minimum Dalal distance by first determining and then comparing the belief base's and change formula's models, performed poorest, successfully terminating in only six out of 528 cases (data set R1). The algorithm instance MaxSAT-SAT on the other hand clearly outperformed all other instances with regards to both data sets as it by far compiled the biggest number of belief change instances. To be precise, it successfully compiled 488 of 528 instances in data set R1 and 267 of 268 instances in data set R2 within the allocated time of 600 seconds. Whereas both the ASP-ASP and ILP-ILP algorithm instances achieved to compile a reasonable amount of instances of data set R1 (261 and 133 respectively), they showed an extremely poor performance with regards to data set R2, compiling only a handful of instances without timing out or encountering an error. However, it is interesting to note that both algorithm instances accomplished to compile a few belief change instances in less time than the MaxSAT-SAT instance. Having a look at Figure 4.2, which shows the results for Satoh's revision on data set R1, we observe that the naive method's performance on Satoh's revision is similar to that on Dalal's revision, with only four out of 528 successfully solved instances. The algorithm instance ILP-ILP compiled 27 belief change instances, constituting a worse performance than the one for Dalal's revision on the same data set. The MaxSAT-SAT and ASP-ASP algorithm instances were able to compile 328 and 344 belief change instances, respectively, the ASP-ASP instance hence performing better than its MaxSAT-SAT counterpart. Further, not only did the ASP-ASP instance compile more instances, but it also compiled many of them in less time than the MaxSAT-SAT instance. It should be noted however, that the runtime differences are only minor, and that it is not possible to identify the exact number of instances for which ASP-ASP outperformed MaxSAT-SAT in the given graphs due to the nature of cactus plots.

Overall, we can conclude from the above compilation results for belief revision, that the algorithm instance MaxSAT-SAT was the overall best performer as it outperformed

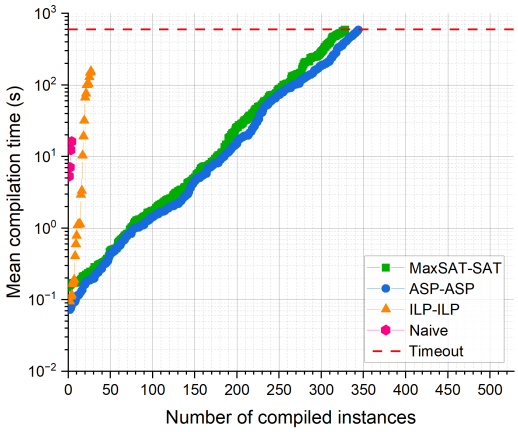
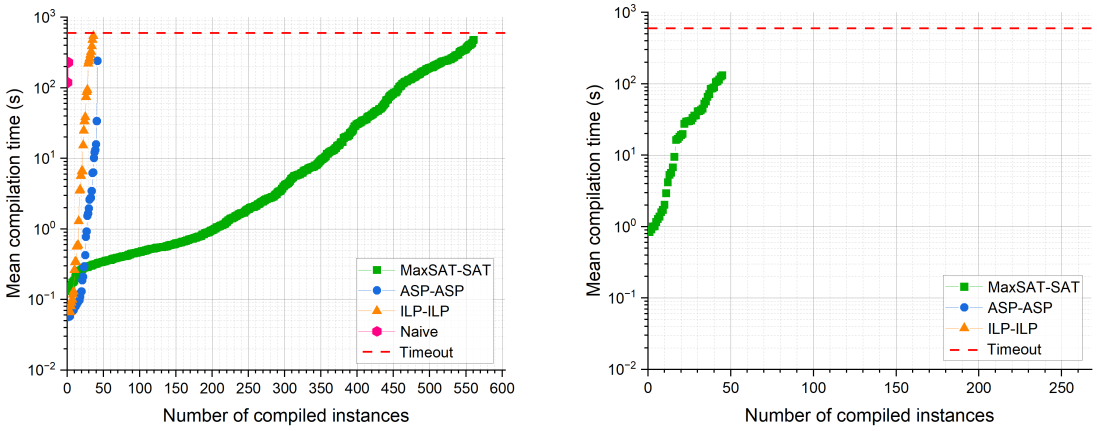


Figure 4.2: Compilation runtime for Satoh's revision with a timeout of 600 s. Data set R1.

the other instances by far with regards to Dalal's revision operator and performed only slightly worse than the ASP-ASP algorithm instance on Satoh's revision operator. Furthermore, we take note that regarding belief revision, the ASP-ASP algorithm instance performed better for Satoh's revision than it did for Dalal's revision, whereas the MaxSAT-SAT and ILP-ILP algorithm instances exhibited the opposite behaviour. From comparing R1 and R2 results for Dalal's revision we can further conclude that, as expected, the instances of data set R2 were overall harder to compile for all three algorithm instances than those of data set R1.



(a) Data set C1 (b) Data set C2

Figure 4.3: Compilation runtime for Dalal's contraction with a timeout of 600 s

Figure 4.3 depicts the mean compilation times for Dalal's contraction operator on data sets C1 and C2. Once more, the algorithm instance MaxSAT-SAT clearly outper-

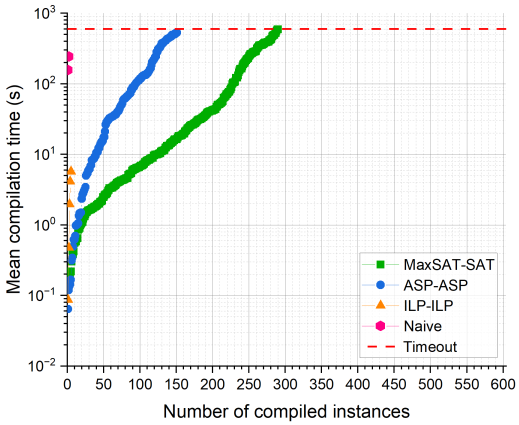


Figure 4.4: Compilation runtime for Satoh's contraction with a timeout of 600 s. Data set C1.

forms the other instances, with the degree of out-performance being even higher than for Dalal's revision. For data set C1 the MaxSAT-SAT instance successfully compiled 561 out of 604 belief change instances within the given time frame, whereas the ILP-ILP and ASP-ASP algorithm instances accomplished the compilation of only 36 and 42 contraction instances, respectively. The naive implementation on the other hand was successful for two contraction instances only. Regarding data set C2 only MaxSAT-SAT was able to compile belief contraction instances without encountering an error or time-out, precisely it compiled 45 out of 268 instances. The compilation results for Satoh's contraction on data set C1, depicted in Figure 4.4, are similar to those of Dalal's contraction on the same data set, albeit with a better performance of the ASP-ASP instance and a poorer performance of the MaxSAT-SAT algorithm instance. The former successfully compiled 151 and the latter 290 out of the 604 belief contraction instances of data set C1. The ILP-ILP instance and the naive implementation were once again the poorest performers with five and two solved instances, respectively. As for Dalal's and Satoh's revision on data set R1, it can be noted that for both Dalal's and Satoh's contraction on data set C1 the ASP-ASP and ILP-ILP algorithm instances solved a few belief change instances in less time than MaxSAT-SAT. Overall, we can confirm our above assumption that the belief change instances of data sets R2 and C2 are harder to compile than those of data sets R1 and C1 since all three algorithm instances performed better on data set C1 than on data set C2. Further, it can be observed once again that the algorithm instance ASP-ASP performs better for Satoh's operator than it does for Dalal's operator, whereas for both MaxSAT-SAT and ILP-ILP the opposite applies.

From the described experimental results it can be concluded that for both belief revision and belief contraction the algorithm instance MaxSAT-SAT performs overall best. It is important to stress however, that it was slightly outperformed by the ASP-ASP instance with regards to Satoh's revision. The naive approach performed poorest for both revision and contraction as it solved only a very few instances and did not accomplish

to outperform any of our algorithm instances with the exception of one instance of data set R1, where ILP-ILP was slowest for Satoh's revision. The second best performer for both belief revision and contraction was the ASP-ASP instance, followed by ILP-ILP.

In order to gain a deeper understanding of the reasons for compilation failure we further analyzed for each evaluated algorithm instance the composition of timeouts and errors. The results are summarized in the bar chart of Figure 4.5. The data show that for all three algorithm instances preprocessing timeouts constituted the major reason for compilation failure. In fact, most detected preprocessing timeouts were timeouts that occurred during the solver calls, which generally raises the question of whether solver calls also consume a huge part of the runtime of successfully compiled instances. For the ILP-ILP algorithm instance the second largest reason were timeouts in the generation of the belief change encoding, followed by a very small number of preprocessing errors that occurred during the glpsol solver calls. The second largest reason for failure for the algorithm instance ASP-ASP were preprocessing errors, to be precise grounding errors of clingo, followed by a very few encoding generation timeouts that are hardly visible in the chart. Finally, for the MaxSAT-SAT instance encoding generation errors - all of them out-of-memory errors - constituted the second largest reason for failure, followed by encoding generation timeouts. Due to the significantly large amount of preprocessing timeouts, it is evident that alternate algorithm instances (those constituting mixtures of different technologies, such as ASP-SAT) cannot be expected to perform significantly better than the evaluated algorithm instances with regards to the amount of successfully compiled belief change instances. However, the question remains whether they could perform significantly better with respect to compilation runtime. To answer both this question and the above question of how much runtime the solver calls actually consume, we computed for each belief change operator and algorithm instance the average ratio of solver time to total runtime. For Dalal's operators the solver time corresponds exactly to the time passed between the start and end of the solver call, whereas for Satoh's operators the solver time corresponds to the total time elapsed from the first to the last solver call, including the time needed for evaluating the solver results and adjusting the optimization encoding. Further, we differentiated between three different cases:

1. Instances with a total runtime of 500 ms and less
2. Instances with a total runtime of 1 000 ms and less
3. Instances with a total runtime of more than 1 000 ms

Figure 4.6 and Figure 4.7 show the results for belief revision and belief contraction, respectively. Data in both mentioned figures show that the higher the overall compilation runtime of a belief change instance, the larger the share of the solver call time. For belief change instances with a total runtime of more than 1 000 ms the percentage of the solver call time is far above 90 percent, in most cases even close to or higher than 99 percent. The only exception to this observation is the solver call time of MaxSAT-SAT for Dalal's contraction, which constitutes 82 percent of the total runtime. From the

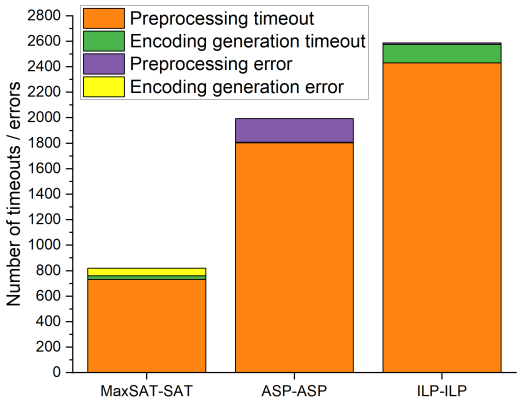


Figure 4.5: Composition of total compilation timeouts and errors per algorithm instance

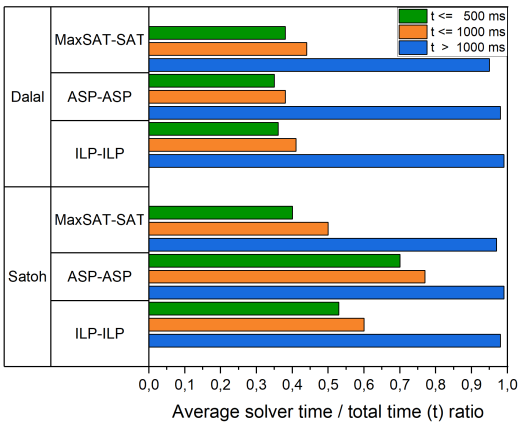


Figure 4.6: Average solver time / total time ratio of revision instances

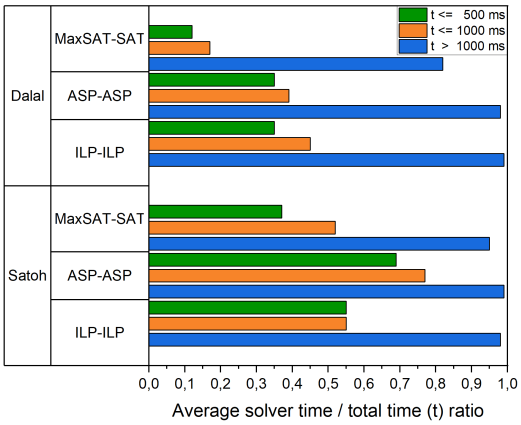


Figure 4.7: Average solver time / total time ratio of contraction instances

described observations it can be concluded that for belief change instances with a total compilation runtime of more than 1 000 ms alternate algorithm instances cannot be expected to perform significantly better than the already evaluated ones with regards to compilation runtime, the reason being the high percentage of the solver call time. For belief change instances with a total compilation runtime of 1 000 ms and less, alternate algorithm instances could potentially perform better. However, even if they do so, the time improvements can only be fractions of a second due to the already short compilation time.

Now that we have elaborated that the solver calls consume a considerable amount of the total compilation runtime, a possible explanation for the extremely good performance of the MaxSAT-SAT algorithm instance as opposed to its ASP and ILP counterparts might lie in the specific focus of the partial MaxSAT technology on optimization problems. Even though both ASP and ILP solvers support the solving of optimization problems, partial MaxSAT solvers were designed and optimized specifically for this kind of problems.

Despite the above conclusions regarding the mixed-technology algorithm instances, we decided to conduct another set of compilation tests, using the algorithm instance MaxSAT-ASP, due to the following observation: despite being the best-performing algorithm instance, MaxSAT-SAT failed for some instances in the encoding generation (out-of-memory errors and timeouts) whereas almost none of the ASP-ASP instances failed in this step (see Figure 4.5). This lead us to the question, whether MaxSAT-ASP could compile more instances than MaxSAT-SAT by combining the speed of partial MaxSAT in the preprocessing with the stability of ASP in the encoding generation. The results of this analysis along with a few additional insights into the relative compilation runtimes are presented and discussed in Section 4.3.4. Next, we continue with the presentation of the inference check results.

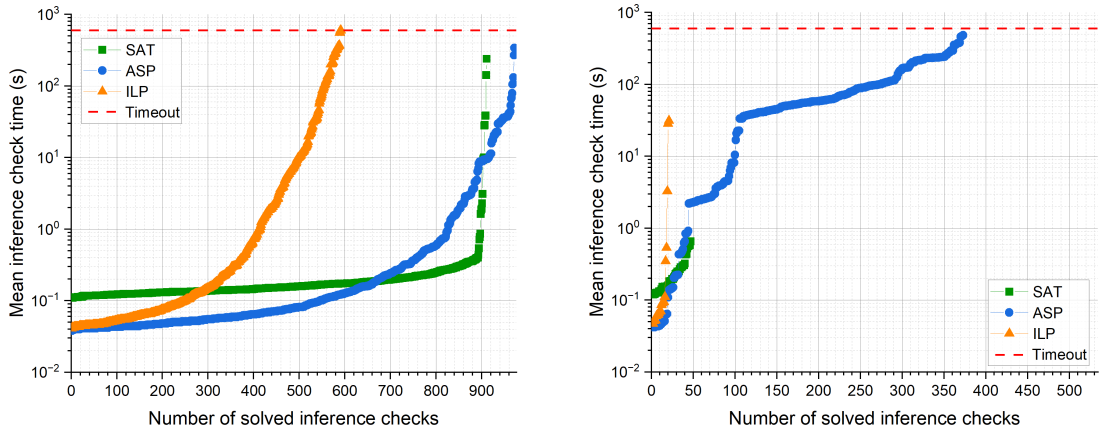
4.3.2 Inference checks

For the inference and model check tests we selected for each data set and operator the SAT encodings that were successfully generated by the algorithm instance MaxSAT-SAT during the above described compilation tests. To ensure an equal number of SAT, ASP, and ILP encodings, we generated the missing corresponding ASP and ILP encodings using the algoritm instances MaxSAT-ASP and MaxSAT-ILP, respectively. This resulted in the following number of encodings for the inference and model checks: 488 per encoding type (SAT, ASP, ILP) from data set R1 for Dalal's revision and 328 for Satoh's revision; 267 per encoding type from data set R2 for Dalal's revision; 560 per encoding type from data set C1 for Dalal's revision and 290 for Satoh's revision; and finally 45 per encoding type from data set C2 for Dalal's revision. Since - as described in Section 4.1 - two inference formulae and two interpretations were generated for each of the SAT instances used as a starting point for our data sets, two inference and two model checks were run for each encoding. Note that for Dalal's contraction on data set C1 there were only 560 encodings per type, even though the MaxSAT-SAT algorithm

instance was able to compile 561 belief change instances during our compilation tests. The reason for this is that we were unable to create ASP and ILP encodings for Dalal's contraction for one instance of data set C1 and as a result this instance was neglected in the inference and model checks.

As the naive implementation performed very poorly in our compilation tests in that it achieved to generate model lists for only a handful of belief change instances, the naive inference and model check tests were limited to these few instances. As a consequence, the inference and model check results for the naive implementation are not considered in the current and subsequent section, but instead addressed in Section 4.3.5.

