Annals of Pure and Applied Logic 175 (2024) 103456

Contents lists available at ScienceDirect

Annals of Pure and Applied Logic

journal homepage: www.elsevier.com/locate/apal

Full Length Article

The ghosts of forgotten things: A study on size after forgetting

Paolo Liberatore

DIAG, Sapienza University of Rome, Italy

ARTICLE INFO

Article history:
Received 3 May 2022
Received in revised form 13 December 2023
Accepted 2 May 2024
Available online 8 May 2024

MSC:
03B42
68T27
68T30
03B05
03D15

Keywords:
Logical forgetting
Boolean minimization

ABSTRACT

Forgetting is removing variables from a logical formula while preserving the constraints on the other variables. In spite of reducing information, it does not always decrease the size of the formula and may sometimes increase it. This article discusses the implications of such an increase and analyzes the computational properties of the phenomenon. Given a propositional Horn formula, a set of variables and a maximum allowed size, deciding whether forgetting the variables from the formula can be expressed in that size is D^p-hard in Σ^p_2. The same problem for unrestricted CNF propositional formulae is D^p_2-hard in Σ^p_3.

1. Introduction

Several articles mention simplification as an advantage of forgetting, if not its motivation. Forgetting means deleting pieces of knowledge, and less is more. Less knowledge is easier to remember, easier to work with, easier to interpret. To cite a few:

- "With an ever growing stream of information, bounded memory and short response time suggest that not all information can be kept and treated in the same way. [...] forgetting [...] helps us to deal with information overload and to put a focus of attention" [21].
- "For example, in query answering, if one can determine what is relevant with respect to a query, then forgetting the irrelevant part of a knowledge base may yield more efficient query-answering" [17].
- "Moreover, forgetting may be applicable in summarizing a knowledge base by suppressing lesser details, or for reusing part of a knowledge base by removing an unneeded part of a larger knowledge base, or in clarifying relations between predicates" [16].

E-mail address: liberato@diag.uniroma1.it.

https://doi.org/10.1016/j.apal.2024.103456

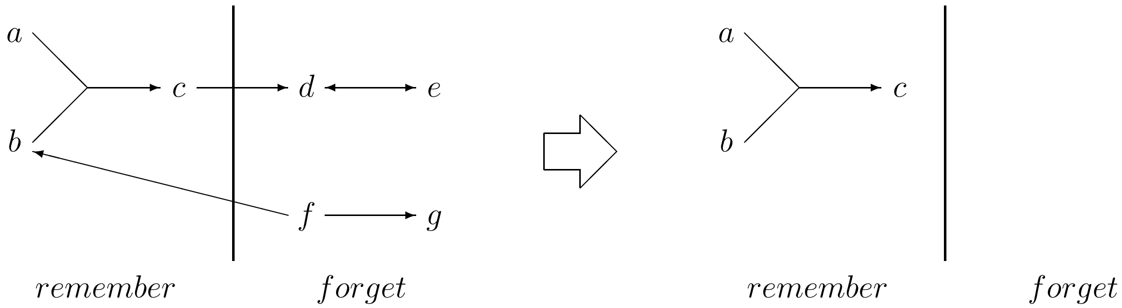


Fig. 1. An example of forgetting some variables. Arrows stand for propositional implications.

- "For performing reasoning tasks (planning, prediction, query answering, etc.) in an action domain, not all actions of that domain might be necessary. By instructing the reasoning system to forget about these unnecessary/irrelevant actions, without changing the causal relations among fluents, we might obtain solutions using less computational time/space" [23].
- "There are often scenarios of interest where we want to model the fact that certain information is discarded. In practice, for example, an agent may simply not have enough memory capacity to remember everything he has learned" [24].
- "The most immediate application of forgetting is to model agents with limited resources (e.g., robots), or agents that need to deal with vast knowledge bases (e.g., cloud computing), or more ambitiously, dealing with the problem of lifelong learning. In all such cases it is no longer reasonable to assume that all knowledge acquired over the operation of an agent can be retained indefinitely" [44].
- "For example, we have a knowledge base K and a query Q. It may be hard to determine if Q is true or false directly from K. However, if we discard or forget some part of K that is independent of Q, the querying task may become much easier" [52].
- "To some extent, all of these can be reduced to the problem of extracting relevant segments out of large ontologies for the purpose of effective management of ontologies so that the tractability for both humans and computers is enhanced. Such segments are not mere fragments of ontologies, but stand alone as ontologies in their own right. The intuition here is similar to views in databases: an existing ontology is tailored to a smaller ontology so that an optimal ontology is produced for specific applications" [20].

These authors are right: if forgetting simplifies the body of knowledge then it is good for reducing the amount of information to store, for increasing the efficiency of querying it, for clarifying the relationships between facts, for obtaining solutions more easily, for retaining by agents of limited memory, for tailoring knowledge to a specific application. If forgetting simplifies the body of knowledge, all these motivations are valid.

If.

What if not? What if forgetting does not simplify the body of knowledge? What if it complicates it? What if it enlarges instead of shrinking it?

This looks impossible. Forgetting is removing. Removing information, but still removing. Removing something leaves less, not more. What remains is less than what before, not more. Forgetting about d, e, f and g in the formula depicted on the left of Fig. 1 only leaves information about a, b and c.

The only information that remains is that a and b imply c. All the rest, like c implying d or f implying b is forgotten. What is left is smaller than what before because it is only a part of that.

This is the prototypical scenario of forgetting, the first that comes to mind when thinking about removing information: some information goes away, the rest remains. The rest is a part of the original. Smaller. Simpler. Easier than that to store, to query, to interpret. But prototypical does not mean exclusive.

Forgetting x from the formula on the left of Fig. 2 complicates it instead of simplifying it. Whenever a, b and c are the case so is x. And x implies n, m and l. Like the neck of an hourglass, x funnels the first three

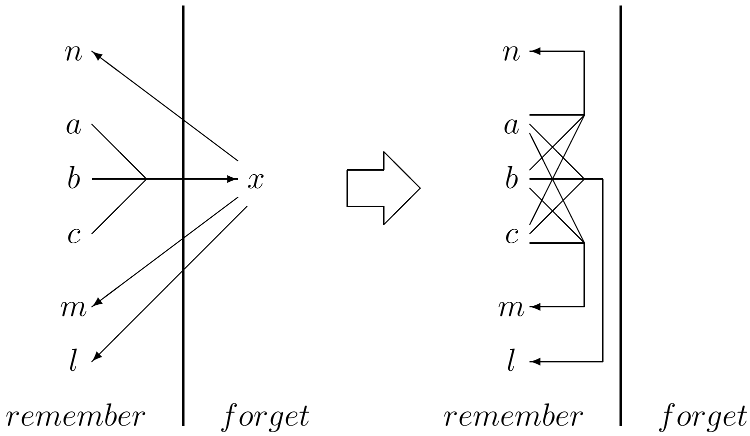


Fig. 2. A formula that is complicated instead of simplified by forgetting.

variables in the upper bulb to the last three in the lower. Without it, these links need to be spelled out one by one: a, b and c imply n; a, b and c imply m; a, b and c imply l. The variable x acts like a shorthand for the first three variables together. Removing it forces repeating them.

Forgetting x deletes x but not its connections with the other variables. The lines that go from a, b and c to n, l and m survive. Like a ghost, x is no longer there in its body, but in its spirit: its bonds. These remain, weaved where x was.

The formula resulting from forgetting is still quite short, but this is only because the example is designed to be simple for the sake of clarity. Cases with larger size increase due to forgetting are easy to find. Forgetting a single variable never increases size much, but forgetting many may increase size exponentially.

The size of the formula resulting from forgetting matters for all reasons cited by the authors above. To summarize, it is important for:

1. sheer memory needed;
2. the cost of reasoning; formulae that are difficult for modern solvers are typically large; while efficiency is not directly related to size, small formulae are usually easy to solve;
3. interpreting the information; the size of a formula tells something about how much the forgotten variables are related to the others.

These points are relevant to different research areas: for example, Delgrande and Wang [17] mention the second point regarding disjunctive logic programming; Erdem and Ferraris [23] do the same in the context of reasoning about actions. The third point is cited in the general survey on forgetting by Eiter and Kern-Isberner [21] and in the article that generalizes forgetting across different logics by Delgrande [16].

This witnesses that the problem of size after forgetting is relevant to different logics. Many of them generalize or can express propositional logic or Horn logic as a subcase. These are the two logics considered here, as greater common divisors of them.

The figures visualize forgetting as a cut between what is remembered and what is forgotten. This cut may divide parts that are easy to separate like in the first figure or parts that are not natural to separate like in the second figure. The first cut glides following the direction of the fabric of the knowledge base. The second is resisted by the connections it cuts.

The number of these connections does not tell the difference. How closely they hold together the parts across the cut does. Even if the links in the first example were $c \rightarrow d_1, \ldots, c \rightarrow d_{100}$ instead of $c \rightarrow d$, the result would be the same. What matters is not how many links connect the parts across the cut, but how they do.

An increase of size gauges the complexity of these connections. The first example is easy to cut because its implications are easy to ignore: c may imply d, but if d is forgotten this implication is removed and nothing else changes. The second example is not so easy to cut: forgetting x does not just remove its implication from a, b and c; it shifts its burden to the remaining variables.

A size increase suggests that the forgotten variables are closely connected to the remaining ones. If forgetting is aimed at subdividing knowledge, it would be like the chapter on Spain next to that of Samoa and far away from France in an atlas. The natural division is by continents, not initials of the name. In general, the natural divisions are by topics, so that things closely connected stay close to each other. Forgetting about Samoa when describing Spain is easier than forgetting about France. Neglecting some obscure diplomatic relations is more natural than neglecting a bordering country.

Forgetting may be abstracting [39]. Cold weather increases virus survival, which facilitates virus transmission, which causes flu. Forgetting about viruses: cold weather causes flu. But forgetting is not always natural as an abstraction. A low battery level, a bad UPS unit and a black-out cause a laptop not to start; which causes a report not to be completed, a movie not to be watched and a game not to be played. Forgetting about the laptop is a complication more than an abstraction: the three preconditions cause the first effect, they cause the second effect, and they cause the third effect. If x is the laptop not starting, this is the example in the second figure, where forgetting increases size. That cold weather causes cold is short, simple, a basic fact of life for most people. Brevity is the soul of abstraction.

In summary, a short formula is preferred for storage and computational reasons. The size of the formula after forgetting is important for epistemological reason, to evaluate how natural a partition or abstraction of knowledge is. Either way, the question is: how large is a formula after forgetting variables?

The question is not as obvious as it looks. Several formulae represent the same piece of knowledge. For example, $a \vee (\neg a \wedge b)$ is the same as the shorter $a \vee b$. The problem of formula size without forgetting eluded complexity researchers for twenty years: it was the prototypical problem for which the polynomial hierarchy was created in the seventies [47], but framing it exactly into one of these classes only succeeded at the end of the nineties [50]. This is the problem of whether a formula is equivalent to another of a given size.

The problem studied in this article is whether forgetting some variables from a formula is equivalent to a formula of given size.

Forgetting is not complicated. A simple recipe for forgetting x from F is: replace x with true in F, replace x with false in F, disjoin the two resulting formulae [32]. If F is in conjunctive normal form, another recipe is: replace all clauses containing x with the result of resolving them [53,16,45]. The first solution may not maintain the syntactic form of the formula. None of them is guaranteed to produce a minimal one.

Forgetting no variable from a formula results in the formula itself. Insisting on forgetting something does not change complexity: every formula F is the result of forgetting x from $F \wedge x$ if x is a variable not in F. The complexity of the size of F is a subcase of the size of forgetting x from F. It is however not an interesting subcase: the question is how much size decreases or increases due to forgetting. If F has size 100 before forgetting and 10 after, this looks like a decrease, but is not if F is equivalent to a formula of size 5 before forgetting and to none of size 9 or less afterwards. This is a size increase, not a decrease.

The main results of this article are the complexity characterization of this problem in the Horn and general propositional case where size is the total number of occurrences of literals. The problem is D^p-hard and belongs to Σ_2^p when the formula is Horn; it is D_2^p-hard and in Σ_3^p for arbitrary CNF formulae. A detailed plan of the article follows.

After Section 2 introduces some basic concepts like resolution, Section 3 formally defines forgetting and gives some results about size. Some equivalent formulations of forgetting are given, as well as some ways to compute forgetting.

Section 4 shows the complexity of the problem in the Horn restriction. It comes before the general case because of its slightly simpler proofs. The problem is D^p-hard and belongs to Σ_2^p.

Section 5 shows that the problem is D_2^p-hard and belongs to Σ_3^p for arbitrary CNF formulae. In both cases, hardness is more difficult to prove than membership; on the other hand, it extends to logics that include propositional or Horn logics as subcases. For example, since modal logics extend propositional logics, the problem of size of forgetting is D_2^p-hard.

Section 6 compares the results in this article with previous work. Section 7 discusses future directions of study.

A number of examples and counterexamples rely on calculating the resolution closure of a formula, its minimal equivalent formulae, the result of forgetting a variable from it and the minimal formulae equivalent to that. The program `minimize.py` does these operations on the formula it reads from another file, for example `allvariables.py` or `outresolve.py`. It is currently available at https://github.com/paololiberatore/minimize.py together with the files that contain the formulae mentioned in this article.

2. Preliminaries

2.1. Formulae

The formulae in this article are all propositional in conjunctive normal form (CNF): they are sets of clauses, a clause being the disjunction of some literals and a literal a propositional variable or its negation. This is not truly a restriction, as every formula can be turned into CNF without changing its semantics. A clause is sometimes identified with the set of literals it contains. For example, a subclause is a subset of a clause.

If l is a negative literal $\neg x$, its negation $\neg l$ is defined as x.

The variables a formula A contains are denoted $Var(A)$.

Definition 1. The size $||A||$ of a formula A is the number of variable occurrences it contains.

This is not the same as the cardinality of $Var(A)$ because a variable may occur multiple times in a formula. For example, $A = \{a, \neg a \vee b, a \vee \neg b\}$ has size five because it contains five literal occurrences even if its variables are only two. The size is obtained by removing from the formula all propositional operators, commas and parentheses and counting the number of symbols left.

The definition of size implies the definition of minimality: a formula is minimal if it is equivalent to no formula smaller than it. Given a formula, a minimal equivalent formula is a possibly different but equivalent formula that is minimal. As an example, $A = \{a, \neg a \vee b, a \vee \neg b\}$ has size five since it contains five literal occurrences; yet, it is equivalent to $B = \{a, b\}$, which only contains two literal occurrences. No formula equivalent to A or B is smaller than that: B is minimal. Minimizing a formula means obtaining a minimal equivalent formula. This problem has long been studied [40,47,11,50,31,10].

Definition 2. The clauses of a formula A that contain a literal l are denoted by $A \cap l = \{c \in A \mid l \in c\}$.

This notation is unambiguous: when is between two sets, the symbol $\cap$ denotes their intersection; when is between a set and a literal, it denotes the clauses of the set that contain the literal. This is like seeing $A \cap l$ as the shortening of $A \cap \text{clauses}(l)$, where $\text{clauses}(l)$ is the set of all possible clauses that contain the literal l.