Before beginning the actual analysis of the obtained inference check results, it should be noted that in contrast to the compilation tests, solely a negligibly small number of inference check tests terminated with an error. Accordingly, almost all unsuccessful inference checks encountered a timeout. Figure 4.8 shows the inference check results for Dalal's revision. For both data sets R1 and R2 the ILP encodings performed overall worst as they succeeded in only 591 out of 976 inference checks for data set R1 and in only 21 out of 534 inference checks for data set R2. The SAT and ASP encodings' performances on data set R1 are both extremely positive with 972 and 911 solved inference checks, respectively. Having a look at the results for data set R2 however, a different behaviour is visible. Using the SAT encodings only 47 of 534 model checks were solved, whereas the ASP model checks succeeded in 373 cases. The cactus plot of Figure 4.9, which plots the results of Satoh's revision on data set R1, is very similar to that of Dalal's revision on data set R1 (Figure 4.8) in that the ILP encoding inference checks successfully terminated in about fifty percent of all cases, whereas the SAT and ASP encodings allowed for solving all (ASP) or almost all (SAT) inference checks within the given time of 600 seconds.



(a) Data set R1

(b) Data set R2

Figure 4.8: Inference check runtime for Dalal's revision with a timeout of 600 s

With regards to Dalal's contraction operator (see Figure 4.10), the ILP encodings once

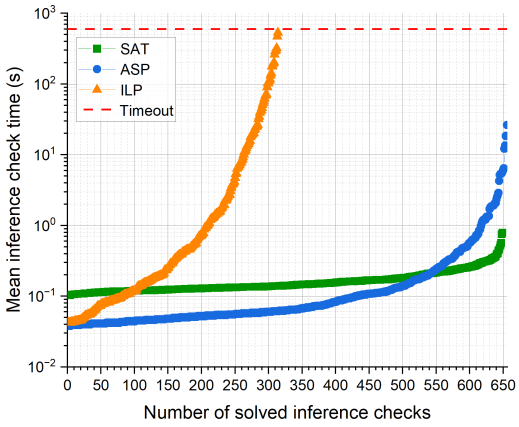
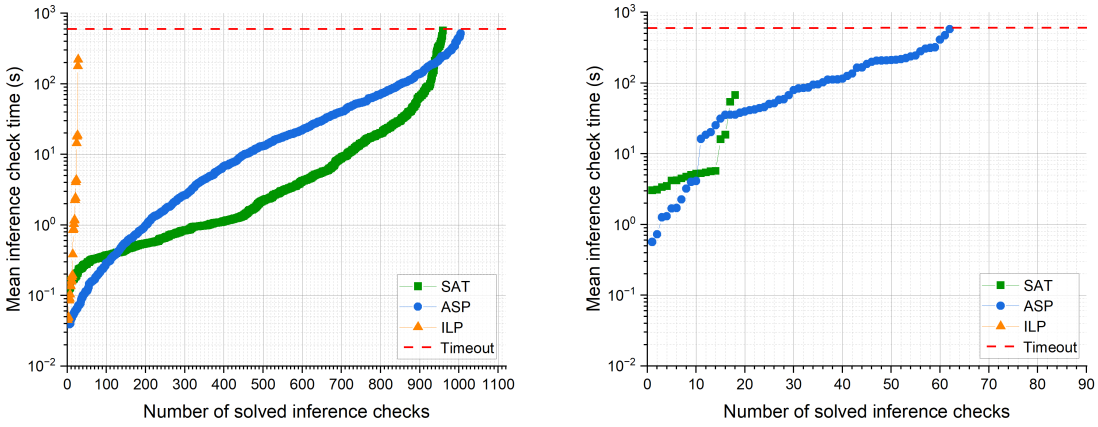


Figure 4.9: Inference check runtime for Satoh's revision with a timeout of 600 s. Data set R1.



(a) Data set C1

(b) Data set C2

Figure 4.10: Inference check runtime for Dalal's contraction with a timeout of 600 s

again performed poorest by solving only a few inference checks of data set C1 and none of data set C2. The SAT and ASP encodings on the other hand performed fairly well for data set C1, with the ASP encodings' termination rate being a bit higher than that of the SAT encodings. For data set C2 the ASP encodings again performed best with 62 out of 90 successful inference checks. Finally, Figure 4.11 visualizes the inference check results of Satoh's contraction on data set C1, which show similar behaviours of the different encoding types as on the Dalal contraction counterpart. The ILP encodings again performed poorest and the SAT and ASP encodings best, by solving almost all (SAT) or all (ASP) of the inference checks.

It can be noted that the ILP encodings accomplish shorter runtimes for some instances, than the SAT encodings. This phenomenon is identical to the one observed

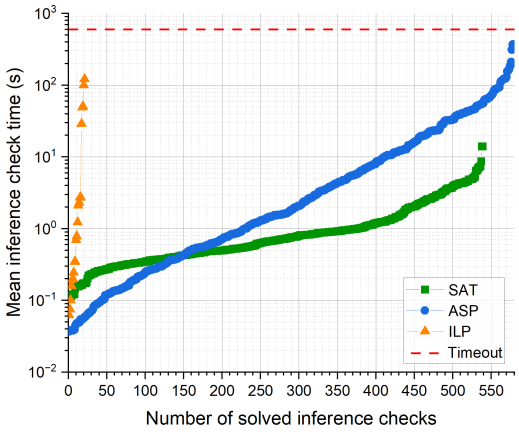


Figure 4.11: Inference check runtime for Satoh's contraction with a timeout of 600 s. Data set C1.

in the compilation runtime analysis of the previous section. Furthermore, the ASP encodings were clearly the best-performer with respect to the number of solved inference checks and it remains to be determined whether they were also the best-performers with regards to runtime. Since cactus plots are not well-suited for obtaining information on the number of test cases in which a given algorithm instance's runtime is shortest, Table 4.2 shows for every encoding approach, the number of inference check instances, for which it accomplished the shortest runtime. Only inference check instances that were solved by at least one approach (SAT, ASP, ILP) within the given time of 600 seconds are considered. In case an instance was solved by only one approach, this approach constitutes the best performer. In case two approaches' runtimes were identical, both are considered the best performer.

Data set	Operator	SAT	ASP	ILP
C1	Dalal	626	490	0
C2	Dalal	8	54	0
R1	Dalal	227	723	32
R2	Dalal	17	356	0
C1	Satoh	314	266	0
R1	Satoh	114	542	0

Table 4.2: Number of inference checks with shortest runtime per encoding type

Since with regards to all operators and data sets except for Dalal's revision on R2 and Dalal's contraction on C2, the SAT and ASP approaches solved a similar number of inference check instances, rows 1, 3, 5 and 6 are of special interest. From the contained data it can be concluded that with regards to belief contraction the SAT encodings had the shortest runtime for more inference check instances than the ASP encodings. With

regards to belief revision on the other hand, the ASP encodings had the shortest runtime for far more inference check instances than the SAT encodings. Despite the SAT encodings' shorter runtimes in many belief contraction inference checks, the ASP encodings can be considered the overall best-performer with respect to inference checks as they accomplished to solve more inference checks than their SAT counterparts, especially with regards to the more difficult data sets R2 and C2. The next section continues our evaluation results analysis by having a look at the model check results.

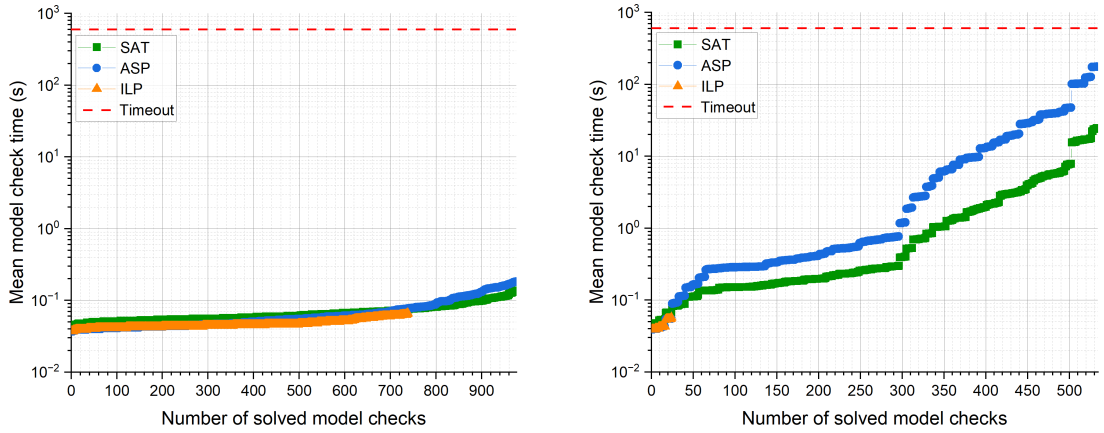
4.3.3 Model checks

In the following we analyze the experimental results of our model check evaluation. As described in the previous section, the SAT, ASP, and ILP encodings used in the model check experiments are identical to those of the inference checks. For each encoding two model checks were run, using the interpretations generated during the initial data set creation, as described in Section 4.1.

Analogously to the inference checks, almost all of the unsuccessful model check tests encountered a timeout rather than error. We begin our model check analysis by having a look at the results for Dalal's revision, depicted in Figure 4.12. It is apparent immediately that all three encoding types performed fairly well on data set R1 with the SAT and ASP approaches successfully terminating in less than a second for all 976 model checks. The ILP approach succeeded for 738 model check instances with an almost identical runtime behaviour as its SAT and ASP counterparts. With regards to data set R2 the ILP approach performed significantly worse whereas the SAT and ASP approaches again successfully solved all model check instances without timing out nor encountering an error. Looking at Figure 4.13, which contains the results for Satoh's revision operator, we observe a similar picture as with Dalal's revision on data set R1: the SAT and ASP approaches solved all model check instances, with almost all checks taking far less than a second, whereas the ILP model checks successfully terminated for 478 out of 656 instances.

What stands out in the above belief revision results is the fact that all of the ILP approach's model check runtimes are below one second, even though this approach encountered many timeouts, signifying that the model check instances were either extremely easy or extremely difficult to solve using the ILP encodings, with the intermediate range being empty.

Next, we analyze the model check results for Dalal's and Satoh's contraction operators. From the cactus plots of Figure 4.14 we obtain that for Dalal's contraction, the ILP encodings performed once again poorest, with only a few successfully solved model checks of data set C1 and no solved model checks of data set C2. The SAT and ASP encodings on the other hand exhibited a fairly well performance with regards to both data sets. Both approaches terminated successfully for all 1 120 model checks of data set C1. Out of the 90 model check instances of data set C2, the SAT approach solved 80 and the ASP approach 61 instances without timing out. Finally, Figure 4.15 depicts the model check results for Satoh's contraction on data set C1, which are very similar



(a) Data set R1 (b) Data set R2

Figure 4.12: Model check runtime for Dalal's revision with a timeout of 600 s

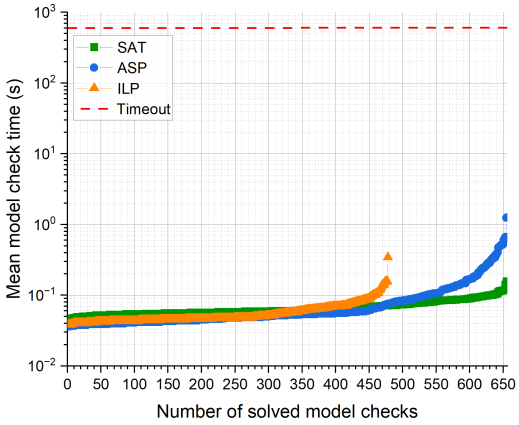
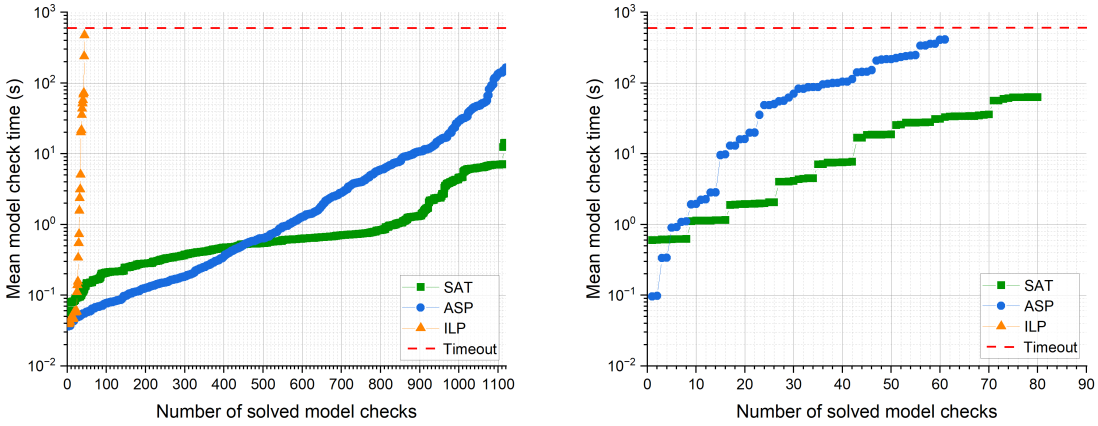


Figure 4.13: Model check runtime for Satoh's revision with a timeout of 600 s. Data set R1.

to those of Dalal's contraction, with the ILP approach performing poorest and the SAT and ASP approaches both successfully solving all of the 580 model check instances. It can be noted that the model check runtimes for belief contraction were significantly higher than the ones for belief revision analyzed above.

Overall, it can be observed that the model checks' runtimes were generally shorter than that of the inference checks with a smaller number of timeouts and errors. Analogously to the observations made in the compilation and inference check analyses above, the ILP encodings once again reached smaller runtimes for some model check instances, than the SAT encodings. With regards to the number of solved model check instances, the SAT and ASP encodings showed an almost identical performance, the only exception being Dalal's contraction on data set C2, where the SAT encodings performed



(a) Data set C1 (b) Data set C2

Figure 4.14: Model check runtime for Dalal's contraction with a timeout of 600 s

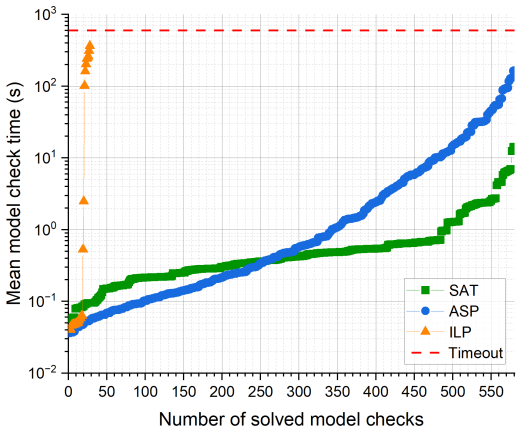


Figure 4.15: Model check runtime for Satoh's contraction with a timeout of 600 s. Data set C1.

slightly better than the ASP ones. In order to gain some better insights into which approach performed better with regards to runtime, we have a look at Table 4.3, which shows for every encoding approach the number of model check instances, for which it accomplished the shortest runtime. Only model check instances that were solved by at least one approach (SAT, ASP, ILP) without timing out are considered. In case an instance was solved by only one approach, this approach is determined the best performer. In case two approaches' runtimes were identical, both methods are considered the best performer. Looking at the table's data, in four rows the SAT approach clearly outperforms the ASP approach with regards to the number of model check instances for which it had the shortest runtime. The results of the remaining two rows (Dalal's revision on data set R1 and Satoh's revision on data set R1), in which the ASP and ILP

approaches outperform the SAT one, are of less relevance since for these two cases the runtimes of all three encoding approaches were fairly similar. Conclusively, whereas the ASP encodings performed best with regards to inference checks, the SAT encodings are the overall best performers with regards to model checks.

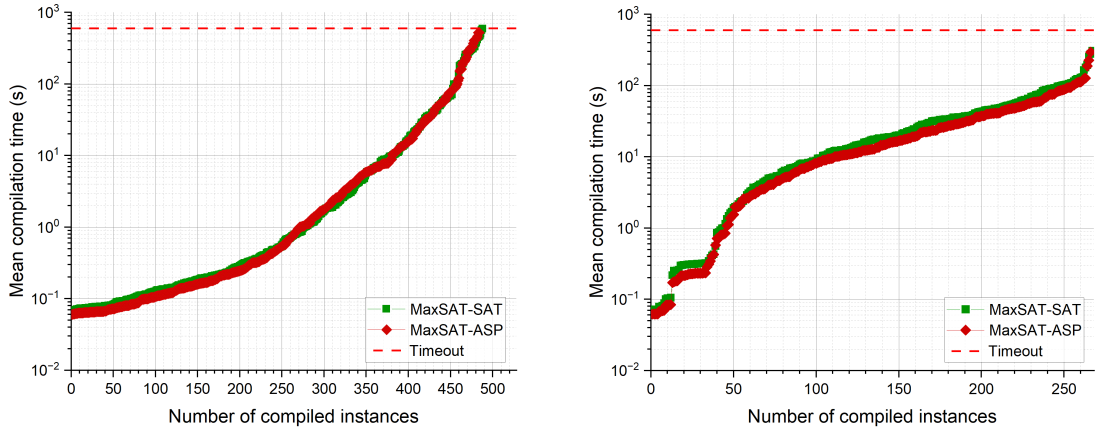
Data set	Operator	SAT	ASP	ILP
C1	Dalal	649	464	8
C2	Dalal	66	19	0
R1	Dalal	177	264	637
R2	Dalal	510	14	20
C1	Satoh	297	282	2
R1	Satoh	186	375	114

Table 4.3: Number of model checks with shortest runtime per encoding type

4.3.4 Further analyses

We recall from Section 4.3.1 that out of the evaluated compilation algorithm instances, MaxSAT-SAT performed best. Further, we concluded in the same section that it is worthwile to run another set of compilation experiments on the algorithm instance MaxSAT-ASP, due to MaxSAT-SAT having encountered several encoding generation errors and timeouts whereas ASP-ASP had not. The need for such additional compilation experiments on the MaxSAT-ASP instance is further underscored by the conclusion of Section 4.3.2, i.e. the significantly better performance of the ASP encodings as opposed to the SAT encodings with regards to inference checks. Conclusively, we carried out another set of compilation tests, using the same belief change instances (data sets R1, R2, C1 and C2) as in the first set described in Section 4.3.1, this time focusing on the algorithm instance MaxSAT-ASP only. Figures 4.16, 4.17, 4.18, and 4.19 contain cactus plots plotting the newly obtained runtimes of the algorithm instance MaxSAT-ASP along with those of the instance MaxSAT-SAT measured in the first set of compilation tests. Again, all plotted runtimes constitute the mean value of three non-sequential runs. With regards to Dalal's revision (Figure 4.16), the results for both algorithm instances are highly similar, with the MaxSAT-ASP instance successfully compiling all but four of the R1 belief change instances and all of the R2 belief change instances that were compiled by MaxSAT-SAT. Further, the runtimes of the MaxSAT-ASP algorithm instance seem to be slightly shorter than those of MaxSAT-SAT. The same observation can be made in the cactus plot of Figure 4.17 on Satoh's revision. Here, both instances successfully compiled the exact same amount of belief revision instances. With regards to Dalal's contraction (Figure 4.18) larger performance differences are observed. For data set C1 the instance MaxSAT-ASP successfully compiled all but eleven of the belief change instances that were successfully compiled by MaxSAT-SAT. Again, the runtimes of MaxSAT-ASP seem to be shorter than those of MaxSAT-SAT. The most relevant observation can be made for data set C2: whereas MaxSAT-SAT compiled only 45 be-

lief change instances, the algorithm instance MaxSAT-ASP accomplished to compile 95 instances. Lastly, with regards to Satoh's contraction, the results of both algorithm instances are almost identical.



(a) Data set R1 (b) Data set R2

Figure 4.16: Runtime comparison between MaxSAT-SAT and MaxSAT-ASP approaches for Dalal's revision with a timeout of 600 s

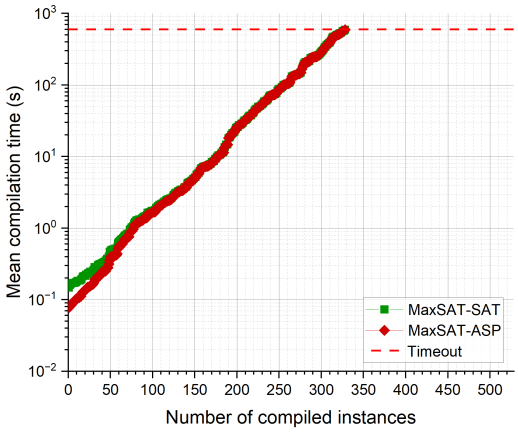
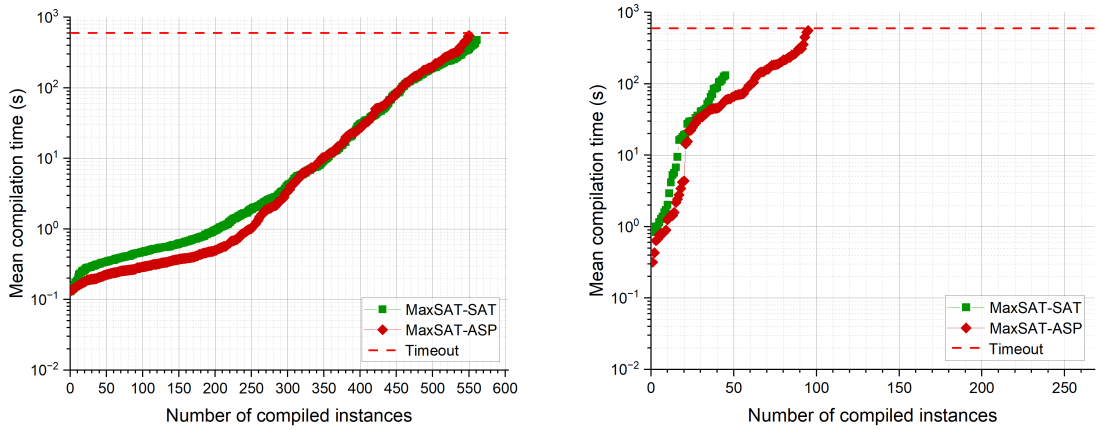


Figure 4.17: Runtime comparison between MaxSAT-SAT and MaxSAT-ASP approaches for Satoh's revision with a timeout of 600 s. Data set R1.

Table 4.4 shows for every algorithm instance (MaxSAT-SAT, MaxSAT-ASP, ASP-ASP, ILP-ILP) the number of belief change instances, for which the algorithm instance performed best with regards to runtime. Again, in case a belief change instance was compiled by only one algorithm instance, this algorithm instance is determined the best performer. In case two runtimes were identical, both algorithm instances are considered the best performer. The provided data shows that for all operators and data sets



(a) Data set C1 (b) Data set C2

Figure 4.18: Runtime comparison between MaxSAT-SAT and MaxSAT-ASP approaches for Dalal's contraction with a timeout of 600 s

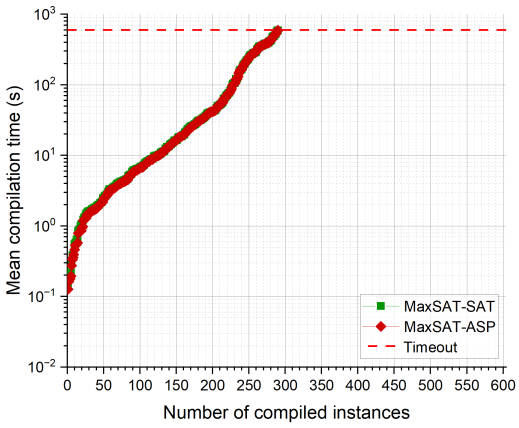


Figure 4.19: Runtime comparison between MaxSAT-SAT and MaxSAT-ASP approaches for Satoh's contraction with a timeout of 600 s. Data set C1.

Data set	Operator	MaxSAT-SAT	ASP-ASP	ILP-ILP	MaxSAT-ASP
C1	Dalal	168	27	0	366
C2	Dalal	6	0	0	89
R1	Dalal	100	184	0	204
R2	Dalal	57	8	0	202
C1	Satoh	5	33	0	263
R1	Satoh	0	199	0	159

Table 4.4: Number of compilations with shortest runtime per algorithm instance

except for Satoh's revision on data set R1, the algorithm instance MaxSAT-ASP clearly outperformed the other algorithm instances in terms of the number of belief change instances, for which it had the shortest runtime.

As a conclusion, our assumption that MaxSAT-ASP might potentially be able to solve some belief change instances, for which MaxSAT-SAT encountered compilation timeouts and errors, was confirmed by its fairly good performance for Dalal's contraction on data set C2, where the MaxSAT-SAT algorithm instance had run into several out-of-memory errors. Moreover, not only did MaxSAT-ASP manage to solve these instances, but also it accomplished overall shorter runtimes than MaxSAT-SAT. This violates our initial assumption of Section 4.3.1 that algorithm instances that use the same technology for both the preprocessing and the encoding generation perform better than mixed-technology instances. One possible explanation for MaxSAT-ASP outperforming MaxSAT-SAT with regards to runtime might be the relatively huge size of the SAT encodings compared to the ASP encodings. In fact, the huge size of the SAT encodings was most probably the reason for the out-of-memory errors in the encoding generation step.

The next section concludes our evaluation with the comparison of our compilation-based approach to the naive baseline implementation.

4.3.5 Comparison with naive implementation

As mentioned above, the naive approach was neglected in the above sections on the inference and model check results, with the reason being its poor performance with regards to the model list generation (see Section 4.3.1). As only 14 model lists were successfully generated within the timeout of 600 seconds, our naive inference and model checks were limited to these few solved instances.

Out of the resulting naive inference and model checks all terminated successfully before the timeout of 600 seconds and for some of the few evaluated instances the naive approach was even able to outperform some of the compilation-based approaches. Given the naive approach's extremely poor performance in the generation of the model list and it occasionally outperforming the compilation-based approaches in the inference and model checks, the question arises, whether the naive approach could potentially outperform the other approaches with regards to the total combined runtime of compilation and inference/model checks if it was given more time for the model list generation. In order to answer this question, we have a look at the average total combined runtimes of our compilation-based approaches.

Figures 4.20 and 4.21 show the average total combined runtime for each data set and operator, using the best performing compilation algorithm instances, namely MaxSAT-ASP for inference checks and MaxSAT-SAT for model checks. From these figures it is evident that our best-performing compilation-based approaches took on average well less than three minutes for the compilation and the inference/model check combined. With regards to model checks, the average total combined runtime was even far below 2 minutes for all data sets and operators. From these insights, combined with the in-

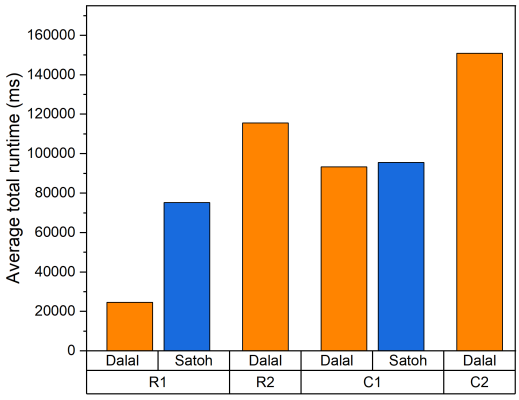


Figure 4.20: Average total runtime of inference checks (compilation time + inference check time) per data set and operator using the compilation algorithm instance MaxSAT-ASP

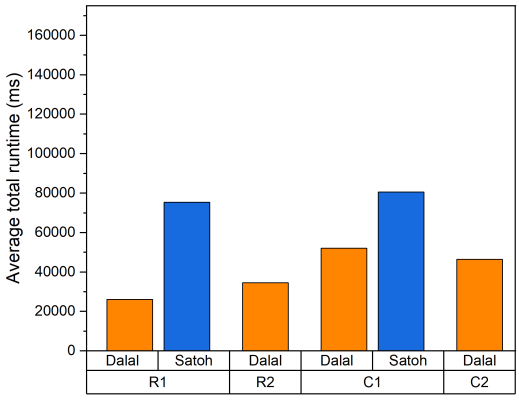


Figure 4.21: Average total runtime for model checks (compilation time + model check time) per data set and operator using the compilation algorithm instance MaxSAT-SAT

sights of Section 4.3.1 with regards to the poor performance of the naive approach, we can conclude that, as expected, our compilation-based approaches clearly outperform the naive implementation. From the available results we estimate the degree of outperformance to be at least of the magnitude of several minutes. However, for obtaining a more reliable and accurate number, further investigations into the performance of the naive implementation are required.

5 Conclusion

In the course of this work we introduced and evaluated a compilation-founded application for distance-based belief change that is able to compile belief change instances into encodings and carry out inference and model checks using the same. The application architecture, implemented algorithms as well as the encoding schemes used therein were described and a thorough experimental analysis was conducted. The results confirmed that our compilation-based solution constitutes a viable and in fact promising approach to belief change by not only clearly outperforming the implemented naive implementation but also successfully solving a large amount of non-trivial inference and model check instances within reasonable time. The data collected in the compilation experiments confirm the observations made by Konieczny et al. [KLM17] that partial MaxSAT encodings can be efficiently used for determining the minimum Dalal distances of non-trivial belief revision instances and that SAT encodings of the revision results can be generated in reasonable time. Furthermore, our results demonstrate that the same is possible for Dalal's contraction, and Satoh's revision and contraction operators, as well as by employing ASP and ILP technologies instead of SAT. Most importantly, however, the results obtained from our inference and model check experiments show that the generated belief change encodings - especially the SAT and ASP encodings - perform fairly well with regards to inference and model checks even in non-trivial cases. Overall, our results demonstrate that the SAT encodings are better suited for model checks, whereas the ASP encodings perform better in inference checks. With regards to the preprocessing, i.e. the determination of the minimum Dalal distance and Satoh's minimal sets, the partial MaxSAT optimization encoding has proven to be the best-performing encoding scheme, hence making our algorithm instances MaxSAT-SAT and MaxSAT-ASP the first choice for model and inference checks, respectively.

Even though the evaluation results of the present work accomplished to prove the potential of compilation-based approaches to belief change, there are several limitations of the conducted evaluation that should be noted and addressed in future research activities. One such limitation emanates from the absence of standard benchmark data for belief change operations. The non-existence of such data generally complicates attempts to conduct thorough comparisons of belief change implementations and the additional unavailability of the evaluation data used by Konieczny et al. [KLM17], hindered us from directly comparing their compilation results to ours.

Another limitation lies in the fact that our data sets might not have exhibited enough variation with regards to the value of the minimum Dalal distance and the inference and model check results. Table 5.1 shows the minimum Dalal distances of those belief change instances of our data sets, that were compiled by at least one algorithm instance, whereas Table 5.2 shows the minimum and maximum number of determined minimal sets for the compiled belief change instances. While the number of minimal sets seems to lie within a reasonably large range of values, the minimum Dalal distance is of value one for the majority of belief change instances. Table 5.3 further shows the results of the inference and model checks for those instances of the data sets, that were both compiled

Data set	$d_{min} = 1$	$d_{min} = 2$	$d_{min} = 3$	$d_{min} = 4$	$d_{min} = 5$	$d_{min} = 6$
C1	561	-	-	-	-	-
C2	95	-	-	-	-	-
R1	457	27	1	3	-	1
R2	247	15	-	4	1	-

Table 5.1: Minimum dalal distances (d_{min}) of compiled belief change instances

Data set	Min # Minimal sets	Max # Minimal sets
C1	2	926
R1	2	2449

Table 5.2: Minimum and maximum number of determined Satoh minimal sets for compiled belief change instances

and passed the inference/model checks without timing out. Again, it can be observed that the majority of both inference and model checks had the result $FALSE$. As the mentioned tables contain information on the successfully solved data set instances only, the question arises of whether our data sets were indeed unbalanced with regards to the minimum Dalal distances and inference and model check results or whether the missing cases were precisely those that failed in the evaluation tests. Since it might very well be the case that aspects such as the minimum Dalal distance value and the inference or model check result directly or indirectly influence the performance of the suggested algorithms and encoding schemes, further research with a highly balanced data set should be conducted.

Check	Operator	Data set	$TRUE$	$FALSE$
Inference	**Dalal**	**C1**	11	1105
Inference	**Dalal**	**C2**	29	33
Inference	**Dalal**	**R1**	28	948
Inference	**Dalal**	**R2**	223	150
Inference	**Satoh**	**C1**	6	574
Inference	**Satoh**	**R1**	12	644
Model	**Dalal**	**C1**	8	1112
Model	**Dalal**	**C2**	0	85
Model	**Dalal**	**R1**	0	976
Model	**Dalal**	**R2**	0	534
Model	**Satoh**	**C1**	8	572
Model	**Satoh**	**R1**	0	656

Table 5.3: Inference and model check results for non-timed-out belief change instances

An interesting observation that we made during the analysis of our experimental

results, is the apparent absence of a direct relation between runtime, and belief change instances' signature size and clause number. Instead, there seems to exist some hidden factor, potentially the number of models, that directly influences runtime. Conclusively, future research needs to ensure that data sets are not only balanced with regards to signature size and clause number, but also in terms of the number of models of the belief base and change formula. Besides, new research activities could focus on attempting to gain an understanding of how different characteristics of belief change instances affect runtime performance of the corresponding encodings.

Additionally, new research could focus on improving the encoding schemes and algorithms presented in this work as well as propose new ones to support additional belief change operators. Also, a more detailed analysis of eveluation results could be conducted, to gain deeper insights into the reasons for one encoding outperforming the other.

Finally, given our observation that solver calls comprise a substantial portion of the overall compilation runtime, it is imperative for future research to explore the adoption of alternate solvers.

References

[AGM85] Carlos Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50:510–530, 1985.

[ANORC13] Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. A parametric approach for smaller and better encodings of cardinality constraints. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, pages 80–96. Springer Berlin Heidelberg, 2013.

[Ara22] Theofanis Aravanis. An ASP-based solver for parametrized-difference revision. *Journal of Logic and Computation*, 32(3):630–666, 2022.