2.2. Resolution

Resolution is a syntactic derivation mechanism that produces a clause that is a consequence of two clauses: $c_1 \vee l$ and $c_2 \vee \neg l$ generate the clause c_3 that results from removing repetitions from $c_1 \vee c_2$. Resolution is denoted $c_1 \vee l, c_2 \vee \neg l \vdash c_3$. Sometimes $\vdash_R$ is used in place of $\vdash$ to emphasize the use of resolution as the syntactic derivation rule. This is unnecessary in this article since no other derivation rule is ever mentioned.

Unless noted otherwise, tautologic clauses are excluded. Writing $c_1 \vee a, c_2 \vee \neg a \vdash c_3$ implicitly assumes that none of the three clauses is a tautology unless explicitly stated. Two clauses that would resolve in a tautology are considered not to resolve, which is not a limitation [38]. Tautologic clauses are forbidden in formulae, which is not a limitation either since tautologies are always satisfied. This assumption has normally little importance, but is crucial to superredundancy, a concept defined in the next section.

In what follows tautologies are excluded from formulae and from resolution derivations. As a result, resolving two clauses always generates a clause different from them.

A resolution proof $F \vdash G$ is a binary forest where the roots are the clauses of G, the leaves are the clauses of F and every parent is the result of resolving its two children.

Definition 3. The resolution closure of a formula F is the set $\mathrm{ResCn}(F) = \{c \mid F \vdash c\}$ of all clauses that result from applying resolution zero or more times from F.

The clauses of F are derivable by zero-step resolutions from F. Therefore, $F \vdash c$ and $c \in \mathrm{ResCn}(F)$ hold for every $c \in F$.

The resolution closure is similar to the deductive closure but not identical. For example, $a \vee b \vee c$ is in the deductive closure of $F = \{a \vee b\}$ but not in the resolution closure. It is a consequence of F but is not obtained by resolving clauses of F.

All clauses in the resolution closure $\mathrm{ResCn}(F)$ are in the deductive closure but not the other way around. The closures differ because resolution does not expand clauses: $a \vee b \vee c$ is not a resolution consequence of $a \vee b$. Adding expansion kills the difference [33,46].

$$F \models c \text{ if and only if } c' \in \mathrm{ResCn}(F) \text{ for some } c' \subseteq c$$

That resolution does not include expansion may suggest that it cannot generate any clause that strictly contains other entailed clauses. That would be too good to be true, since the shortest entailed clauses would be exactly the ones generated by resolution. In fact, it is not the case, as seen in the formula $\{a \vee b \vee c, a \vee b \vee e, \neg e \vee c \vee d\}$: the second and third clauses resolve to $a \vee c \vee b \vee d$, which contains the first clause of the formula, $a \vee b \vee c$.

What is the case is that resolution generates all prime implicates [33,46], the minimally entailed clauses. The relation between $\mathrm{ResCn}(F)$ and the deductive closure of F tells that if a clause is entailed, a subset of it is generated by resolution; since the only entailed subclause of a prime implicate is itself, it is the only one resolution may generate. Removing all clauses that contain others from $\mathrm{ResCn}(F)$ results in the set of the prime implicates of F.

While $\mathrm{ResCn}(F)$ contains all clauses generated by an arbitrary number of resolutions, some properties used in the following require the clauses obtained by a single resolution step.

Definition 4. The resolution of two formulae is the set of clauses obtained by resolving each clause of the first formula with each clause of the second:

$$\mathrm{resolve}(A, B) = \{c \mid c', c'' \vdash c \text{ where } c' \in A \text{ and } c'' \in B\}$$

If either of the two formulae comprises a single clause, the abbreviations $\mathrm{resolve}(A, c) = \mathrm{resolve}(A, \{c\})$, $\mathrm{resolve}(c, B) = \mathrm{resolve}(\{c\}, B)$ and $\mathrm{resolve}(c, c') = \mathrm{resolve}(\{c\}, \{c'\})$ are used.

This set contains only the clauses that results from resolving a single clause of A with a single clause of B. Exactly one resolution of one clause with one clause. Not zero, not multiple ones. A clause of A is not by itself in $\text{resolve}(A, B)$ unless it is also the resolvent of another clause of A with a clause of B.

2.3. Superredundancy

A clause of a formula is superredundant if it is redundant in the resolution closure of the formula [36]: $\text{ResCn}(F) \backslash \{c\} \models c$. The following properties of superredundancy and superirredundancy are used in this article.

Lemma 1 *([36], Lemma 2). If c is a superirredundant clause of F, it is contained in every minimal CNF formula equivalent to F.*

Lemma 2 *([36], Lemma 5). If a formula contains only superirredundant clauses, it is minimal.*

Lemma 3 *([36], Lemma 8). If no two clauses of F resolve, then a clause of F is superredundant if and only if F contains a clause that is a strict subset of it.*

Lemma 4 *([36] Lemma 11). If a clause c of F is superredundant, it is also superredundant in $F \cup \{c'\}$.*

Lemma 5 *([36] Lemma 12). A clause c of $F[\mathsf{true}/x]$ is superredundant if it is superredundant in F, it contains neither x nor $\neg x$ and F does not contain $c \vee \neg x$. The same holds for $F[\mathsf{false}/x]$ if F does not contain $c \vee x$.*

Superirredundancy differs from the related concepts of irredundancy [28,35], essentiality [29,7,9], and membership in all minimal formulae. This is shown by the clause a in the formula $F = \{a, \neg a \vee b, \neg b \vee a\}$: it is not superirredundant, but is irredundant, is an essential prime implicate and is in all minimal formulae that are equivalent to F.

- the resolution closure of F is $\{a, b, \neg a \vee b, \neg b \vee a\}$, where a is redundant; therefore, a is superredundant in F, not superirredundant;
- removing a from F results in $\{\neg a \vee b, \neg b \vee a\}$, which does not entail a; therefore, a is irredundant in F;
- the prime implicates of F are a and b; the only CNF formula equivalent to F comprising prime implicates is $\{a, b\}$, which contains a; therefore, a is an essential prime implicate of F;
- the prime implicates a and b of F do not resolve; therefore, the resolution closure of the set of prime implicates of F is $\{a, b\}$; neither two clauses of F nor two clauses of $\{a, b\}$ resolve in a; therefore, $\{a\}$ is both essential and prime essential for F, its set of prime implicates, the resolution closure of its set of prime implicates and its represented Boolean function;
- the only minimal-size formula equivalent to F is $\{a, b\}$; as a result, a belongs to all minimal-size formulae equivalent to F.

Essentiality and superirredundancy differ. They both prove membership to all minimal-size formulae equivalent to the formula, but superirredundancy follows from Lemma 3, Lemma 4 and Lemma 5. These lemmas shorten the formula by replacing certain variables with true or false until its resolution closure is easy to calculate in full.

3. Forgetting

A Boolean function over a set of n variables is a mapping $\{0, 1\}^n \rightarrow \{0, 1\}$. Forgetting some variables results in a Boolean function over the remaining variables.

Definition 5. Forgetting all variables but Y from a Boolean function over variables X is the Boolean function g over variables Y such that $g(M) = 1$ where M is a model over Y if and only if there exists a model M' over $X \backslash Y$ such that $f(M \cup M') = 1$.

Forgetting is typically applied to formulae [32] rather than functions. A common definition is: the formula over the remaining variables that entails the same consequences over the remaining variables. Being based on the semantical concept of entailment, this definition is unaffected by the syntax of the formula. All equivalent formulae are the same when forgetting. The result of forgetting is the same as its equivalent formulae. The semantical definition solves the ambiguity since equivalent formulae represent the same Boolean function.

The definition of forgetting on formulae follows. A formula represents a Boolean function. Variables are forgotten from the Boolean function. The resulting Boolean function is represented by another formula. This other formula is the result of forgetting. Actually, every formula representing the same Boolean function is the result of forgetting. Every such formula expresses forgetting.

Definition 6. A formula B expresses forgetting all variables except Y from a formula A over variables X if forgetting all variables but Y from the Boolean function represented by A results in the Boolean function represented by B.