[BHvMW09] Armin Biere, Marijn Heule, Hans van Maaren, and Tory Walsh. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[CKM17] Thomas Caridroit, Sébastien Konieczny, and Pierre Marquis. Contraction in propositional logic. *International Journal of Approximate Reasoning*, 80:428–442, 2017.

[Dal88] Mukesh Dalal. Investigations into a theory of knowledge base revision: Preliminary report. In *AAAI-88 Proceedings*, pages 475–479, 1988.

[DLRT05] James Delgrande, Jérôme Lang, Hans Rott, and Jean-Marc Tallon. 05321 Executive Summary – Belief Change in Rational Agents: Perspectives from Artificial Intelligence, Philosophy, and Economics. In James Delgrande, Jerome Lang, Hans Rott, and Jean-Marc Tallon, editors, *Belief Change in Rational Agents: Perspectives from Artificial Intelligence, Philosophy, and Economics*, volume 5321 of *Dagstuhl Seminar Proceedings (DagSemProc)*, pages 1–5. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2005.

[DLST07] James P. Delgrande, Daphne H. Liu, Torsten Schaub, and Sven Thiele. Coba 2.0: A consistency-based belief change system. In Khaled Mellouli, editor, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pages 78–90. Springer-Verlag Berlin Heidelberg, 2007.

[DP97] Adnan Darwiche and Judea Pearl. On the logic of iterated belief revision. *Artificial Intelligence*, 89:1–29, 1997.

[EG92] Thomas Eiter and Georg Gottlob. On the complexity of propositional knowledge base revision, updates, and counterfactuals. *Artificial Intelligence*, 57(2):227–270, 1992.

[FH18] Eduardo Fermé and Sven Ove Hansson. *Belief change: introduction and overview*. Springer, 2018.

[For89] Kenneth D. Forbus. Introducing actions into qualitative simulation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1273–1278. Morgan Kaufmann, 1989.

[FUV83] Ronald Fagin, Jeffrey D. Ullman, and Moshe Y. Vardi. On the semantics of updates in databases. In *PODS '83: Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 352–365, 1983.

[Gin86] Matthew L. Ginsberg. Counterfactuals. *Artificial Intelligence*, 30(1):35–79, 1986.

[GLM14] Éric Grégoire, Jean-Marie Lagniez, and Bertrand Mazure. Multiple contraction through partial-Max-SAT. In *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, pages 321–327, 2014.

[GM88] Peter Gärdenfors and David Makinson. Revisions of knowledge systems using epistemic entrenchment. In *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge, Pacific Grove, CA*, pages 83–95, 1988.

[Gä88] Peter Gärdenfors. *Knowledge in Flux. Modelling the Dynamics of Epistemic States*. MIT Press, 1988.

[HA19] Aaron Hunter and John Agapeyev. An efficient solver for parametrized difference revision. In Jixue Liu and James Bailey, editors, *AI 2019: Advances in Artificial Intelligence, 32nd Australasian Joint Conference Adelaide, SA, Australia, December 2-5, 2019, Proceedings*, Lecture Notes in Computer Science, pages 143–152. Springer International Publishing, 2019.

[Han91] Sven Ove Hansson. Belief contraction without recovery. *Studia Logica*, 50:251–260, 1991.

[Han99] Sven Ove Hansson. *A Textbook of Belief Dynamics: Theory Change and Database Updating*. Springer Science & Business Media, 1999.

[Har76] William L. Harper. Rational conceptual change. In *PSA: Proceedings of the Biennial Meeting of the Philosophy of Science Association*, volume 1976, pages 462–494. University of Chicago Press, 1976.

[HV91a] Joseph Y. Halpern and Moshe Vardi. Belief revision and default reasoning: Syntax-based approaches. In J.A. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning (KR'91)*, pages 417–428. Morgan Kaufmann, 1991.

[HV91b] Joseph Y. Halpern and Moshe Vardi. Model checking vs. theorem proving: A manifesto. In J.A. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on the Principles of Knowledge*

Representation and Reasoning (KR'91), pages 325–334. Morgan Kaufmann, 1991.

[IS19] Markus Iser and Carsten Sinz. A problem meta-data library for research in sat. In Daniel Le Berre and Matti Järvisalo, editors, *Proceedings of Pragmatics of SAT 2015 and 2018*, volume 59 of *EPiC Series in Computing*, pages 144–152. EasyChair, 2019.

[KLM17] Sébastien Konieczny, Jean-Marie Lagniez, and Pierre Marquis. Boosting distance-based revision using SAT encodings. In *Logic, Rationality, and Interaction, 6th International Workshop, LORI 2017, Sapporo, Japan, September 11-14, 2017, Proceedings*, Lecture Notes in Computer Science, pages 480–496. Springer Berlin, Heidelberg, 2017.

[KM89] Hirofumi Katsuno and Alberto O. Mendelzon. A unified view of propositional knowledge base updates. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1413–1419. Morgan Kaufmann, 1989.

[KM91] Hirofumi Katsuno and Alberto O. Mendelzon. Propositional knowledge base revision and minimal change. *Artificial Intelligence*, 52(3):263–294, 1991.

[KM92] Hirofumi Katsuno and Alberto O. Mendelzon. On the difference between updating a knowledge base and revising it. In P. Gärdenfors, editor, *Belief Revision*, pages 183–203. Cambridge University Press, 1992.

[Lev77] Isaac Levi. Subjunctives, dispositions and chances. *Synthese*, 34:423–455, 1977.

[Lib98] Paolo Liberatore. *Compilation of Intractable Problems and Its Application to Artificial Intelligence*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1998.

[Lif19] Vladimir Lifschitz. *Answer Set Programming*. Springer International Publishing, 2019.

[LS01] Paolo Liberatore and Marco Schaerf. Belief revision and update: Complexity of model checking. *Journal of Computer and System Sciences*, 62:43–72, 2001.

[LZD04] Ruiming Li, Dian Zhou, and Donglei Du. Satisfiability and integer programming as complementary tools. In *Proceedings of the 2004 Conference on Asia South Pacific Design Automation: Electronic Design and Solution Fair 2004*, pages 879–882, 2004.

[MP75] David E. Muller and Franco P. Preparata. Bounds to complexities of networks for sorting and for switching. *Journal of the Association for Computing Machinery*, 22(2):195–201, 1975.

[Neb89] Bernhard Nebel. A knowledge level analysis of belief revision. In R. Brachman, H. J. Levesque, and R. Reiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 1st International Conference (KR'89)*, pages 301–311. Morgan Kaufmann, 1989.

[Sat88] Ken Satoh. Nonmonotonic reasoning by minimal belief revision. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 455–462, 1988.

[Sch86] Alexander Schrijver. Theory of linear and integer programming. In *Wiley-Interscience series in discrete mathematics and optimization*, 1986.

[SDP09] Mariette Sérayet, Pierre Drap, and Odile Papini. Encoding the revision of partially preordered information in Answer Set Programming. In Claudio Sossai and Gaetano Chemello, editors, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pages 421–433. Springer-Verlag Berlin Heidelberg, 2009.

[Sin05] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005*, Lecture Notes in Computer Science, page 827–831. Springer Berlin Heidelberg, 2005.

[Tse68] Grigori S. Tseitin. On the complexity of derivation in propositional calculus. *Structures in Constructive Mathematics and Mathematical Logic*, pages 115–125, 1968.

[Win88] Marianne Winslett. Reasoning about action using a possible models approach. In *AAAI-88 Proceedings*, 1988.

A Proofs

Lemma 1. *For each model M of $\kappa^y \wedge \mu$ the following holds: $proj(M, \{y_1, ..., y_n\}) = M_\kappa$, with $M_\kappa \in Mod(\kappa)$, and $proj(M, \{x_1, ..., x_n\}) = M_\mu$, with $M_\mu \in Mod(\mu)$. Further, each possible combination of members of $Mod(\kappa)$ and $Mod(\mu)$ is covered by a model of $\kappa^y \wedge \mu$.*

Proof. As κ^y is identical to κ with the only difference being the used variables, $Mod(\kappa^y) = Mod(\kappa)$. Furthermore, since $Var(\kappa^y) \cap Var(\mu) = \emptyset$, the two conjuncts κ^y and μ do not affect each other and thus for each model $M_\kappa \in Mod(\kappa)$, there are $|Mod(\mu)|$ models $M \in Mod(\kappa^y \wedge \mu)$ that combine M_κ with a model of $Mod(\mu)$. □

Lemma 2. *Formula 4 ensures that $d_j = 1$, whenever $x_j \neq y_j$.*

Proof. Lemma 2 can be proven by the corresponding truth table:

x_j	y_j	d_j	$(d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j)$
T	T	T	T
T	T	F	T
T	F	T	T
T	F	F	F
F	T	T	T
F	T	F	F
F	F	T	T
F	F	F	T

□

Theorem 1. *From the optimal solution S to the partial MaxSAT encoding $S^O_{+_D}(\kappa, \mu)$ the minimum Dalal distance $d_{min}(\kappa, \mu)$ can be obtained by $d_{min}(\kappa, \mu) = \sum_{j=1}^{n} value(S, d_j)$.*

Proof. By looking at the truth table in the proof of Lemma 2, it becomes evident that Lemma 1 also applies to formula $H = \kappa^y \wedge \mu \wedge \bigwedge_{1 \leq j \leq n} \Big((d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j) \Big)$, since Formula 4 can evaluate to true, irrespective of the truth assignments of x_j and y_j, as long as $d_j = 1$ if $x_j \neq y_j$. Accordingly, each model M of H represents the comparison of a model M_κ to a model M_μ and due to Lemma 2 each variable d_j, for which $value(M, d_j) = 0$ holds, indicates that $x_j = y_j$. Adding a soft clause of the form $\neg d_j$ for every $d_j \in \{d_1, ..., d_n\}$ and assigning the weight 1 to each soft clause, forces a partial MaxSAT solver to find a solution to the encoding, that corresponds to a model of H with the highest possible number of d_j, for which $value(M, d_j) = 0$. Note that the fact that d_j can be either 0 or 1 in the case of $x_j = y_j$ does not interfere with the above establishment, since a partial MaxSAT solver will - in its attempt to maximize the satisfied soft clauses - assign the value 0 to each d_j, wherever possible. Maximizing the number of variables d_j with value 0 corresponds to minimizing the number of variables d_j with value 1 and hence the optimal solution to $S^O_{+_D}(\kappa, \mu)$ indicates the minimum Dalal distance $d_{min}(\kappa, \mu)$ through the total number of d_j variables with an assigned value of 1. □

Theorem 2. *For the SAT encoding $S_{+_D}(\kappa, \mu)$ following relation holds: $proj(Mod(S_{+_D}(\kappa, \mu)), \{x_1, ..., x_n\}) = Mod(\kappa \dot{+}_D \mu)$*

Proof. Dalal's revision operator $\dot{+}_D$ is defined as selecting exactly those models of the revision formula μ, that have the minimum Dalal distance $d_{min}(\kappa, \mu)$ to at least one of the belief base's (κ) models. From the proof of Theorem 1 we already know that each model M of $\kappa^y \wedge \mu \wedge \bigwedge_{1 \leq j \leq n} \Big((d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j) \Big)$ represents the comparison of a model M_κ to a model M_μ and each variable d_j, for which $value(M, d_j) = 0$, indicates that $x_j = y_j$. By adding the exactly-*k* constraint $E(\{d_1, ..., d_n\}, d_{min}(\kappa, \mu))$ it is ensured that in all models of $S_{\dot{+}_D}(\kappa, \mu)$ the number of positive variables $\{d_1, ..., d_n\}$ equals $d_{min}(\kappa, \mu)$. Again, it should be noted that the fact that each variable d_j can be either 0 or 1 in case of $x_j = y_j$ does not affect this behaviour as we know that $d_{min}(\kappa, \mu)$ is the smallest possible sum of positive d_j variables and setting the sum equal to $d_{min}(\kappa, \mu)$ ensures automatically that d_j turns 0 whenever possible. The

exact same applies to the fact mentioned in Section 3.3.1 that the counter bit variables in $E(\{d_1, ..., d_n\}, d_{min}(\kappa, \mu))$ are not forced to take the value 0, but only forced to take the value 1, whenever their corresponding binary number bit is 1. Now that we know that in each model of $S_{\dot{+}_D}(\kappa, \mu)$ the number of positive variables in $\{d_1, ..., d_n\}$ equals $d_{min}(\kappa, \mu)$, we can state that the interpretations $proj(Mod(S_{\dot{+}_D}(\kappa, \mu)), \{y_1, ..., y_n\})$ are equal to those models of κ, for which a model of μ exists such that the Dalal distance between the models is $d_{min}(\kappa, \mu)$. Moreover, we can state that the interpretations $proj(Mod(S_{\dot{+}_D}(\kappa, \mu)), \{x_1, ..., x_n\})$ are equal to those models of μ, for which at least one model of κ exists, such that the Dalal distance between the models is $d_{min}(\kappa, \mu)$. The last description corresponds to the definition of Dalal's revision operator, hence $proj(Mod(S_{\dot{+}_D}(\kappa, \mu)), \{x_1, ..., x_n\}) = Mod(\kappa \dot{+}_D \mu)$. □

Theorem 3. *From the optimal solution S to the partial MaxSAT encoding $S^O_{\dot{-}_D}(\kappa, \phi)$ the minimum Dalal distance $d_{min}(\kappa, \neg\phi)$ can be obtained by $d_{min}(\kappa, \neg\phi) = \sum_{j=1}^{n} value(S, d_j)$.*

Proof. The encoding $S^O_{\dot{-}_D}(\kappa, \phi)$ is almost identical to encoding $S^O_{\dot{+}_D}(\kappa, \mu)$ of the corresponding revision Theorem 1 since $t(\neg\phi)$ and μ are both CNF formulae, with the only difference being that $t(\neg\phi)$ contains some additional auxiliary variables $V^t_{\neg\phi}$, that were introduced by the Tseitin transformation. However, as $proj(Mod(t(\neg\phi)), Var(\neg\phi)) = Mod(\neg\phi)$ due to CNF formulae resulting from Tseitin transformations being equisatisfiable to the initial formula, and because the variables $V^t_{\neg\phi}$ only occur within $t(\neg\phi)$, their presence can be neglected. The remaining proof is thus identical to the proof of Theorem 1. □

Lemma 3. *For the SAT encoding*

$$D = \kappa^y \ \wedge \ t(\neg\phi) \ \wedge \bigwedge_{1 \leq j \leq n} \Big((d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j) \Big) \wedge \ E(\{d_1, ..., d_n\}, d_{min}(\kappa, \neg\phi))$$

the following holds:

$$proj(Mod(D), \{x_1, ..., x_n\}) = Mod(\kappa \dot{+} \neg\phi)$$

Proof. D is almost identical to encoding $S_{\dot{+}_D}(\kappa, \mu)$ of the corresponding revision Theorem 2 since $t(\neg\phi)$ and μ are both CNF formulae, with the only difference being that $t(\neg\phi)$ contains the auxiliary variables $V^t_{\neg\phi}$ introduced by the Tseitin transformation. As $proj(Mod(t(\neg\phi)), Var(\neg\phi)) = Mod(\neg\phi)$ due to CNF formulae resulting from Tseitin transformations being equisatisfiable to the initial formula, and because the variables $V^t_{\neg\phi}$ only occur within $t(\neg\phi)$, their presence can be neglected. The remaining proof is thus identical to the proof of Theorem 2. □

Lemma 4. *For the SAT encoding $F(X, Z, W, n)$ the following holds:*

$$\begin{aligned} proj(Mod(F(X, Z, W, n)), X) \ = \ & proj(Mod(F(X, Z, W, n)), Z) \\ & \cup \ proj(Mod(F(X, Z, W, n)), W) \end{aligned}$$

Proof. To obtain $F(X, Z, W, n)$, we start with formula

$$F' = \Big(\bigwedge_{1 \leq i \leq n} \Big((x_i \vee \neg z_i) \wedge (\neg x_i \vee z_i) \Big) \Big) \ \vee \ \Big(\bigwedge_{1 \leq i \leq n} \Big((x_i \vee \neg w_i) \wedge (\neg x_i \vee w_i) \Big) \Big)$$

Having a look at the first disjunct, in particular at $(x_i \vee \neg z_i) \wedge (\neg x_i \vee z_i)$ and below truth table, it is evident, that this formula ensures that x_i and z_i have identical truth assignments.

x_i	z_i	$(x_i \vee \neg z_i) \wedge (\neg x_i \vee z_i)$
T	T	T
T	F	F
F	T	F
F	F	T

By creating such a formula for each $x \in X$ and connecting all resulting formulae by $\wedge$, it is ensured that for all $z_i \in Z$ the corresponding x_i has the same truth assignment as z_i. The same applies to the second disjunct and the variables W. Thus, overall, formula F' ensures that

- all $x_i \in X$ have the same truth assignment as their corresponding $z_i \in Z$
- OR all $x_i \in X$ have the same truth assignment as their corresponding $w_i \in W$.