The definition sets a constraint over B rather than uniquely defining a specific formula. Every formula B fits it as long as it is built over the right variables and represents the same Boolean function.

Syntax is irrelevant to this definition. As it should: every B' that is syntactically different but equivalent to B carries the same information. There is no reason to confer $A[\mathsf{true}/x] \vee A[\mathsf{false}/x]$ a special status among all formulae holding the same information. Every formula equivalent to it is an equally valid result of forgetting.

The definition captures this parity among formulae by not defining forgetting as a single specific formula and then delegating the identification of its alternatives to equivalence. If B expresses forgetting some variables from A and B' is equivalent to B and contains the same variables, then B' also expresses forgetting.

The common definition of forgetting based on entailment becomes a consequence: A and B entail the same formulae over the variables Y. This is proved in steps. The first is that A and B are equisatisfiable with the same sets of literals that mentions exactly the variables Y. This result is also used in the hardness proofs.

Theorem 1. *A formula B over the variables Y expresses forgetting all variables except Y from A if and only if $S \cup A$ is equisatisfiable with $S \cup B$ for all sets of literals S over variables Y that mention all variables in Y.*

The condition based on equisatisfiability extends from sets of literals to arbitrary formulae.

Theorem 2. *A formula B over the variables Y expresses forgetting all variables from A except Y if and only if $A \wedge D$ is equisatisfiable with $B \wedge D$ for every formula D over variables Y.*

The usual definition of forgetting in terms of consequences turns into a theorem.

Theorem 3. *A formula B over variables Y expresses forgetting all variables from A except Y if and only if $B \models C$ is the same as $A \models C$ for all formulae C such that $Var(C) \subseteq Y$.*

The condition that S mentions all variables of Y can be dropped from the equisatisfiability of $S \cup A$ and $S \cup B$.

Theorem 4. *A formula B over the variables Y expresses forgetting all variables except Y from A if and only if $S \cup A$ is equisatisfiable with $S \cup B$ for all sets of literals S over variables Y.*

The following Section 3.1 discusses the main focus of the analysis of this article: the size of a formula when forgetting variables; the following Section 3.2 shows how to actually compute forgetting in general and in two specific cases; finally, Section 3.3 proves that in some cases, certain literals are always in the result of forgetting, which is important when computing the size after forgetting.

3.1. Size of forgetting

Many formulae B express forgetting the same variables X from a formula A. Some may be large and some may be small. Producing an artificially large formula is straightforward: if $\{a \vee b, b \vee c\}$ expresses forgetting, also $\{a \vee b, b \vee c, \neg a \vee a, a \vee b \vee \neg c\}$ does: adding tautologies and consequences does not change the semantics of a formula. The question is not whether a large expression of forgetting exists.

The question is whether a small expression of forgetting exists. In this context, "small" means "of polynomial size". Technically: given a formula A and a set of variables X, does any formula of size polynomial in that of A express forgetting X from A?

Forgetting each variable x from the CNF formula A is expressed by Boole elimination [6] $A[\mathsf{true}/x] \vee A[\mathsf{false}/x]$, which can be converted back into a CNF of quadratic size. Forgetting many variables this way produces an exponentially large formula. Yet, this formula may be equivalent to a short one.

This is not the case for all formulae [32]. Yet, it is the case for some formulae. It depends on the formula. For example, forgetting variables from negation-free CNF formulae amounts to removing the clauses that contain these variables. The question is the existence of a small formula expressing forgetting from a specific formula. This will be proved D^p-hard and in Σ_2^p for Horn formulae by the following Theorem 6 and D_2^p-hard and in Σ_3^p for unrestricted CNF formulae by the following Theorem 7.

3.2. How to forget

Three properties related to computing forgetting are proved: it can be performed one variable at time, it can be performed by resolution, and it may be performed on the independent parts of the formula, if any.

Lemma 6 *([16]). If B expresses forgetting the variables Y from A and C expresses forgetting the variables Z from B, then C expresses forgetting $Y \cup Z$ from A.*

Forgetting can be done by resolution with the Davis-Putnam elimination method [15,18,53,16]. The function $\text{resolve}(A, B)$ provided by Definition 4 gives the clauses obtained by resolving each clause of A with each clause of B, if they resolve. The notation $A \cap l$ introduced in Definition 2 gives the clauses of A that contain the literal l.

Theorem 5 *([53, Theorem 6], [16, Theorem 6]). The formula $A \backslash (A \cap x) \backslash (A \cap \neg x) \cup \mathit{resolve}(A \cap x, A \cap \neg x)$ expresses forgetting x from A.*

Forgetting a single variable is not a limitation because Lemma 6 tells that forgetting a set of variables can be performed one variable at time: forgetting x first and $Y \backslash \{x\}$ then is the same as forgetting Y.

The problem is that forgetting this way may produce non-minimal formulae even from minimal ones. For example, $A = \{a \vee b \vee x, \neg x \vee c, a \vee c\}$ is minimal, but resolving x out to forget it produces $\{a \vee b \vee c, a \vee c\}$, which is not minimal since the first clause is entailed by the second. The proof that A is minimal is long and tedious, and is therefore omitted. The formulae in the `outresolve.py` file of `minimize.py` show similar examples where the formula obtained by resolving out a variable either contains a redundant literal or is irredundant although not minimal.

Since resolving Horn clauses produces Horn clauses, this theorem indirectly shows that forgetting variables from Horn formulae is expressed by a Horn formula [18]. That formula may not be minimal, yet its minimal equivalent formulae cannot be non-Horn: as mentioned in Section 2.2, resolution derives all clauses of all minimal equivalent formulae.

When a formula comprises two independent parts with no shared variable, forgetting from the formula is the same as forgetting from the two parts separately. This property is used in the following hardness proofs that merge two polynomial-time reductions.

Lemma 7 *([13,32]). Let A and B be two formulae built over disjoint alphabets: $Var(A) \cap Var(B) = \emptyset$. A formula C expresses forgetting the variables Y from A and D expresses forgetting the variables Y from B if and only if $C \cup D$ expresses forgetting the variables Y from $A \cup B$.*

3.3. Necessary literals

Finding a minimal version of a formula is difficult [40,49,26]. Finding a minimal formula expressing forgetting is further complicated by the addition of forgetting. Determining the exact complexity of this problem proved difficult; not so much for membership to classes in the polynomial hierarchy but for hardness. Fortunately, proving hardness does not require finding the minimal size of arbitrary formulae, just for the formulae that are targets of the reduction. NP-hardness is for example proved by translating a formula (to be checked for satisfiability) to another formula and a set of variables (where the variables have to be forgotten from the formula). Such a reduction does not generate all possible formulae. Only for the ones generated by the reduction, the minimal size after forgetting is necessary.

This is good news, because reductions do not generate all possible formulae. Rather the opposite: they usually produce formulae of a very specific form. Still better, a reduction can be altered to simplify computing the minimal size of the formulae it produces. If the minimal size is difficult to assess for the formulae produced by a reduction, the reduction itself can be changed to simplify them.

The reductions used in this article rely on two tricks to allow for simple proofs. The first is that some clauses of the formulae they generate are in all minimal-size equivalent formulae; this part of the minimal size is therefore always the same. The second is that the rest of the minimal size depends on the presence or absence of certain literals in all formulae expressing forgetting; where these literals occur if present does not matter, only whether they are present or not.

The first trick is based on superredundancy [36], defined in Section 2.

The second requires proving that a literal is contained in all formulae that express forgetting. This is preliminarily proved when no forgetting is involved.

Lemma 8. *If S is a set of literals such that $S \cup A$ is consistent, but $S \backslash \{l\} \cup \{\neg l\} \cup A$ is not, the CNF formula A contains a clause that contains l.*

Since consistency with S and with $S \backslash \{l\} \cup \{\neg l\}$ is unaffected by syntactic changes, they are the same for all formulae equivalent to A. In other words, if the conditions of the lemma hold for A they also hold for every formula equivalent to A.