We can thus state that $proj(Mod(F'), X) = proj(Mod(F'), Z) \cup proj(Mod(F'), W)$.
Moreover, after applying the boolean transformation rules to F' to convert it into CNF, we see that it is logically equivalent to $F(X, Z, W, n)$:

$$
\begin{aligned}
F' &= \Big(\bigwedge_{1\le i\le n} \Big((x_i \vee \neg z_i) \wedge (\neg x_i \vee z_i)\Big)\Big) \vee \Big(\bigwedge_{1\le i\le n} \Big((x_i \vee \neg w_i) \wedge (\neg x_i \vee w_i)\Big)\Big) \\
&= \bigwedge_{1\le j\le n} \Big(\Big(\Big(\bigwedge_{1\le i\le n} \Big((x_i \vee \neg z_i) \wedge (\neg x_i i \vee z_i)\Big)\Big) \vee (x_j \vee \neg w_j)\Big) \\
&\quad \wedge \Big(\Big(\bigwedge_{1\le i\le n} \Big((x_i \vee \neg z_i) \wedge (\neg x_i \vee z_i)\Big)\Big) \vee (\neg x_j \vee w_j)\Big)\Big) \\
&= \bigwedge_{1\le j\le n} \Big(\Big(\bigwedge_{1\le i\le n} \Big((x_i \vee \neg z_i \vee x_j \vee \neg w_j) \wedge (\neg x_i \vee z_i \vee x_j \vee \neg w_j)\Big)\Big) \\
&\quad \wedge \Big(\bigwedge_{1\le i\le n} \Big((x_i \vee \neg z_i \vee \neg x_j \vee w_j) \wedge (\neg x_i \vee z_i \vee \neg x_j \vee w_j)\Big)\Big)\Big) \\
&= \bigwedge_{1\le j\le n} \Big(\Big(\bigwedge_{\substack{1\le i\le n \\ i\ne j}} \Big((x_i \vee \neg z_i \vee x_j \vee \neg w_j) \wedge (\neg x_i \vee z_i \vee x_j \vee \neg w_j)\Big) \\
&\quad \wedge (x_j \vee \neg z_j \vee x_j \vee \neg w_j) \wedge (\neg x_j \vee z_j \vee x_j \vee \neg w_j)\Big) \\
&\quad \wedge \Big(\bigwedge_{\substack{1\le i\le n \\ i\ne j}} \Big((x_i \vee \neg z_i \vee \neg x_j \vee w_j) \wedge (\neg x_i \vee z_i \vee \neg x_j \vee w_j)\Big) \\
&\quad \wedge (x_j \vee \neg z_j \vee \neg x_j \vee w_j) \wedge (\neg x_j \vee z_j \vee \neg x_j \vee w_j)\Big)\Big) \\
&= \bigwedge_{1\le j\le n} \Big(\Big(\bigwedge_{\substack{1\le i\le n \\ i\ne j}} \Big((x_i \vee \neg z_i \vee x_j \vee \neg w_j) \wedge (\neg x_i \vee z_i \vee x_j \vee \neg w_j)\Big) \\
&\quad \wedge (x_j \vee \neg z_j \vee \neg w_j)\Big) \\
&\quad \wedge \Big(\bigwedge_{\substack{1\le i\le n \\ i\ne j}} \Big((x_i \vee \neg z_i \vee \neg x_j \vee w_j) \wedge (\neg x_i \vee z_i \vee \neg x_j \vee w_j)\Big) \\
&\quad \wedge (\neg x_j \vee z_j \vee w_j)\Big)\Big) \\
&= \bigwedge_{1\le j\le n} \Big(\bigwedge_{\substack{1\le i\le n \\ i\ne j}} \Big((x_i \vee \neg z_i \vee x_j \vee \neg w_j) \wedge (\neg x_i \vee z_i \vee x_j \vee \neg w_j) \\
&\quad \wedge (x_i \vee \neg z_i \vee \neg x_j \vee w_j) \wedge (\neg x_i \vee z_i \vee \neg x_j \vee w_j)\Big) \\
&\quad \wedge (x_j \vee \neg z_j \vee \neg w_j) \wedge (\neg x_j \vee z_j \vee w_j)\Big) \\
&= F(X, Z, W, n)
\end{aligned}
$$

□

Theorem 4. *For the SAT encoding $S_{\dot{-}_D}(\kappa, \phi)$ following relation holds: $proj(Mod(S_{\dot{-}_D}(\kappa, \phi)), \{x_1, ..., x_n\}) = Mod(\kappa \dot{-}_D \phi)$*

Proof. $S_{\dot{-}_D}(\kappa, \phi)$ consists of the above mentioned encodings E'' and $F(X, Z, W, n)$:

$$S_{\dot{-}_D}(\kappa, \phi) = E'' \wedge F(X, Z, W, n) = E' \wedge \kappa^w \wedge F(X, Z, W, n)$$

Due to Lemma 3 and because the only difference between E' and E is that in the former the variables $\{x_1, ..., x_n\}$ have been replaced by new variables $\{z_1, ..., z_n\}$, we obtain $proj(Mod(E'), \{z_1, ..., z_n\}) = Mod(\kappa \dot{+}_D \neg\phi)$. Further, because of $Var(E') \cap Var(\kappa^w) = \emptyset$ also $proj(Mod(E''), \{z_1, ..., z_n\}) = Mod(\kappa \dot{+}_D \neg\phi)$ and $proj(Mod(E''), \{w_1, ..., w_n\}) = Mod(\kappa)$. Moreover, due to Lemma 4 and $Var(E'') \cap X = \emptyset$ (i.e. $F(X, Z, W, n)$ does not place any restrictions on the variables Z and W in E''), we obtain

$$proj(Mod(S_{\dot{-}_D}(\kappa, \phi)), \{z_1, ..., z_n\}) = proj(Mod(E''), \{z_1, ..., z_n\}) = Mod(\kappa \dot{+}_D \neg\phi)$$

and

$$proj(Mod(S_{\dot{-}_D}(\kappa, \phi)), \{w_1, ..., w_n\}) = proj(Mod(E''), \{w_1, ..., w_n\}) = Mod(\kappa)$$

from which we can finally conclude

$$\begin{aligned} proj(Mod(S_{\dot{-}_D}(\kappa, \phi)), X) &= proj(Mod(S_{\dot{-}_D}(\kappa, \phi)), Z) \\ &\quad \cup\ proj(Mod(S_{\dot{-}_D}(\kappa, \phi)), W) \\ &= Mod(\kappa \dot{+}_D \neg\phi) \cup Mod(\kappa) \\ &= Mod(\kappa \dot{-}_D \phi) \end{aligned}$$

□

Lemma 5. *For the formula*

$$\begin{aligned} G = \kappa^y \wedge \mu \wedge \bigwedge_{1 \leq j \leq n} \Big(&(d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j) \\ &\wedge (\neg d_j \vee \neg x_j \vee \neg y_j) \wedge (\neg d_j \vee x_j \vee y_j) \Big) \end{aligned} \tag{5}$$

where κ^y is a copy of κ, wherein each variable $x_j \in \{x_1, ..., x_n\}$ is replaced by a new variable $y_j \in \{y_1, ..., y_n\}$ and $\{d_1, ..., d_n\}$ are discrepancy variables, the following holds:

$$proj(Mod(G), \{d_1, ..., d_n\}) = D_S^v(\kappa, \mu)$$

where $D_S^v(\kappa, \mu)$ is the set of bit vector representations of all difference sets between $Mod(\kappa)$ and $Mod(\mu)$.

Proof. Due to Lemma 1 we know that each model of $\kappa^y \wedge \mu$ is a combination of a model of κ and a model of μ, with every possible combination of models of κ and models of μ being covered. By looking at the truth table of

$$(d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j) \wedge\ (\neg d_j \vee \neg x_j \vee \neg y_j) \wedge (\neg d_j \vee x_j \vee y_j)$$

it is further evident, that this still holds even after adding this formula since this formula is satisfiable for any truth assignments to x_j and y_j:

x_j	y_j	d_j	$(d_j \vee \neg x_j \vee y_j) \wedge (d_j \vee x_j \vee \neg y_j) \wedge (\neg d_j \vee \neg x_j \vee \neg y_j) \wedge (\neg d_j \vee x_j \vee y_j)$
T	T	T	F
T	T	F	T
T	F	T	T
T	F	F	F
F	T	T	T
F	T	F	F
F	F	T	F
F	F	F	T

From the truth table we can further see that $d_j = 0$ whenever $x_j = y_j$ and $d_j = 1$ whenever $x_j \neq y_j$. Accordingly, given a model $M \in Mod(G)$, the bit vector $proj(M, \{d_1, ..., d_n\})$ is the bit vector representation of the difference set between the bit vector $proj(M, \{x_1, ..., x_n\})$, which is a model of μ, and the bit vector $proj(M, \{y_1, ..., y_n\})$, which is a model of κ. Since as mentioned above each combination of models of κ and models of μ is covered by a model of G, we can thus conclude that $proj(Mod(G), \{d_1, ..., d_n\}) = D_S^v(\kappa, \mu)$. □

Lemma 6. *Each model M of the formula $G \wedge G'$, with G and G' as in equations 5 and 6, contains bit vector representations of two difference sets: $proj(M, \{d_1, ..., d_n\}) \in D_S^v(\kappa, \mu)$ and $proj(M, \{d'_1, ..., d'_n\}) \in D_S^v(\kappa, \mu)$. Furthermore, each possible combination of two difference sets of $D_S^v(\kappa, \mu)$ is addressed by a model of $G \wedge G'$.*

Proof. Lemma 6 is a direct consequence of Lemma 5 as $Var(G) \cap Var(G') = \emptyset$ and G' is identical to G apart from differing variables. □

Lemma 7. *Each model M of the formula*

$$H = G \wedge G' \wedge \bigwedge_{1 \leq j \leq n} \Big((d''_j \vee \neg d_j \vee d'_j) \wedge (d''_j \vee d_j \vee \neg d'_j) \\ \wedge (\neg d''_j \vee \neg d_j \vee \neg d'_j) \wedge (\neg d''_j \vee d_j \vee d'_j) \Big) \wedge \bigvee_{1 \leq j \leq n} d''_j \tag{7}$$

contains bit vector representations of two difference sets: $proj(M, \{d_1, ..., d_n\}) \in D_S^v(\kappa, \mu)$ and $proj(M, \{d'_1, ..., d'_n\}) \in D_S^v(\kappa, \mu)$ with $proj(M, \{d_1, ..., d_n\}) \neq proj(M, \{d'_1, ..., d'_n\})$. Furthermore, each possible combination of two distinct difference sets of $D_S^v(\kappa, \mu)$ is addressed by a model of H.

Proof. From the proof of Lemma 5 we know that the conjunction ensures that $d''_j = 0$ whenever $d_j = d'_j$ and $d''_j = 1$ whenever $d_j \neq d'_j$. Furthermore, since this conjunction is satisfiable for all truth assigments to d_j and d'_j, adding the conjunction to the formula $G \wedge G'$ does not impact on the insights from Lemma 6. By adding the disjunction, however, the case that all variables $\{d''_1, ..., d''_n\}$ are *false*, which happens when $proj(M, \{d_1, ..., d_n\}) = proj(M, \{d'_1, ..., d'_n\})$, is no longer possible. As a consequence, for each model $M \in Mod(H)$ the following holds: $proj(M, \{d_1, ..., d_n\}) \neq proj(M, \{d'_1, ..., d'_n\})$. In addition, as demonstrated in Lemma 6, each possible combination of two difference sets of $D_S^v(\kappa, \mu)$ is addressed by a model of H, with the new restriction that the two difference sets are distinct. □

Lemma 8. *Each model M of the formula*

$$I = H \wedge \bigwedge_{1 \leq j \leq n} \Big(d_j \vee \neg d'_j \Big) \tag{8}$$

contains bit vector representations of two difference sets $proj(M, \{d_1, ..., d_n\}) \in D_S^v(\kappa, \mu)$ and $proj(M, \{d'_1, ..., d'_n\}) \in D_S^v(\kappa, \mu)$ with the second one being a proper subset of the first one. Furthermore, the set $proj(Mod(I), \{d'_1, ..., d'_n\})$ corresponds to the complete set of difference sets in $D_S^v(\kappa, \mu)$, that are proper subsets of some difference set.

Proof. With regards to the first statement, the constraint $d_j \vee \neg d'_j$ ensures that $d_j = 1$, whenever $d'_j = 1$, since otherwise $d_j \vee \neg d'_j$ evaluates to *false*. Combined with the insights from Lemma 7 on formula H, in particular the fact that $proj(M, \{d_1, ..., d_n\}) \neq proj(M, \{d'_1, ..., d'_n\})$, we obtain that the requirements for being a proper subset are fulfilled for the difference set represented by $proj(M, \{d'_1, ..., d'_n\})$. The second statement can be derived from the insight of Lemma 7 that each possible combination of two distinct difference sets of $D_S^v(\kappa, \mu)$ is addressed by a model of H. □

Theorem 5. *For the optimal solution S to the partial MaxSat encoding $S^O_{\dot{+}_S}(\kappa, \mu)$ the bit vector $proj(S, \{d'_1, ..., d'_n\})$ corresponds to the bit vector representation of a minimal set of the set of all difference sets $D_S(\kappa, \mu)$, that is also a proper subset of some set in $D_S(\kappa, \mu)$.*

Proof. Since the encoding's hard clauses are equal to the above defined formula I, we know from Lemma 8 that $proj(S, \{d'_1, ..., d'_n\})$ is a bit vector representation of a set $m \in D_S(\kappa, \mu)$, that is a proper subset of at least one set in $D_S(\kappa, \mu)$. The soft clauses attempt to minimize the number of variables in $\{d'_1, ..., d'_n\}$, that have a value of 1. As a result, m is the smallest set in $D_S(\kappa, \mu)$, that is a proper subset of some other set in $D_S(\kappa, \mu)$, from which we can conclude that m is a minimal set of $D_S(\kappa, \mu)$ since we know that there is no other set in $D_S(\kappa, \mu)$, that is a proper subset of some set and that contains fewer elements than m. □

Theorem 6. *For the SAT encoding $S_{\dot{+}_S}(\kappa, \mu)$ following relation holds: $proj(Mod(S_{\dot{+}_S}(\kappa, \mu)), \{x_1, ..., x_n\}) = Mod(\kappa \dot{+}_S \mu)$*

Proof. Let us first consider the case $min_S^p(\kappa,\mu) = \emptyset$, for which $S_{\dot{+}_S}(\kappa,\mu) = \mu$. In this case we know that there exists no set in $D_S(\kappa,\mu)$, that is a proper subset of some set in $D_S(\kappa,\mu)$. Hence, no set in $D_S(\kappa,\mu)$ has a proper subset and therefore each set in $D_S(\kappa,\mu)$ is a minimal set. Looking at Definition 2.8 of Satoh's revision operator, it is evident, that in this case the models of the revision result are exactly the models of the revision formula μ, hence $Mod(\kappa \dot{+}_S \mu) = Mod(\mu) = proj(Mod(S_{\dot{+}_S}(\kappa,\mu)), \{x_1,...,x_n\})$.
Next, we consider the case $min_S^p(\kappa,\mu) = \{\emptyset\}$, i.e. the empty set is the only minimal set. The empty set is then a difference set in $D_S(\kappa,\mu)$ and therefore there exists at least one model in μ, that is also a model of κ. As a consequence, the minimum Dalal distance $d_{min}(\kappa,\mu) = 0$. Hence, we can define the encoding $S_{\dot{+}_S}(\kappa,\mu)$ as the encoding $S_{\dot{+}_D}(\kappa,\mu)$ with a minimum Dalal distance of 0. As we know from Theorem 2, that $proj(Mod(S_{\dot{+}_D}(\kappa,\mu)), \{x_1,...,x_n\}) = Mod(\kappa \dot{+}_D \mu)$, we can conclude that $proj(Mod(S_{\dot{+}_S}(\kappa,\mu)), \{x_1,...,x_n\}) = Mod(\kappa \dot{+}_S \mu)$ for the case $min_S^p(\kappa,\mu) = \{\emptyset\}$.
Finally, in all other cases the encoding is a conjunction of formula G from Lemma 5 and encoding $L(\{d_1,...,d_n\}, min_S^p(\kappa,\mu))$. From Lemma 5 we know that $proj(Mod(G), \{d_1,...,d_n\}) = D_S^v(\kappa,\mu)$. Further, it is known that $proj(Mod(G), \{x_1,...,x_n\}) = Mod(\mu)$. As shown above, encoding $L(\{d_1,...,d_n\}, min_S^p(\kappa,\mu))$ ensures that for every minimal set $m \in min_S^p(\kappa,\mu)$ no difference set represented by variables $\{d_1,...,d_n\}$, contains more elements than m AND all elements of m at the same time. That way it is ensured that no difference set represented by variables $\{d_1,...,d_n\}$ is a non-minimal set and thus all these sets are minimal sets. As a consequence, $proj(Mod(S_{\dot{+}_S}(\kappa,\mu)), \{x_1,...,x_n\})$ contains only those models of μ, for which a model of κ exists, such that $d_S(\kappa,\mu)$ is a minimal set of $D_S(\kappa,\mu)$, which is precisely the definition of Satoh's revision, and hence $proj(Mod(S_{\dot{+}_S}(\kappa,\mu)), \{x_1,...,x_n\}) = Mod(\kappa \dot{+}_S \mu)$. □

Theorem 7. *For the optimal solution S to the partial MaxSat optimization encoding $S^O_{\dot{-}_S}(\kappa,\phi)$, the bit vector $proj(S, \{d'_1,...,d'_n\})$ corresponds to the bit vector representation of a minimal set of the set of all difference sets $D_S(\kappa,\neg\phi)$, that is a proper subset of some set in $D_S(\kappa,\neg\phi)$.*

Proof. Encoding $S^O_{\dot{-}_S}(\kappa,\phi)$ is almost identical to encoding $S^O_{\dot{+}_S}(\kappa,\phi)$ of the corresponding revision Theorem 5 since $t(\neg\phi)$ and μ are both CNF formulae, with the only difference being that $t(\neg\phi)$ additionally contains some new auxiliary variables $V^t_{\neg\phi}$, that are introduced by the Tseitin transformation. However, as $proj(Mod(t(\neg\phi)), Var(\neg\phi)) = Mod(\neg\phi)$ (CNF formulae resulting from Tseitin transformations are equisatisfiable to the initial formulae) and since these new variables occur only within $t(\neg\phi)$, their presence can be neglected and the remaining proof is thus identical to the proof of Theorem 5. □

Theorem 8. *For the SAT encoding $S_{\dot{-}_S}(\kappa,\phi)$ following relation holds: $proj(Mod(S_{\dot{-}_S}(\kappa,\phi)), \{x_1,...,x_n\}) = Mod(\kappa \dot{-}_S \phi)$*

Proof. For the case $min_S^p(\kappa,\neg\phi) = \emptyset$ we can conclude, analogously to the proof of Theorem 6, that every set in $D_S(\kappa,\neg\phi)$ is a minimal set and thus $Mod(\kappa \dot{+}_S \neg\phi) = Mod(\neg\phi)$. As a result, $Mod(\kappa \dot{-}_S \phi) = Mod(\kappa) \cup Mod(\neg\phi)$. Looking at the encoding $S_{\dot{-}_S}(\kappa,\phi)$, we can see that $proj(Mod(S_{\dot{-}_S}(\kappa,\phi)), Y) = Mod(\kappa)$ and $proj(Mod(S_{\dot{-}_S}(\kappa,\phi)), Z) = Mod(\neg\phi)$. Further, due to Lemma 4 $proj(Mod(F(X,Y,Z,n)), X) = proj(Mod(F(X,Y,Z,n)), Y) \cup proj(Mod(F(X,Y,Z,n)), Z)$. Accordingly, $proj(Mod(S_{\dot{-}_S}(\kappa,\phi)), X) = Mod(\kappa) \cup Mod(\neg\phi) = Mod(\kappa \dot{-}_S \phi)$.
For the case $min_S^p(\kappa,\neg\phi) = \{\emptyset\}$, we can conclude analogously to the proof of Theorem 6, that $d_{min}(\kappa,\neg\phi) = 0$ and can also define the encoding as the encoding $S_{\dot{-}_D}(\kappa,\phi)$ with a minimum Dalal distance of 0. Due to Theorem 4, then $proj(Mod(S_{\dot{-}_S}(\kappa,\phi)), \{x_1,...,x_n\}) = Mod(\kappa) \cup Mod(\kappa \dot{+}_D \neg\phi) = Mod(\kappa \dot{-}_D \phi)$.
Lastly, in all other cases the encoding $S_{\dot{-}_S}(\kappa,\phi)$ is similar to encoding $S_{\dot{+}_S}(\kappa,\phi)$, with only a few adjustments. Let us denote the part of the encoding before the conjunct κ^w as C. C corresponds to encoding $S_{\dot{+}_S}(\kappa,\phi)$ after replacing the formula μ with $t(\neg\phi)^z$. Since $Mod(t(\neg\phi)^z) = Mod(\neg\phi)$ and as the contained auxiliary variables can be, as usual, neglected, Theorem 6 allows us to state $proj(Mod(C), \{z_1,...,z_n\}) = Mod(\kappa \dot{+}_S \neg\phi)$. Adding $\kappa^w \wedge F(X,Z,W,n)$ to C finally leads to $proj(Mod(S_{\dot{-}_S}(\kappa,\phi)), W) = Mod(\kappa)$. Since $proj(Mod(S_{\dot{-}_S}(\kappa,\phi)), Z) = Mod(\kappa \dot{+}_S \neg\phi)$ still holds and due to Lemma 4, we obtain $proj(Mod(S_{\dot{-}_S}(\kappa,\phi)), X) = Mod(\kappa) \cup Mod(\kappa \dot{+}_S \neg\phi) = Mod(\kappa \dot{-}_S \phi)$. □

Theorem 9. *The inference check SAT encoding $S_I(B,\gamma)$ is unsatisfiable if and only if $B \models \gamma$.*

Proof. We address solely the case $B = (\kappa \dot{+}_D \mu)$, since the other three cases can be proven analogously. From Theorem 2 we know that $proj(Mod(S_{\dot{+}_D}(\kappa,\mu)), \{x_1, ..., x_n\}) = Mod(\kappa \dot{+}_D \mu)$. Further, $proj(Mod(t(\neg\gamma)), Var(\neg\gamma)) = Mod(\neg\gamma)$ because the Tseitin transformation results in a formula, that is equisatisfiable to the initial formula. Moreover, $Var(\gamma) \subseteq \{x_1, ..., x_n\}$ by definition.

Let us first assume that $S_{\dot{+}_D}(\kappa,\mu) \wedge t(\neg\gamma)$ is unsatisfiable. As $V^t_{\neg\gamma} \cap Var(S_{\dot{+}_D}(\kappa,\mu)) = \emptyset$, there is only one conceivable reason for the formula being unsatisfiable: $proj(Mod(S_{\dot{+}_D}(\kappa,\mu)), Var(\neg\gamma)) \cap proj(Mod(t(\neg\gamma)), Var(\neg\gamma)) = \emptyset$. From this we obtain $Mod(\kappa \dot{+}_D \mu) \cap Mod(\neg\gamma) = \emptyset$, i.e. there is no model of B, that does not satisfy γ, or in other words every model of B satisfies γ, hence $B \models \gamma$.

Let us now assume that $B \models \gamma$. This means that there is no model of $\kappa \dot{+}_D \mu$, for which $\neg\gamma$ evaluates to true. We obtain $proj(Mod(S_{\dot{+}_D}(\kappa,\mu)), Var(\gamma)) \cap proj(Mod(t(\neg\gamma)), Var(\gamma)) = \emptyset$, which implies that $S_I(B,\gamma) = S_{\dot{+}_D}(\kappa,\mu) \wedge t(\neg\gamma)$ is unsatisfiable. □

Theorem 10. *The model check SAT encoding $S_M(B,N)$ is satisfiable if and only if $N \in Mod(B)$.*

Proof. We address solely the case $B = (\kappa \dot{+}_D \mu)$, since the other three cases can be proven analogously.

Let us first assume that $S_M(B,N)$ is satisfiable. Each clause in the conjunction $m(N,X)$ consists by definition of exactly one literal x_j (if $value(N,x_j) = true$) or $\neg x_j$ (if $value(N,x_j) = false$) with $x_j \in X$. From this we can conclude that $Mod(m(N,X)) = \{N\}$. From Theorem 2 we further know that $proj(Mod(S_{\dot{+}_D}(\kappa,\mu)), \{x_1, ..., x_n\}) = Mod(\kappa \dot{+}_D \mu)$. As $S_{\dot{+}_D}(\kappa,\mu) \wedge m(N,X)$ is satisfiable (by assumption), we know that $proj(Mod(S_{\dot{+}_D}(\kappa,\mu)), \{x_1, ..., x_n\}) \cap Mod(m(N,X)) \neq \emptyset$ and thus obtain $proj(Mod(S_{\dot{+}_D}(\kappa,\mu)), \{x_1, ..., x_n\}) \cap Mod(m(N,X)) = \{N\}$. This last expression implies that $N \in proj(Mod(S_{\dot{+}_D}(\kappa,\mu)), \{x_1, ..., x_n\})$, i.e. $N \in Mod(B)$.

Let us now assume that $N \in Mod(B)$, i.e. $N \in Mod(\kappa \dot{+}_D \mu)$. This means that $N \in proj(Mod(S_{\dot{+}_D}(\kappa,\mu)), \{x_1, ..., x_n\})$ and further $proj(Mod(S_{\dot{+}_D}(\kappa,\mu)), \{x_1, ..., x_n\}) \cap Mod(m(N,X)) = \{N\}$. Looking at the encoding $S_M(B,N) = S_{\dot{+}_D}(\kappa,\mu) \wedge m(N,X)$ it is evident that this last expression implies that $S_M(B,N)$ is satisfiable. □

Theorem 11. *The optimal value of the optimal solution to $I^O_{\dot{+}_D}(\kappa,\mu)$ corresponds to the minimum Dalal distance $d_{min}(\kappa,\mu)$.*

Proof. Encoding $I^O_{\dot{+}_D}(\kappa,\mu)$ is a direct translation of the SAT encoding scheme $S^O_{\dot{+}_D}(\kappa,\mu)$ into ILP constraints, where all parts of $I^O_{\dot{+}_D}(\kappa,\mu)$ without the final optimization constraint correspond to the hard clauses of $S^O_{\dot{+}_D}(\kappa,\mu)$ and the final optimization constraint corresponds exactly to the soft clauses of $S^O_{\dot{+}_D}(\kappa,\mu)$. For this reason this theorem follows from Theorem 1. □

Theorem 12. *For the set S of bit vector representations of all solutions to $I_{\dot{+}_D}(\kappa,\mu)$ the following relation holds:*

$$proj(S, X^I) = Mod(\kappa \dot{+}_D \mu)$$

Proof. Encoding $I_{\dot{+}_D}(\kappa,\mu)$ is a direct translation of the SAT encoding scheme $S_{\dot{+}_D}(\kappa,\mu)$ into ILP constraints, where the final constraint $\sum_{j=1}^{n} \left(d^I_j\right) = d_{min}(\kappa,\mu)$ fulfills the same purpose as $E(\{d_1, ..., d_n\}, d_{min}(\kappa,\mu))$ in $S_{\dot{+}_D}(\kappa,\mu)$. Consequently, this theorem follows from Theorem 2. □

Lemma 9. *Given a CNF formula α and two sets of binary ILP variables P and Q with $|P| = |Var(\alpha)|$, $|Q| = |C(\alpha)|$ and $P \cap Q = \emptyset$, for the set S of bit vector representations of all solutions to $con_N(\alpha, X, A)$ the following holds: $proj(S,P) = Mod(\neg\alpha)$*

Proof. Let us first have a look at how a clause is translated into a constraint in $con_N(\alpha, P, Q)$: each positive literal is replaced by a positive ILP expression (b_i), that evaluates to 1, whenever $b_i = 1$. Each negative literal is replaced by a negative ILP expression ($1 - b_i$), that evaluates to 1, when $b_i = 0$. Thus far, this is the usual way of how a clause is translated into an ILP constraint. However, while the usual translation rule would now ensure that the sum of all ILP expressions is greater than or equal to 1 (which corresponds to a disjunction), instead the sum is divided by the number of literals in the clause and it is ensured that the result is smaller than or equal to the value of the auxiliary binary ILP variable, that was explicitly created for the clause. Hence, in the case that none of the addends of the sum evaluate to 1 (the disjunction evaluates to *false*), the left part of the inequality equation evaluates to 0. In

case all of the addends evaluate to 1, it evaluates to 1, whereas in all other cases it evaluates to a value somewhere between 0 and 1. As a consequence, the auxiliary variable on the right side of the inequality equation must have the value 1 whenever at least one of the addends evaluates to 1 (i.e. the left side has a value > 0) since the relation is of the type '$\leq$'. In the case that none of the addends evaluates to 1, the auxiliary variable can be 0 or 1. Given a value assignment to the variables in P, we can conclude that if a clause's auxiliary variable has the value 0, then the clause evaluates to *false*, since otherwise the auxiliary variable would have the value 1. The last constraint of $con_N(\alpha, P, Q)$ implements the restriction that the sum of all auxiliary variables must be smaller than or equal to the number of clauses minus 1. This constraint can only be fulfilled when at least one of the auxiliary variables has the value 0 due to $|Q| = |C(\alpha)|$. Hence, all solutions to the given ILP program consist of value assignments to the variables in P, that make at least one clause evaluate to *false*, which corresponds to $\neg\alpha$; thus $proj(S, P) = Mod(\neg\alpha)$. □

Theorem 13. *The optimal value of the optimal solution to $I^O_{\dot{-}_D}(\kappa, \phi)$ corresponds to $d_{min}(\kappa, \neg\phi)$.*

Proof. $I^O_{\dot{+}_D}(\kappa, \mu)$ (defined in Section 3.4.1) and $I^O_{\dot{-}_D}(\kappa, \phi)$ differ only in that $I^O_{\dot{-}_D}(\kappa, \phi)$ contains additional definition lines for the set of auxiliary variables A^I and in that $con_N(\phi, Z^I, A^I)$ is used instead of $con(\mu, Z^I)$ and variables Z^I instead of X^I. The objective of the set of constraints $con(\mu, Z^I)$ is to ensure that the variables of Z^I represent the models of μ. Due to Lemma 9 we know that the constraints $con_N(\phi, Z^I, A^I)$ do the same for the models of $\neg\phi$. Further, the new variables A^I do not occur in any other constraints, but in $con_N(\phi, Z^I, A^I)$, and can thus be neglected. From Theorem 11 and the above elaborations we can then conclude Theorem 13. □

Theorem 14. *For the set S of bit vector representations of all solutions to $I_{\dot{-}_D}(\kappa, \phi)$ the following relation holds:*

$$proj(S, X^I) = Mod(\kappa \mathbin{\dot{-}_D} \phi)$$

Proof. For an ILP program I^1 consisting of the constraints $con(\kappa, Y^I)$, $con_N(\phi, Z^I, A^I)$, $con_D(D^I, Y^I, Z^I)$ and $\sum_{j=1}^{n}\left(d_j^I\right) = d_{min}(\kappa, \neg\phi)$ only, the following holds: $proj(S^1, Z^I) = Mod(\kappa \dot{+} \neg\phi)$, where S^1 is the set of bit vector representations of all solutions to I^1. This follows from Theorem 12. Let adding the constraint $con(\kappa, B^I)$ to I^1 result in the new interim encoding I^2. For the set S^2 of bit vector representations of all solutions to I^2, $proj(S^2, Z^I) = Mod(\kappa \dot{+} \neg\phi)$ still holds, since none of the variables in $Y^I \cup Z^I \cup A^I \cup D^I$ occur in the new set of constraints and the new variables B^I do not occur in I^1. Further, $proj(S^2, B^I) = Mod(\kappa)$ due to the definition of $con(\kappa, B^I)$.
Let us denote the interim encoding obtained by adding $con_D(E^I, X^I, B^I)$ and $con_D(F^I, X^I, Z^I)$ to I^2 by I^3. Then for the set S^3 of bit vector representations of all solutions to I^3, $proj(S^3, Z^I) = Mod(\kappa \dot{+} \neg\phi)$ and $proj(S^3, B^I) = Mod(\kappa)$ still hold because the new constraints do not place any restrictions on the variables X^I, Z^I and B^I, but only on the variables E^I and F^I by forcing the variable e_j^I (f_j^I) to take the value 1, whenever $x_j^I \neq b_j^I$ ($x_j^I \neq z_j^I$). Since variables E^I and F^I do not occur in any other constraints, they can satisfy these new constraints.
Let adding constraints $\left(\sum_{j=1}^{n}\left(e_j^I\right)\right) \div n \leq g_1^I$ and $\left(\sum_{j=1}^{n}\left(f_j^I\right)\right) \div n \leq g_2^I$ to I^3 result in interim encoding I^4. For the set S^4 of bit vector representations of all solutions to I^4, $proj(S^4, Z^I) = Mod(\kappa \dot{+} \neg\phi)$ and $proj(S^4, B^I) = Mod(\kappa)$ still hold because the newly added constraints do not impact the value assignments of any variables apart from the variables g_1^I and g_2^I, which do not occur in any other constraints. The variable g_1^I (g_2^I) must have the value 1, when the sum of the values of variables E^I (F^I) is greater than 0. In case the sum is equal to 0, g_1^I (g_2^I) can be 0 or 1. From this we can conclude that whenever $value(s^4, g_1^I) = 0$ for a solution $s^4 \in S^4$, then $proj(s^4, X^I) = proj(s^4, B^I) = Mod(\kappa)$ and whenever $value(s^4, g_2^I) = 0$, then $proj(s^4, X^I) = proj(s^4, Z^I) = Mod(\kappa \dot{+} \neg\phi)$.