This property carries over to formulae expressing forgetting by constraining S to only contain variables not to be forgotten.

Lemma 9. *If $S \cup \{l\}$ is a set of literals over the variables Y such that $S \cup A$ is consistent, but $S \backslash \{l\} \cup \{\neg l\} \cup A$ is not, every CNF formula that expresses forgetting all variables except Y from A contains a clause that contains l.*

How is this lemma used? To prove that reductions from a problem to the problem of minimal size of forgetting work. Not all reductions can be proved correct this way. The ones used in this article are built to allow that. They generate a formula that contains a clause that contains a certain literal l that may or may not meet the condition of the lemma. Depending on this, l may or may not be necessary after forgetting. This is a $+0$ or a $+1$ in the size of the minimal formulae expressing forgetting. If the other literals occurrences are k, the minimal size is $k + 0$ or $k + 1$ depending on whether the conditions of Lemma 9 are met.

In order for this to work, the $+0/+1$ separation is not enough. Equally important to the $k + 0$ vs. $k + 1$ size is that the other addend k stays the same. This is the number of the other literal occurrences. The formulae produced by the reduction may or may not contain a literal l, but this is useless if the rest of the formula changes. For example, if k changes from 10 to 9 the total size is either $10 + 0$ or $9 + 1$, which are the same. Lemma 9 concerns the presence of l in a formula, but this tells its overall size only when the rest of the formula has a fixed form. This is ensured by superirredundancy [36], defined in Section 2.

4. Size after forgetting, Horn case

How much forgetting variables increases or decreases size? Given a formula A and a set of variables Y, how much space forgetting Y from A takes? Technically, how large is a formula expressing forgetting Y from A? A complexity analysis of a decision problem requires turning it into a yes/no question. Given k, A and Y, does a formula B of size bounded by k express forgetting?

This is a decision problem: each of its instances comprises a number k, a formula A and a set of variables Y; the solution is yes or no. Yet, it may not always capture the question of interest. For example, A may be a formula of size 100 that can be reduced to size 20 by forgetting the variables Y. This looks like a good result: the resulting formula takes much less space to be stored, checking what can be inferred from it is usually easier, and its literals are probably related in some simple way. Yet, all of this may be illusory: formula A has size 100, but only because it is extremely redundant; it could be reduced to size 10 just by rearrangements, without forgetting anything. That forgetting can be expressed in size 20 no longer looks good. It is not even a size decrease, it is a size doubling.

If forgetting was required independent on size, and checking size is a side question, the problem still makes sense: is forgetting A expressed by a formula of size 20? If forgetting is done for size reasons, or for reasons that depend on size, the problem is not this but rather "does forgetting reduce size?" or "how much forgetting increases or decreases size?" These questions depend on the original size of the formula. The answer is not "20". It is rather "forgetting increases size from 10 to 20". It is certainly not "forgetting decreases size from 100 to 20", since the formula can be shrunk more without forgetting.

The solution is to disallow formulae of size 100 that can be reduced to 10 without forgetting. A formula of size 100 really has size 100. It is not the inflated version of a formula of size 10. This way, if size can be reduced from 100 to 20 when forgetting, this reduction is only due to forgetting, not to the original formula being larger than necessary.

The following lemmas and theorems include this assumption that the formula is minimal in size. For example, the problem of checking the size after forgetting is proved hard for the complexity class D^p when the formula is minimal. It is also proved to be in the class Σ_2^p. The proof of the latter also holds when the formula is not minimal: it holds in both cases.

Checking whether forgetting Y from A can be expressed in space k is easy to be proved in Σ_2^p if k is unary or polynomially bounded by the size of A: all it takes is checking all formulae of size k for their equisatisfiability with all sets of literals S over the remaining variables by Theorem 4. Hardness is not so easy to prove, and in fact leaves a gap to membership: it is only proved D^p-hard in this article.

Not that D^p-hardness is easy to prove. It requires two long lemmas, one for a coNP-hardness reduction and one for an NP-hardness reduction. These reductions have a form that allows them to be merged into a single D^p-hardness reduction.

A generic NP-hardness reduction is "if F is satisfiable then forgetting takes space less than or equal to k and greater otherwise". It may not be merged. An additional property is required: forgetting can never be expressed in size less than k. If this is also a property of a coNP-hardness reduction where the size bound is l, the overall size is always $k+l$ or greater, with $k+l$ being only possible when the first formula is satisfiable and the second unsatisfiable.

This explains why the lemmas are formulated with "equal to k" in one case and "greater than k" in the other. Their other peculiarity, that the formula generated by the reduction is required to be minimal, is due to the reasons explained above.

Lemma 10. *There exists a polynomial algorithm that turns a CNF formula F into a minimal-size Horn formula A, a subset $X_C \subseteq Var(A)$ and a number k such that forgetting all variables except X_C from A is expressed by a Horn formula of size k if F is unsatisfiable and only by Horn formulae of size greater than or equal to $k+2$ if F is satisfiable.*

This lemma shows a polynomial reduction from propositional unsatisfiability to the problem of forget size in the Horn case. As for all polynomial reductions, it translates a formula F without knowing its satisfiability, which however affects the minimal size of expressing forgetting.

Being a polynomial reduction from propositional unsatisfiability, it proves the forgetting size problem coNP-hard. Yet, the lemma is not formulated this way. It instead predicates about the reduction itself. Only this way it could include the additional property that the minimal size is either k or at least $k+2$. This allows merging it with another reduction to form a proof of D^p-hardness.

The following lemma also shows the problem NP-hard: a formula F is satisfiable if and only if forgetting some variables from A can be expressed in a certain space. However, its statement refers to the reduction itself for the same reason of the previous lemma: merging with the previous reduction into a D^p-hardness reduction.

Lemma 11. *There exists a polynomial algorithm that turns a CNF formula F into a minimal-size Horn formula A, a subset $X_C \subseteq Var(A)$ and a number k such that forgetting all variables except X_C from A is expressed by a Horn formula of size k if F is satisfiable and only by Horn formulae of size greater than k otherwise.*

The problem of size after forgetting is the target of both a reduction from propositional satisfiability and from propositional unsatisfiability. This alone proves it both NP-hard and coNP-hard. These reductions have the additional property that forgetting variables from the formulae they generate cannot be expressed in size less than k. This allows merging them into a single D^p-hardness proof.

Lemma 12. *Checking whether forgetting some variables from a minimal-size Horn formula is expressed by a CNF or Horn formula bounded by a certain size is D^p-hard.*

Proving hardness takes most of this section, but still leaves a gap between the complexity lower bound it shows and the upper bound in the next theorem. The problem is D^p-hard, which is just a bit above NP-hardness and coNP-hardness, but belongs to a class of the next level of the polynomial hierarchy: Σ_2^p.

Theorem 6. *Checking whether forgetting some variables from a Horn formula is expressed by a CNF or Horn formula bounded by a certain size expressed in unary is D^p-hard and in Σ_2^p, and remains hard even if the formula is restricted to be of minimal size.*

The assumption that the size bound is represented in unary is technical. When formulated as a decision problem, the size of forgetting is the question whether forgetting certain variables X from a formula A is expressed by a formula of size k, but the actual problem is to find such a formula. If k is exponential in the size of A, a formula of size k may very well exist, but is unpractical to represent. Unless A is very small. The requirement that k is in unary forces the input of the problem to be as large as the expected output. If the available space is enough for storing a resulting formula of size k, it is also enough for storing an input string of length k, which k in unary is. In the other way around, representing k in unary witnesses the ability of storing a resulting formula of size k. The similar assumption "k is polynomial in the size of A" fails to include the case where A is very small but the space available for expressing forgetting is large.