The final constraint $g_1^I + g_2^I \leq 1$ ensures that at least one of the variables g_1^I and g_2^I is 0. This requirement impacts the value assignment of the variables E^I (F^I), forcing them to take the value 0, whenever possible, and thus indirectly that of the variables X^I, since they are not bound by any further constraints, whereas the variables B^I (Z^I) are. The need for g_1^I or g_2^I to have the value 0 signifies that in every solution $s \in S$ to the ILP encoding $I_{\dot{-}_D}(\kappa, \phi)$ one or both (in case $d_{min}(\kappa, \neg\phi) = 0$) of $proj(s, X^I) = proj(s, B^I) = Mod(\kappa)$ and $proj(s, X^I) = proj(s, Z^I) = Mod(\kappa \dot{+} \neg\phi)$ need to hold. For the entire set S of solutions, this leads to $proj(S, X^I) = Mod(\kappa) \cup Mod(\kappa \dot{+}_D \neg\phi) = Mod(\kappa \mathbin{\dot{-}_D} \phi)$. □

Theorem 15. *Given the bit vector representation S of the optimal solution to $I^O_{\dot{+}_S}(\kappa, \mu)$ the bit vector $proj(S, \{ds_1^I, ..., ds_n^I\})$ represents a minimal set of the set of all difference sets $D_S(\kappa, \mu)$, that is also a proper subset of some set in $D_S(\kappa, \mu)$.*

Proof. Encoding $I^O_{+_S}(\kappa, \mu)$ is a direct translation of the SAT encoding scheme $S^O_{+_S}(\kappa, \mu)$ into ILP constraints, where all parts of $I^O_{+_S}(\kappa, \mu)$ without the final optimization constraint correspond to the hard clauses of $S^O_{+_S}(\kappa, \mu)$ and the final optimization constraint corresponds exactly to the soft clauses of $S^O_{+_S}(\kappa, \mu)$. For this reason this theorem follows from Theorem 5. □

Lemma 10. *Constraints $con_{MS}(min^p_S(\kappa, \mu), D^I, S^I)$ ensure that the set represented by variables D^I is not a proper superset of any of the sets represented by the bit vectors $min^p_S(\kappa, \mu)$.*

Proof. For a set a to be a proper superset of a set b, it must contain all elements of b and more elements than b. The first constraint for every $m \in min^p_S(\kappa, \mu)$ in $con_{MS}(min^p_S(\kappa, \mu), D^I, S^I)$ ensures that the auxiliary variable s^I_1 is 1 whenever the set represented by variables D^I contains all elements of the minimal set represented by m since in this case the left side of the inequality equation evaluates to 1. In all other cases s^I_1 can be both 0 and 1. The second constraint guarantees that the auxiliary variable s^I_2 is 1 whenever the set represented by variables D^I contains more elements than the minimal set represented by m since in this case the left side of the inequality equation evaluates to 1 or to a value between 0 and 1. In all other cases s^I_2 can be both 0 and 1. Finally, the third constraint implements the restriction that the sum of s^I_1 and s^I_2 must be smaller than or equal to 1, ensuring that s^I_1 and s^I_2 cannot be 1 at the same time. It follows that the set represented by variables D^I cannot be a proper superset of any of the sets represented by the bit vectors $min^p_S(\kappa, \mu)$. □

Theorem 16. *For the set S of bit vector representations of all solutions to $I_{\dot{+}_S}(\kappa, \mu)$ the following relation holds: $proj(S, X^I) = Mod(\kappa \dot{+}_S \mu)$*

Proof. Encoding $I_{\dot{+}_S}(\kappa, \mu)$ is a direct translation of the SAT encoding scheme $S_{\dot{+}_S}(\kappa, \mu)$ into ILP constraints, where constraints $con_{MS}(min^p_S(\kappa, \mu), D^I, S^I)$ fulfill the same purpose as $L(\{d_1, ..., d_n\}, min^p_S(\kappa, \mu))$ in $S_{\dot{+}_S}(\kappa, \mu)$ as established by Lemma 10. Consequently, this theorem follows from Theorem 6. □

Theorem 17. *For the bit vector representation S of the optimal solution to $I^O_{\dot{-}_S}(\kappa, \phi)$ the bit vector $proj(S, \{ds^I_1, ..., ds^I_n\})$ corresponds to the bit vector representation of a minimal set of the set of all difference sets $D_S(\kappa, \neg\phi)$, that is also a proper subset of some set in $D_S(\kappa, \neg\phi)$.*

Proof. $I^O_{+_S}(\kappa, \mu)$ (defined in Section 3.4.3) and $I^O_{\dot{-}_S}(\kappa, \phi)$ differ only in that $I^O_{\dot{-}_S}(\kappa, \phi)$ contains additional definition lines for the sets of auxiliary variables A^I and AS^I and in that $con_N(\phi, Z^I, A^I)$ replaces $con(\mu, X^I)$ and $con_N(\phi, ZS^I, AS^I)$ replaces $con(\mu, ZS^I)$. Moreover, variables Z^I are used instead of variables X^I. The objective of the set of constraints $con(\mu, Z^I)$ ($con(\mu, ZS^I)$) is to ensure that the variables of Z^I (ZS^I) represent the models of μ. Due to Lemma 9 we know that the constraints $con_N(\phi, Z^I, A^I)$ ($con_N(\phi, ZS^I, AS^I)$) do the same for the models of $\neg\phi$. Further, the new variables A^I and AS^I do not occur in any other constraints, but in $con_N(\phi, Z^I, A^I)$ and $con_N(\phi, ZS^I, AS^I)$, and can thus be neglected. From Theorem 15 and the above elaborations we can then conclude Theorem 17. □

Theorem 18. *For the set S of bit vector representations of all solutions to $I_{\dot{-}_S}(\kappa, \phi)$ the following relation holds: $proj(S, X^I) = Mod(\kappa \dot{-}_S \phi)$*

Proof. For the case $min^p_S(\kappa, \neg\phi) = \emptyset$ the encoding $I_{\dot{-}_S}(\kappa, \phi)$ is similar to encoding $I_{\dot{-}_D}(\kappa, \phi)$ with the difference that the parts $con(\kappa, Y^I)$, $con_D(D^I, Y^I, Z^I)$ and $\sum_{j=1}^{n} \left(d^I_j\right) = d_{min}(\kappa, \neg\phi)$ as well as the variable sets Y^I and D^I are absent.
Accordingly, the only remaining restriction on the variable set Z^I is that these variables must represent models of the formula $\neg\phi$. From this, combined with Theorem 14, we obtain that the variables X^I represent models of κ and models of $\neg\phi$, i.e. $proj(S, X^I) = Mod(\kappa) \cup Mod(\neg\phi))$. Since the assumed case $min^p_S(\kappa, \neg\phi) = \emptyset$ implies that all sets in $D_S(\kappa, \neg\phi)$ are minimal sets we can conclude $Mod(\kappa \dot{-}_S \phi) = Mod(\kappa) \cup Mod(\neg\phi))$ and thus $proj(S, X^I) = Mod(\kappa \dot{-}_S \phi)$.
Next, we address the case $min^p_S(\kappa, \neg\phi) = \{\emptyset\}$, which implies that the empty set is the only minimal set of $D_S(\kappa, \neg\phi)$. Accordingly, the empty set is a member of $D_S(\kappa, \neg\phi)$ and hence $d_{min}(\kappa, \neg\phi) = 0$. Hence, we can define the encoding $I_{\dot{-}_S}(\kappa, \phi)$ as encoding $I_{\dot{-}_D}(\kappa, \phi)$ with a minimum Dalal distance of 0. From Theorem 14 we then obtain $proj(S, X^I) = Mod(\kappa \dot{-}_S \phi)$.

Finally, we consider the remaining cases, for which encoding $I_{\dot{-}_S}(\kappa, \phi)$ is identical to encoding $I_{\dot{-}_D}(\kappa, \phi)$ except for the replacement of $\sum_{j=1}^{n} \left(d_j^I\right) = d_{min}(\kappa, \neg\phi)$ with $con_{MS}(min_S^p(\kappa, \neg\phi), D^I, S^I)$ and the additional variable set S^I. From Lemma 10 we know that $con_{MS}(min_S^p(\kappa, \neg\phi), D^I, S^I)$ ensure that the set represented by variables D^I is not a proper superset of any of the sets represented by the bit vectors $min_S^p(\kappa, \neg\phi)$. Accordingly, the set represented by variables D^I is always a minimal set of $D_S(\kappa, \neg\phi)$. Analogously to the proof of Theorem 14 we then obtain $proj(S, X^I) = Mod(\kappa) \cup Mod(\kappa \dot{+}_S \neg\phi)$ which is precisely the definition of Satoh's contraction, hence $proj(S, X^I) = Mod(\kappa \dot{-}_S \phi)$. □

Theorem 19. *The inference check ILP encoding $I_I(B, \gamma)$ has no solution if and only if $B \models \gamma$.*

Proof. We address solely the case $B = (\kappa \dot{+}_D \mu)$ since the other three cases can be proven analogously. Due to Theorem 12 and Lemma 9 $proj(S, X^I) = Mod(\kappa \dot{+}_D \mu)$ for the set S of bit vector representations of all solutions to $I_{\dot{+}_D}(\kappa, \mu)$ and $proj(S', X^I) = Mod(\neg\gamma)$ for the set S' of bit vector representations of all solutions to $con_N(\gamma, X^I, A_2^I)$.

Let us first assume that encoding $I_I((\kappa \dot{+}_D \mu), \gamma)$ has no solution. As $I_{\dot{+}_D}(\kappa, \mu)$ does not contain any variables of the set A_2^I, and $con_N(\gamma, X^I, A_2^I)$ does not contain any further variables other than X^I, the reason for there not being any solution must be $proj(S, X^I) \cap proj(S', X^I) = \emptyset$, i.e. $Mod(\kappa \dot{+}_D \mu) \cap Mod(\neg\gamma) = \emptyset$. In other words, every model of $\kappa \dot{+}_D \mu$ (there exists at least one due to the restriction that B represents a consistent belief base) satisfies γ, hence $B \models \gamma$.

Let us now assume that $B \models \gamma$, which means that there is no model of $\kappa \dot{+}_D \mu$, for which $\neg\gamma$ evaluates to *true*. Accordingly, $proj(S, X^I) \cap proj(S', X^I) = \emptyset$, which results in $I_I((\kappa \dot{+}_D \mu), \gamma)$ not having any solution. □

Theorem 20. *The model check ILP encoding $I_M(B, N)$ has a solution if and only if $N \in Mod(B)$.*

Proof. We address solely the case $B = (\kappa \dot{+}_D \mu)$, since the other three cases can be proven analogously. Let us first assume that $I_M(B, N)$ has at least one solution. Theorem 12 states $proj(S, X^I) = Mod(\kappa \dot{+}_D \mu)$ for the set S of bit vector representations of all solutions to $I_{\dot{+}_D}(\kappa, \mu)$. Moreover, constraints $con_M(N, X^I)$ ensure that $proj(S', X^I) = \{N\}$ for the set S' of bit vector representations of all solutions to $con_M(N, X^I)$. As $I_M(B, N)$ by assumption has at least one solution, $Mod(\kappa \dot{+}_D \mu) \cap \{N\} \neq \emptyset$ and thus $N \in Mod(\kappa \dot{+}_D \mu)$, i.e. $N \in Mod(B)$.

Let us now assume that $N \in Mod(B)$, i.e. $N \in Mod(\kappa \dot{+}_D \mu)$ and thus $N \in proj(S, X^I)$. Since the constraints $con_M(N, X^I)$ ensure that $proj(S', X^I) = \{N\}$ for the set S' of bit vector representations of all solutions to $con_M(N, X^I)$, the equation $proj(S, X^I) \cap proj(S', X^I) = \{N\}$ holds and accordingly $I_M(B, N)$ has a solution. □

Theorem 21. *The optimal value of the optimal answer set to the ASP logic program $A^O_{\dot{+}_D}(\kappa, \mu)$ corresponds to the minimum Dalal distance $d_{min}(\kappa, \mu)$.*

Proof. Due to the choice rule the answer set solver is aware that any of the propositional atoms can potentially be *true*. As described in Section 2.8.3, the integrity constraints $constraints[\kappa, Y^I]$ and $constraints[\mu, X^I]$ thus lead to $interpretation(A, Y^I, t/1) = Mod(\kappa)$ and $interpretation(A, X^I, t/1) = Mod(\mu)$ (due to $X^I \cap Y^I = \emptyset$), with A denoting the set of all answer sets. Moreover, the encoding's only two rules ensure that whenever two integers of X^I and Y^I represent the same propositional atom, which is defined by the representation facts, and their corresponding $t/1$ predicate atoms differ in their truth assignments, the $d/1$ predicate atom, that takes the X^I integer as argument, is derived to be *true*. Further, in case that two integers of X^I and Y^I represent the same propositional atom and their corresponding $t/1$ predicate atoms have identical truth assignments, the predicate atom $d/1$, that takes the X^I integer as argument, cannot be derived and is thus *false* due to the concept of default negation. From this we can conclude, that the number of such predicate atoms in an answer set a corresponds to the Dalal distance between $interpretation(a, X^I, t/1)$ and $interpretation(a, Y^I, t/1)$, i.e. the Dalal distance between a model of κ and a model of μ. In addition to that the minimize expression ensures that the answer set solver looks for an optimal answer set, i.e. one in which the number of predicate atoms $d/1$ is smallest. The determined smallest number of predicate atoms then corresponds exactly to the minimum Dalal distance $d_{min}(\kappa, \mu)$ as it is defined as the smallest possible Dalal distance between a model of κ and a model of μ. □

Theorem 22. *For the set A of all answer sets to the ASP logic program $A_{\dot{+}_D}(\kappa, \mu)$ the following relation holds:*

$$interpretation(A, X^I, t/1) = Mod(\kappa \dot{+}_D \mu)$$

Proof. From the proof of Theorem 21 we know that without the minimization expression $interpretation(A, Y^I, t/1) = Mod(\kappa)$ and $interpretation(A, X^I, t/1) = Mod(\mu)$. Moreover, we know that the number of predicate atoms $d/1$ in an answer set a corresponds to the Dalal distance between $interpretation(a, X^I, t/1)$ and $interpretation(a, Y^I, t/1)$, i.e. between a model of κ and a model of μ. Due to the newly added integrity constraint, which ensures that in every answer set the number of predicate atoms $d/1$ is equal to $d_{min}(\kappa, \mu)$, we can thus conclude that $interpretation(A, X^I, t/1)$ contains exactly those models of μ, that have the Dalal distance $d_{min}(\kappa, \mu)$ to at least one model of κ. Since this is precisely the definition of $Mod(\kappa \dot{+}_D \mu)$, we obtain $interpretation(A, X^I, t/1) = Mod(\kappa \dot{+}_D \mu)$. □

Theorem 23. *The optimal value of the optimal answer set to the ASP logic program $A^O_{\dot{-}_D}(\kappa, \phi)$ corresponds to the minimum Dalal distance $d_{min}(\kappa, \neg\phi)$.*

Proof. Encoding $A^O_{\dot{-}_D}(\kappa, \phi)$ is almost identical to encoding $A^O_{\dot{+}_D}(\kappa, \phi)$, with the only difference being the replacement of $constraints[\mu, X^I]$ by the integrity constraint $:- cConstraints[\phi, X^I]$. Hence, neglecting the minimization constraint, we obtain $interpretation(A, X^I, t/1) = Mod(\neg\phi)$ instead of $interpretation(A, X^I, t/1) = Mod(\mu)$ for the set A of all answer sets. Combined with the proof of Theorem 21 we can then conclude that the number of predicate atoms $d/1$ in any answer set a to $A^O_{\dot{-}_D}(\kappa, \phi)$ denotes the Dalal distance between $interpretation(a, X^I, t/1)$ and $interpretation(a, Y^I, t/1)$, meaning the Dalal distance between a model of κ and a model of $\neg\phi$. Analogously to the proof of Theorem 21 we obtain that due to the minimization constraint, the optimal value of the optimal answer set to $A^O_{\dot{-}_D}(\kappa, \phi)$ corresponds to the minimum Dalal distance $d_{min}(\kappa, \neg\phi)$. □

Theorem 24. *For the set A of all answer sets to the ASP logic program $A_{\dot{-}_D}(\kappa, \phi)$ the following relation holds:*

$$interpretation(A, X^I, t/1) = Mod(\kappa \dot{-}_D \phi)$$

.

Proof. Let us first consider lines 1-5 of the encoding in Definition 3.26 and let A' denote the set of all answer sets to an encoding comprising these lines. Due to the choice rule in line 1 the answer set solver is aware that any of the propositional atoms can potentially be true. Further, as described in Section 2.8.3, the integrity constraints $constraints[\kappa, Y^I]$ ensure that $interpretation(A', Y^I, t/1) = Mod(\kappa)$. Moreover, the combination of lines 2, 4 and 5 ensures that whenever two integers of X^I and Y^I represent the same propositional atom, which is defined by the representation facts, and their corresponding $t/1$ predicate atoms differ in their truth assignments, the $d/1$ predicate atom, that takes as argument the X^I integer, is derived to be *true*. From this we can conclude, that the number of such predicate atoms in an answer set $a' \in A'$ corresponds to the Dalal distance between $interpretation(a', X^I, t/1)$ and $interpretation(a', Y^I, t/1)$. Note that so far there are no constraints involving the atoms $X^I = 1, ..., n$. As a consequence, $interpretation(A', X^I, t/1)$ is equal to the set of all possible interpretations of $Var(\kappa) \cup Var(\phi)$, thus $Mod(\kappa) \subseteq interpretation(A', X^I, t/1)$ and $Mod(\kappa \dot{+}_D \neg\phi) \subseteq interpretation(A', X^I, t/1)$. Line 6 of encoding $A_{\dot{-}_D}(\kappa, \phi)$ ensures that the constant $isRevisionModel$ is satisfied whenever there is at least one predicate atom $d/1$ in the answer set, i.e. when for a given answer set a'' to an encoding comprising lines 1-6 the models $interpretation(a'', X^I, t/1)$ and $interpretation(a'', Y^I, t/1)$ are distinct, thus $interpretation(a'', X^I, t/1) \notin Mod(\kappa)$. Next, taking into account the definition of $cConstraints[\phi, X^I]$ in Section 3.5.1, adding line 7 ensures that whenever the models $interpretation(a''', X^I, t/1)$ and $interpretation(a''', Y^I, t/1)$ are distinct for a given answer set a''' to an encoding comprising lines 1-7 and thus the constant $isRevisionModel$ is derived to be $true$, then $interpretation(a''', X^I, t/1) \in Mod(\neg\phi)$. In other words, given the set A''' of all answer sets to the encoding comprising lines 1-7, $interpretation(A''', X^I, t/1) = Mod(\kappa) \cup Mod(\neg\phi)$. Finally, line 8 adds the additional constraint that whenever the models $interpretation(a, X^I, t/1)$ and $interpretation(a, Y^I, t/1)$ are distinct for a given answer set $a \in A$, then model $interpretation(a, X^I, t/1)$ must additionally have the Dalal distance $d_{min}(\kappa, \neg\phi)$ to the model $interpretation(a, Y^I, t/1)$, thus $interpretation(a, X^I, t/1) \in Mod(\kappa \dot{+}_D \neg\phi)$. Accordingly, $interpretation(A, X^I, t/1) = Mod(\kappa) \cup Mod(\kappa \dot{+}_D \neg\phi) = Mod(\kappa \dot{-}_D \phi)$. □

Theorem 25. *For the optimal answer set a to the ASP logic program $A^O_{\dot{+}_S}(\kappa, \mu)$, the bit vector $interpretation(a, Z^I, d2/1)$ corresponds to the bit vector representation of a minimal set of the set of all difference sets $D_S(\kappa, \mu)$, that is a proper subset of some set in $D_S(\kappa, \mu)$.*

Proof. Due to the choice rule the answer set solver is aware that any of the propositional atoms can potentially be *true*. As described in Section 2.8.3, the integrity constraints $constraints[\kappa, Y^I]$, $constraints[\kappa, W^I]$, $constraints[\mu, X^I]$ and $constraints[\mu, Z^I]$ lead to $interpretation(A, Y^I, t/1) = Mod(\kappa)$ and $interpretation(A, X^I, t/1) = Mod(\mu)$ (due to $X^I \cap Y^I = \emptyset$) as well as to $interpretation(A, W^I, t/1) = Mod(\kappa)$ and $interpretation(A, Z^I, t/1) = Mod(\mu)$ (due to $Z^I \cap W^I = \emptyset$), with A denoting the set of all answer sets. Further, $rFacts[X^I, Y^I]$ and the two rules defining predicate $d/1$, ensure that whenever two integers of X^I and Y^I represent the same propositional atom, which is defined by the representation facts, and their corresponding $t/1$ predicate atoms differ in their truth assignments, the $d/1$ predicate atom, that takes the X^I integer as argument, is derived to be *true*. Further, in case that two integers of X^I and Y^I represent the same propositional atom and their corresponding $t/1$ predicate atoms have identical truth assignments, the predicate atom $d/1$, that takes the X^I integer as argument, cannot be derived and is thus *false* due to the concept of default negation. The exact same is ensured for the integers of W^I and Z^I by $r2Facts[Z^I, W^I]$ and the two rules defining predicate $d2/1$. Accordingly, for a given answer set a, the presence of $d/1$ predicate atoms and $d2/1$ predicate atoms in a each describe a difference set $d \in D_S(\kappa, \mu)$ in the following way: for every $d(x_i^I)$ ($d2(x_i^I)$), that is contained in the answer set a, the corresponding propositional atom $x_i \in X$ is an element of d. Next, due to $r3Facts[X^I, Z^I]$, combined with the two rules defining predicate $d3/1$ and the integrity constraint $:- \ \#count\{A : d3(A)\} = 0.$, the two difference sets represented by the presence of $d/1$ and $d2/1$ predicate atoms in a are ensured to be distinct, since a must contain at least one $d3/1$ predicate atom. The rule $not\ d2(B) \ :- \ r3(A, B),\ not\ d(A).$ accomplishes that whenever a does not contain $d(x_i^I)$, then it also cannot contain $d2(z_i^I)$ with x_i^I and z_i^I representing the same propositional atom $x_i \in X$. From this we obtain that the difference set represented by the presence of $d2/1$ predicate atoms in a is a proper subset of the difference set represented by the presence of $d/1$ predicate atoms since additionally both difference sets cannot be identical. Lastly, the minimize statement $\#minimize\{1, P : d2(P)\}.$ attempts to minimize the number of $d2/1$ predicate atoms in a. This ensures that the difference set represented by the presence of $d2/1$ predicate atoms is not only a proper subset of some other set in $D_S(\kappa, \mu)$, but also a minimal set of $D_S(\kappa, \mu)$, since there is no other difference set, that is a proper subset of some difference set and that contains fewer elements. □

Theorem 26. *For the set A of all answer sets to the ASP logic program $A_{\dot{+}_S}(\kappa, \mu)$, the following relation holds:*

$$interpretation(A, X^I, t/1) = Mod(\kappa \dot{+}_S \mu)$$

Proof. Let us first consider the case $min_S^p(\kappa, \mu) = \emptyset$. In this case we know that there exists no set in $D_S(\kappa, \mu)$, that is a proper subset of some set in $D_S(\kappa, \mu)$. Hence, no set in $D_S(\kappa, \mu)$ has a proper subset and therefore each set in $D_S(\kappa, \mu)$ is a minimal set. Looking at Definition 2.8 of Satoh's revision operator, it is evident, that in this case the models of the revision result are exactly the models of the revision formula μ, i.e. $Mod(\kappa \dot{+}_S \mu) = Mod(\mu)$. Since the integrity constraints $constraints(\mu, X^I)$ lead to $interpretation(A, X^I, t/1) = Mod(\mu)$ as described in Section 2.8.3 we can conclude that $interpretation(A, X^I, t/1) = Mod(\kappa \dot{+}_S \mu)$ for the case $min_S^p(\kappa, \mu) = \emptyset$.
Next, we consider the case $min_S^p(\kappa, \mu) = \{\emptyset\}$, i.e. the empty set is the only minimal set. The empty set is then a difference set in $D_S(\kappa, \mu)$ and therefore there exists at least one model in μ, that is also a model of κ. As a consequence, $d_{min}(\kappa, \mu) = 0$. Hence, we can define the encoding $A_{\dot{+}_S}(\kappa, \mu)$ as the encoding $A_{\dot{+}_D}(\kappa, \mu)$ with a minimum Dalal distance of 0. Due to Theorem 22 we then obtain $interpretation(A, X^I, t/1) = Mod(\kappa \dot{+}_S \mu)$ for the case $min_S^p(\kappa, \mu) = \{\emptyset\}$.
Finally, we address the remaining cases. Due to the choice rule the answer set solver is aware that any of the propositional atoms can potentially be *true*. As described in Section 2.8.3, the integrity constraints $constraints[\kappa, Y^I]$ and $constraints[\mu, X^I]$ thus lead to $interpretation(A, Y^I, t/1) = Mod(\kappa)$ and $interpretation(A, X^I, t/1) = Mod(\mu)$ (due to $X^I \cap Y^I = \emptyset$). Moreover, the encoding's only two rules ensure that whenever two integers of X^I and Y^I represent the same propositional atom, which is defined by the representation facts, and their corresponding $t/1$ predicate atoms differ in their truth assignments, the $d/1$ predicate atom, that takes the X^I integer as argument, is derived to be *true*. Further, in case that two integers of X^I and Y^I represent the same propositional atom and their corresponding $t/1$ predicate atoms have identical truth assignments, the predicate atom $d/1$, that takes the X^I integer as argument, cannot be derived and is thus *false* due to the concept of default negation. Accordingly, for a given answer set a, the presence of $d/1$ predicate atoms in a describes a difference set $d \in D_S(\kappa, \mu)$ in the following way: for every $d(x_i^I)$, that is contained in the answer set a, the corresponding propositional atom $x_i \in X$ is an element of d. The set $minSetConstraints[min_S^p(\kappa, \mu), X^I]$ of integrity constraints ensures that for every minimal set $m \in min_S^p(\kappa, \mu)$ no difference set represented by the presence of $d/1$ predicate atoms contains

more elements than m AND all elements of m at the same time. That way it is ensured that no difference set represented by the $d/1$ predicate atoms is a non-minimal set and thus all these sets are minimal sets. As a consequence, $interpretation(A, X^I, t/1)$ contains only those models of μ, for which a model of κ exists, such that $d_S(\kappa, \mu)$ is a minimal set of $D_S(\kappa, \mu)$, which is precisely the definition of Satoh's revision. From this we can conclude $interpretation(A, X^I, t/1) = Mod(\kappa \dot{+}_S \mu)$. □

Theorem 27. *For the optimal answer set a to the ASP logic program $A^O_{\dot{-}_S}(\kappa, \phi)$, the bit vector $interpretation(a, Z^I, d2/1)$ corresponds to the bit vector representation of a minimal set of the set of all difference sets $D_S(\kappa, \neg\phi)$, that is a proper subset of some set in $D_S(\kappa, \neg\phi)$.*

Proof. Encoding $A^O_{\dot{-}_S}(\kappa, \phi)$ is almost identical to encoding $A^O_{\dot{+}_S}(\kappa, \phi)$ of the corresponding revision theorem 25 except that $constraints[\mu, X^I]$ are replaced by $:- \ cConstraints[\phi, X^I]$. and $constraints[\mu, Z^I]$ replaced by $:- \ cConstraints[\phi, Z^I]$. As a result, $interpretation(A, X^I, t/1) = Mod(\neg\phi)$ and $interpretation(A, Z^I, t/1) = Mod(\neg\phi)$, with A denoting the set of all answer sets. The remaining proof is analogous to the proof of Theorem 25. □

Theorem 28. *For the set A of all answer sets to the ASP logic program $A_{\dot{-}_S}(\kappa, \phi)$, the following relation holds:*

$$interpretation(A, X^I, t/1) = Mod(\kappa \dot{-}_S \phi)$$

Proof. Let us first consider the case $min^p_S(\kappa, \neg\phi) = \emptyset$. In this case we know that there exists no set in $D_S(\kappa, \neg\phi)$, that is a proper subset of some set in $D_S(\kappa, \neg\phi)$. Hence, no set in $D_S(\kappa, \neg\phi)$ has a proper subset and therefore each set in $D_S(\kappa, \neg\phi)$ is a minimal set. Taking into account Definition 2.11 of Satoh's contraction operator, it is evident, that in this case the models of the contraction result are exactly the union of the models of the negated contraction formula ϕ and the models of κ, i.e. $Mod(\kappa \dot{-}_S \phi) = Mod(\kappa) \cup Mod(\neg\phi)$. As has been shown in the proof of Theorem 24, $interpretation(B, X^I, t/1) = Mod(\kappa) \cup Mod(\neg\phi)$ given the set B of all answer sets to an encoding e consisting of lines 1-7 of encoding $A_{\dot{+}_S}(\kappa, \phi)$. Since for the case $min^p_S(\kappa, \neg\phi) = \emptyset$ encoding $A_{\dot{-}_S}(\kappa, \phi)$ is defined to be identical to e, we obtain $interpretation(A, X^I, t/1) = Mod(\kappa \dot{-}_S \phi)$.
Next, we consider the case $min^p_S(\kappa, \neg\phi) = \{\emptyset\}$, i.e. the empty set is the only minimal set. The empty set is then a difference set in $D_S(\kappa, \neg\phi)$ and therefore there exists at least one model of $\neg\phi$, that is also a model of κ. As a consequence, $d_{min}(\kappa, \neg\phi) = 0$. Hence, we can define the encoding $A_{\dot{-}_S}(\kappa, \phi)$ as the encoding $A_{\dot{-}_D}(\kappa, \phi)$ with a minimum Dalal distance of 0. Due to Theorem 24 we then obtain $interpretation(A, X^I, t/1) = Mod(\kappa \dot{-}_S \phi)$ for the case $min^p_S(\kappa, \neg\phi) = \{\emptyset\}$.
Finally, for all other cases the encoding is almost identical to $A_{\dot{-}_D}(\kappa, \phi)$ with the exception that the last rule (line 8) of $A_{\dot{-}_D}(\kappa, \phi)$ is replaced by the set of rules $minSetRules[min^p_S(\kappa, \neg\phi), X^I]$. From the proof of Theorem 24 we know that $interpretation(B, X^I, t/1) = Mod(\kappa) \cup Mod(\neg\phi)$ given the set B of all answer sets to an encoding e consisting of lines 1-7 of encoding $A_{\dot{+}_S}(\kappa, \phi)$. Further, we know that for a given answer set a, the presence of $d/1$ predicate atoms in a describes a difference set $d \in D_S(\kappa, \neg\phi)$ in the following way: for every $d(x^I_i)$, that is contained in the answer set a, the corresponding propositional atom $x_i \in X$ is an element of d. In case there exists at least one predicate atom $d/1$ in the answer set the rules $minSetRules[min^p_S(\kappa, \neg\phi), X^I]$ ensure that for every minimal set $m \in min^p_S(\kappa, \neg\phi)$ no difference set represented by the presence of $d/1$ predicate atoms contains more elements than m AND all elements of m at the same time. That way it is ensured that no difference set represented by the $d/1$ predicate atoms is a non-minimal set and thus all these sets are minimal sets. As a consequence, $interpretation(A, X^I, t/1)$ contains only models of κ and those models of $\neg\phi$, for which a model of κ exists, such that $d_S(\kappa, \neg\phi)$ is a minimal set of $D_S(\kappa, \neg\phi)$, which is precisely the definition of Satoh's contraction. □

Theorem 29. *For the set A of answer sets to the inference check ASP encoding $A_I(B, \gamma)$ $A = \emptyset$ holds if and only if $B \models \gamma$.*

Proof. We address solely the case $B = (\kappa \dot{+}_D \mu)$ since the other three cases can be proven analogously. From Theorem 22 we know that $interpretation(A', X^I) = Mod(\kappa \dot{+}_D \mu)$ for the set A' of all answer sets to the encoding $A_{\dot{+}_D}(\kappa, \mu)$. Further, in Section 3.5.1 we have shown that adding the integrity constraint $:- \ cConstraints[\gamma, X^I]$ ensures that $Mod(\gamma) \cap interpretation(A', X^I) = \emptyset$, thus $interpretation(A', X^I) \subseteq Mod(\neg\gamma)$.

Let us first assume that encoding $A_I((\kappa \dot{+}_D \mu), \gamma)$ has no answer sets, i.e. $A = \emptyset$. As $cConstraints[\gamma, X^I]$ does not contain any other variables than X^I, the reason for it not having any answer sets must then be due to

$interpretation(A', X^I) \cap Mod(\neg\gamma) = \emptyset$, which can also be expressed by $Mod(\kappa \dot{+}_D \mu) \cap Mod(\neg\gamma) = \emptyset$, i.e. every model of B satisfies γ, hence $B \models \gamma$. Note that this situation can also arise if γ is a tautology.

Let us now assume that $B \models \gamma$, which means that there is no model of $\kappa \dot{+}_D \mu$, for which $\neg\gamma$ evaluates to true. Due to $interpretation(A, X^I) \subseteq Mod(\neg\gamma)$, which is represented by the integrity constraint : $-cConstraints[\gamma, X^I]$, encoding $A_I((\kappa \dot{+}_D \mu), \gamma)$ cannot have any answer sets, i.e. $A = \emptyset$. □

Theorem 30. *For the model check ASP encoding $A_M(B, N)$ the equation $A \neq \emptyset$, with A denoting the set of all answer sets to the encoding, holds if and only if $N \in Mod(B)$.*

Proof. We address solely the case $B = (\kappa \dot{+}_D \mu)$, since the other three cases can be proven analogously. Let us first assume that $A_M(B, N)$ has at least one answer set, i.e. $A \neq \emptyset$. From Theorem 22 we know that $interpretation(A', X^I) = Mod(\kappa \dot{+}_D \mu)$, with A' denoting the set of all answer sets to $A_{\dot{+}_D}(\kappa, \mu)$. Further, the facts $modelFacts[N, X^I]$ by definition ensure that $interpretation(A, X^I) = \{N\}$. From this we can conclude that $Mod(\kappa \dot{+}_D \mu) \cap \{N\} \neq \emptyset$ and thus $N \in Mod(\kappa \dot{+}_D \mu)$ and $N \in Mod(B)$.

Let us now assume that $N \in Mod(B)$, i.e. $N \in Mod(\kappa \dot{+}_D \mu)$ and $N \in interpretation(A', X^I)$. The second part of the encoding ($modelFacts[N, X^I]$) requires that for every answer set a, the statement $interpretation(a, X^I) = \{N\}$ holds. The fulfillment of this requirement by $N \in interpretation(A', X^I)$ implies that $A_M(B, N)$ has an answer set, thus $A \neq \emptyset$. □