5. Size after forgetting, general case

The complexity analysis for general CNF formulae mimics that of the Horn case. Two reductions prove the problem hard for the two basic classes of a level of the polynomial hierarchy. They are merged into a single proof that slightly increases the lower bound. A membership proof for a class of the next level ends the analysis.

The difference is that the level of the polynomial hierarchy is the second instead of the first. The two reductions prove the problem hard for Π_2^p and Σ_2^p. They are merged into a D_2^p-hardness proof. Finally, the problem is located within Σ_3^p.

As for the Horn case, the first lemma proves the problem Π_2^p-hard, but is formulated in terms of the reduction because the reduction is needed to raise the lower bound to D_2^p-hard.

Lemma 13. *There exists a polynomial algorithm that turns a CNF formula F into a minimal-size CNF formula A, a subset $X_C \subseteq Var(A)$ and a number k such that forgetting all variables from A except X_C is expressed by a CNF formula of size k if $\forall X \exists Y.F$ is valid and only by CNF formulae of size $k+2$ or greater otherwise.*

The second lemma is again about a reduction. Its statement implies that the problem is Σ_2^p-hard, but it predicates about the reduction rather than the hardness. This allows it to be merged with the first lemma into a proof of D_2^p-hardness. The existing proof of Σ_2^p-hardness of the problem without forgetting [50] also proves the problem with forgetting Σ_2^p-hard, but does not allow such a merging and does not hold in the restriction of minimal formulae.

Lemma 14. *There exists a polynomial algorithm that turns a DNF formula $F = f_1 \vee \cdots \vee f_m$ over variables $X \cup Y$ into a minimal-size CNF formula A, a subset $X_C \subseteq Var(A)$ and a number k such that forgetting all variables except X_C from A is expressed by a CNF formula of size k if $\exists X \forall Y.F$ is valid, and only by larger CNF formulae otherwise.*

As anticipated, the two reductions merge into one that proves the problem of forgetting size D_2^p-hard.

Lemma 15. *Checking whether forgetting a given set of variables from a minimal-size CNF formula is expressed by a CNF formula bounded by a certain size is D_2^p-hard.*

The next theorem adds a complexity class membership to the hardness of the problem of size of forgetting proved in the previous lemma.

Theorem 7. *Checking whether forgetting some variables from a CNF formula is expressed by a CNF formula of a certain size expressed in unary is D_2^p-hard and in Σ_3^p, and remains hard even if the CNF formula is restricted to be of minimal size.*

The technical assumption that the size bound is expressed in unary is the same as in Theorem 6; it is motivated after that theorem.

6. Related work

Reducing the size of a propositional formula when no forgetting is involved has been much studied [47,3,11,50,12,7,8,31,30]. The practical problem of synthesizing a minimal formula representing a Boolean function is formalized by the decision problem of deciding whether a propositional formula has an equivalent form of a given size or less. This problem depends on the definition of size. Size may be the total number of occurrences of literals in a formula [3,50,8,31,30], but may also be its clause count [3,7]. It may be the total size of the bodies of the clauses [3] or a cost function obeying some constraints [11,12]. The problem changes depending on the definition of size. While complexity may look the same, it requires specific proofs. The ones for the number of occurrences of literals do not in general work for the number of clauses or the other measures in the literature.

The existing literature provides mechanisms for forgetting variables from a formula and overall bounds on the minimal size of expressing forgetting for all formulae. They leave open the question in between: the minimal size of expressing forgetting for a specific formula. An example result of the first kind is: "Salient features of the solution provided include linear time complexity, and linear size of the output of (iterated) forgetting" [1]: forgetting generates linear formulae. An example result of the second kind is: "the size of the result of forgetting may be exponentially large in the size of the input program" [16]: forgetting may produce exponentially large formulae. It may, not must. For some formulae, forgetting may not increase size. The only previous complexity result about the size of forgetting for a specific formula is discussed later [57]. Other authors reported worst-case results [23,22], and some the opposite, as certain forgetting mechanisms of certain logics can be expressed in polynomial size [27]. These results are usually unaffected by whether size is defined as total length or number of clauses.

Forgetting propositional variables is also called variable elimination, especially in the context of automated reasoning [19]: it is a way to simplify a formula before processing. As such, it has stricter efficiency requirements than general forgetting. For example, the NiVER preprocessor "resolves away a variable only if there will be no increase in space" [48]. A quadratic increase would be too much, given the aim of reducing the overall runtime of automated reasoning.

Forgetting is often identified by its dual concept of uniform interpolation, especially in first-order, modal and description logics [5]. While forgetting is always expressible in exponential space in propositional logics, uniform interpolants in other logics may be larger, if they exist at all. For example, their size is at least triple-exponential in certain description logics, provided that they exist [42]. Analogous to the question of checking their size is checking their existence [2].

A way to forget in propositional logic is Boole elimination [6], but the resulting formula $A[\mathsf{true}/x] \vee A[\mathsf{false}/x]$ does not maintain the syntactic form of A: a CNF like $a \wedge (x \vee b \vee c)$ becomes the non-CNF $((a) \vee (a \wedge (b \vee c))$. While this formula can be turned into CNF, directly combining clauses is more convenient when working on CNFs.

An alternative is given by the Davis-Putnam elimination method [15], as proved by Delgrande and Wassermann [18] in the Horn case and extended to the general case by Wang [53] and Delgrande [16]. Theorem 5 shows that forgetting is expressed by $A \backslash (A \cap x) \backslash (A \cap \neg x) \cup \text{resolve}(A \cap x, A \cap \neg x)$, where $\text{resolve}(A, B)$ gives the clauses obtained by resolving each clause of A with each clause of B if they resolve and $A \cap l$ gives the clauses of A that contain the literal l.

Forgetting this way may produce non-minimal formulae even from minimal ones. For example, $A = \{a \vee b \vee x, \neg x \vee c, a \vee c\}$ is minimal, but resolving x out to forget it produces $\{a \vee b \vee c, a \vee c\}$, which is not minimal since the first clause is entailed by the second. The proof that A is minimal is long and tedious, and is therefore omitted. The formulae in the `outresolve.py` file of `minimize.py` show similar examples where the formula obtained by resolving out a variable either contains a redundant literal or is irredundant although not minimal.

Lemma 11 incidentally proves the problem of forgetting size NP-hard. An alternative is given by Theorem 2 by Zhou [57]. The formulae generated by the reduction in its proof are not Horn, but can be straightforwardly turned so by switching the sign of some variables. However, they are not enforced to be of size k or more as required by the D^p-hardness proof. While the Horn restriction is easy to satisfy, this constraint does not look so.

Generating only minimal formulae is instead unattainable. The reduction in the proof of Theorem 2 by Zhou [57] hinges on non-minimal formulae. It works by translating non-minimal formulae into non-minimal formulae. The original formula is added three variables so that it is recovered by forgetting them. This is the hearth of the reduction: the size after forgetting follows the size before the translation; this proves the problem of size after forgetting as hard as the problem of size without forgetting, which is NP-hard [3,29] in the Horn case. The addition of the three variables links the size of the formulae. Of all three formulae: the original, the result of the translation and the result of forgetting. The reduction is an NP-hardness proof because checking whether the original formula is equivalent to one of a certain size or less is NP-hard. The translated formula before forgetting can only be made minimal by making the original minimal, which trivializes the check.

While the reduction proves the problem NP-hard, proving NP-hardness is not the final aim. It is the D^p-hardness when the formula is minimal. The D^p-hardness proof requires the bound on size. Converting non-minimal formulae into non-minimal formulae turns the problem from size reduction by forgetting to size reduction without forgetting. This also applies to the case of arbitrary formulae, proved Σ_2^p-hard by reduction from the problem of formula minimization [50].

7. Conclusions

Forgetting variables from formulae may increase size, instead of decreasing it. This phenomenon is already recognized as a problem [17,4]. Deciding whether it takes place or not for a specific formula and variables to forget is difficult. While checking inference is polynomial in the Horn case, checking whether forgetting is expressed by a formula of a certain size is at least D^p-hard, which implies it both NP-hard and coNP-hard; the same for the general case, where inference is coNP-complete but checking size after forgetting is at least D_2^p-hard.

The precise characterization of complexity is an open problem. For Horn formulae, Theorem 6 leaves a gap between the lower bound of D^p-hardness and the upper bound of Σ_2^p-membership. According to what proved so far, the problem could be as easy as D^p-complete or as hard as Σ_2^p-complete. Nothing in the results obtained so far favors either possibility. Actually, nothing indicates for certain that the problem is complete for either class; it could be complete for any class in between, like $\Delta_2^p[\log n]$ or Δ_2^p.

Anecdotal evidence hints that the problem is Σ_2^p-complete. The analogous problems without forgetting for unrestricted formulae kept a gap between NP and Σ_2^p for twenty years before being closed as Σ_2^p-complete [47,50]. Proving membership was easy; proving hardness was not.

This is a common pattern, not limited to formula minimization: in many cases, hardness is more difficult to prove than membership. Not always, but hardness proofs are often more complicated than membership proofs. The hardness and membership lemmas in this article are an example: several pages of proof for hardness, ten lines for membership. A proof of Σ_2^p-hardness may very well exist but is just difficult to find. As it was for the problem without forgetting.

All of this is anecdotal. Technically, the complexity of the problem could be anything in between D^p and Σ_2^p.

As a personal opinion, not based on the technical results, the author of this article would bet on the problem being Σ_2^p-complete. The missing proof of Σ_2^p-hardness could be an extension of that of Lemma 11, since both Σ_2^p and NP are based on an existential quantification. The extension of an already difficult proof would be further complicated by the addition of an inner universal quantification.

A way to partly close the issue is to further restrict the Horn case to simplify the problem to D^p. The gap would close to its lower end for such a class of formulae.

This is the first direction for further studies in how forgetting affects size. Another is the investigation in subcases other than the general propositional case and its Horn restriction. Many are relevant. Forgetting is very easy on formulae in DNNF [14], as it amounts to simply removing literals. It is also easy for the Krom restriction [53] as resolving binary clauses always produces binary clauses, which are at most quadratically many. It may not on other tractable cases in Post's lattice [43].

Forgetting has variants and is defined for many logics other than propositional logic. The problem of size applies to all of them. What is its complexity? This article characterizes it for one version of forgetting in propositional logic. The other versions and the other logics are still open. Some results may apply to them as well. Logic programs embed Horn clauses; the hardness results for the Horn case may hold for them as well. More generally, how hard it is to check whether forgetting in logic programs is expressed within a certain size? How hard it is in first-order logic? In description logics? How hard it is when forgetting literals rather than variables?

The size after forgetting matters not only when forgetting variables, but also literals [32], possibly with varying variables [41]. All variants inhibit the values of some variables to matter: forgetting variables makes their values irrelevant to the satisfaction of the formula; forgetting literals makes only the true or false value not to matter; varying variables allows some other variables to change. These variants generalize forgetting variables, inheriting the problem of size with the same complexity at least.

Forgetting applies to frameworks other than propositional logic. The problem of size applies to them as well.

Forgetting from logic programs [52,54,27] is usually backed by the need of solving conflicts rather than an explicit need of reducing size. Yet, an increase in size is recognized as a problem: "Whereas relying on such methods to produce a concrete program is important in the sense of being a proof that such a program exists, it suffers from several severe drawbacks when used in practice: In general, it produces programs with a very large number of rules" [4]; "It can also be observed that forgetting an atom results in at worst a quadratic blowup in the size of the program. [...] While this may seem comparatively modest, it implies that forgetting a set of atoms may result in an exponential blowup" [17].

Another common area of application of forgetting is first-order logic [37]. Size after forgetting is related to bounded forgetting [58], which is forgetting with a constraint on the number of nested quantifiers. The difference is that the bound is an additional constraint rather than a limit to check. Bounded forgetting still involves a measure (the number of quantifiers), but forcing the result by that measure makes it close to bounding PSPACE problems [34]. Enforcing size rather than checking it is another possible direction of expansion of the present article.

As are the other logics where forgetting is applied like description logics [20,55] and modal logics [56,51], where forgetting is often referred to as its dual concept of uniform interpolant, and also temporal logics [25], logics for reasoning about actions [23,44], and defeasible logics [1].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Appendix A. Proofs

Theorem 1. *A formula B over the variables Y expresses forgetting all variables except Y from A if and only if $S \cup A$ is equisatisfiable with $S \cup B$ for all sets of literals S over variables Y that mention all variables in Y.*

Proof. The Boolean functions A, B and S represent are respectively denoted $a()$, $b()$ and $s()$. The variables of A are $X \cup Y$, where X is to forget and Y is not. Every set of literals S that mentions exactly the variables of Y has a single model over Y, denoted M_S.

The satisfiability of $S \cup A$ is the existence of a model over variables X that satisfies both S and A. Alternatively, it is the existence of two models M and M' respectively over Y and $X \backslash Y$ such that $s(M \cup M') = 1$ and $a(M \cup M') = 1$. Since S does not mention any variable in $X \backslash Y$, the former subcondition $s(M \cup M') = 1$ equates to $s(M) = 1$. This is also equivalent to $M = M_S$ since S is only satisfied by M_S. As a result, the satisfiability of $S \cup A$ simplifies to the existence of a model M and a model M' such that $M = M_S$ and $a(M \cup M') = 1$. This is the same as the existence of a model M' such that $a(M_S \cup M') = 1$.

For the same reasons, the satisfiability of $S \cup B$ equates to the existence of a model M' such that $b(M_S \cup M') = 1$. Since B only mentions the variables Y, this condition simplifies to the existence of a model M' such that $b(M_S) = 1$. The subcondition $b(M_S) = 1$ does not mention M', negating the need of quantifying over this model. The satisfiability of $S \cup B$ eventually turns into just $b(M_S) = 1$.

The equisatisfiability of $S \cup A$ and $S \cup B$ is the same as: $b(M_S) = 1$ holds if and only if there exists a model M' such that $a(M_S \cup M') = 1$ for every set of literals S that mentions all variables in Y. The universal quantification over S is the same as a universal quantification of a model M_S over variables Y. The result is: for every model M_S over variables Y, $b(M_S) = 1$ holds if and only if there exists a model M' over variables $X \backslash Y$ such that $a(M_S \cup M') = 1$". This is the definition of B expressing forgetting $X \backslash Y$ from A. □

Theorem 2. *A formula B over the variables Y expresses forgetting all variables from A except Y if and only if $A \wedge D$ is equisatisfiable with $B \wedge D$ for every formula D over variables Y.*

Proof. Theorem 1 reformulates forgetting as the equisatisfiability of A and B with every set of literals S that mentions exactly the variables Y. This is a special case of the equisatisfiability of A and B with every formula D over Y. This proves one part of the theorem, from equisatisfiability to forgetting.

The other part is from forgetting to equisatisfiability: if B expresses forgetting then $A \wedge D$ and $B \wedge D$ are equisatisfiable for every formula D over Y.

Every formula D over Y is equivalent to the disjunction of some sets of literals, each mentioning exactly the variables Y. Formally, D is equivalent to $S_1 \vee \cdots \vee S_m$.

As a result, $A \wedge D$ is equivalent to $A \wedge (S_1 \vee \cdots \vee S_m)$, which is equivalent to $(A \wedge S_1) \vee \cdots \vee (A \wedge S_m)$. Theorem 1 tells that each $A \wedge S_i$ is equisatisfiable with $B \wedge S_i$. As a result, $(A \wedge S_1) \vee \cdots \vee (A \wedge S_m)$ is equisatisfiable with $(B \wedge S_1) \vee \cdots \vee (B \wedge S_m)$, which is equivalent to $B \wedge D$.

This chain of equivalences proves the equisatisfiability of $A \wedge D$ and $B \wedge D$. □

Theorem 3. *A formula B over variables Y expresses forgetting all variables from A except Y if and only if $B \models C$ is the same as $A \models C$ for all formulae C such that $Var(C) \subseteq Y$.*

Proof. The entailment $B \models C$ is the same as the unsatisfiability of $B \cup \neg C$. The same for $A \models C$. The equivalence of $A \models C$ and $B \models C$ coincides with the equisatisfiability of $B \cup \neg C$ and $A \cup \neg C$. By Theorem 2, this is the same as B expressing forgetting from A. □

Theorem 4. *A formula B over the variables Y expresses forgetting all variables except Y from A if and only if $S \cup A$ is equisatisfiable with $S \cup B$ for all sets of literals S over variables Y.*

Proof. The equisatisfiability in this theorem is a special case of that in Theorem 2, which is proved to hold if B expresses forgetting.

The equisatisfiability in Theorem 1 is a special case of the one in this theorem, which therefore implies that B expresses forgetting. □

Lemma 8. *If S is a set of literals such that $S \cup A$ is consistent, but $S \backslash \{l\} \cup \{\neg l\} \cup A$ is not, the CNF formula A contains a clause that contains l.*

Proof. Since $S \cup A$ is consistent, it has a model M.

The claim is that A contains a clause that contains l. This is proved by contradiction, assuming that no clause of A contains l. By construction $S \backslash \{l\}$ does not contain l either. As a result, $A' = S \backslash \{l\} \cup A$ does not contain l. It is still satisfied by M because M satisfies its superset $S \cup A$. Let M' be the model that sets l to false and all other variables the same as M. Let $l_1 \vee \cdots \vee l_m$ be an arbitrary clause of A'. Since M satisfies A, it satisfies at least one of these literals l_i. Since A does not contain l, this literal l_i is either $\neg l$ or a literal over a different variable. In the first case M' satisfies $l_i = \neg l$ because it sets l to false; in the second because it sets l_i the same as M, which satisfies l_i. This happens for all clauses of A', proving that M' satisfies A'.

Since M' also satisfies $\neg l$ because it sets l to false, it satisfies $A' \cup \{\neg l\} = S \backslash \{l\} \cup \{\neg l\} \cup A$, contrary to its assumed unsatisfiability. □

Lemma 9. *If $S \cup \{l\}$ is a set of literals over the variables Y such that $S \cup A$ is consistent, but $S \backslash \{l\} \cup \{\neg l\} \cup A$ is not, every CNF formula that expresses forgetting all variables except Y from A contains a clause that contains l.*

Proof. Let B be a formula expressing forgetting all variables from A but Y. By Theorem 4, since S is a set of literals over Y, the consistency of $S \cup A$ equates that of $S \cup B$. The same holds for $S \backslash \{l\} \cup \{\neg l\}$ since its variables are all in Y.

The lemma assumes the consistency of $S \cup A$ and the inconsistency of $S \backslash \{l\} \cup \{\neg l\} \cup A$. They imply the consistency of $S \cup B$ and the inconsistency of $S \backslash \{l\} \cup \{\neg l\} \cup B$. These two conditions imply that B contains a clause that contains l by Lemma 8. □

Lemma 10. *There exists a polynomial algorithm that turns a CNF formula F into a minimal-size Horn formula A, a subset $X_C \subseteq Var(A)$ and a number k such that forgetting all variables except X_C from A is expressed by a Horn formula of size k if F is unsatisfiable and only by Horn formulae of size greater than or equal to $k + 2$ if F is satisfiable.*

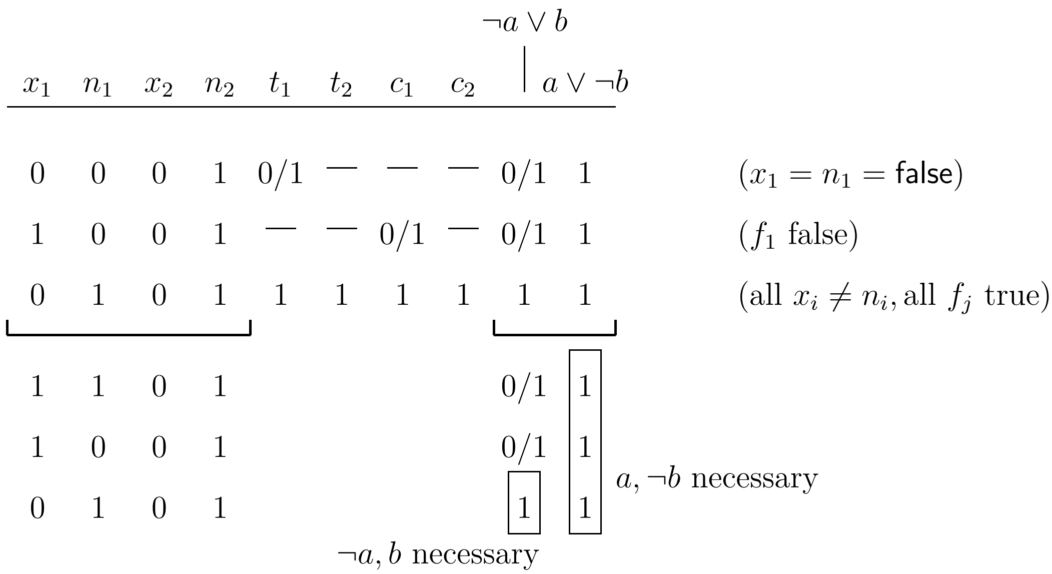
Proof. Let $F = \{f_1, \ldots, f_m\}$ be a CNF formula built over the alphabet $X = \{x_1, \ldots, x_n\}$. The reduction employs the fresh variables $E = \{e_1, \ldots, e_n\}$, $T = \{t_1, \ldots, t_n\}$, $C = \{c_1, \ldots, c_m\}$ and $\{a, b\}$. The formula A, the set of variables X_C and the number k are:

$$\begin{aligned} A = \; & \{\neg x_i \vee \neg e_i, \neg x_i \vee t_i, \neg e_i \vee t_i \mid x_i \in X\} \cup \\ & \{\neg x_i \vee c_j \mid x_i \in f_j,\ f_j \in F\} \cup \{\neg e_i \vee c_j \mid \neg x_i \in f_j,\ f_j \in F\} \cup \\ & \{\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee \neg a \vee b\} \cup \\ & \{a \vee \neg b\} \end{aligned}$$

$$X_C = X \cup E \cup \{a, b\}$$
$$k = 2 \times n + 2$$

Before formally proving the claim, how the reduction works is summarized. Some literals are still necessary after forgetting, and some of them are necessary only if F is satisfiable. The clauses $\neg x_i \vee \neg e_i$ make $\neg x_i$ and $\neg e_i$ necessary. The clause $a \vee \neg b$ makes a and $\neg b$ necessary. If F is always false, then for every value of the variables $X \cup E$ either some t_i can be set to false (if $x_i = e_i = \mathsf{false}$) or some c_j can be set to false (because e_i is the negation of x_i, and at least a clause of F is false). This makes the clause

$\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee \neg a \vee b$ satisfied regardless of a and b. Instead, if the formula is satisfied by an evaluation of X and E is its opposite, then all c_j and t_i have to be true, turning $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee \neg a \vee b$ into $\neg a \vee b$. This makes $\neg a$ and b necessary as well.



The figure shows three models as an example. In the first model, the assignments $x_1 = e_1 = \mathsf{false}$ allow t_1 to take any value (denoted 0/1); regardless of the other variables (irrelevant values are marked $-$), $t_1 = \mathsf{false}$ satisfies the clause $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee \neg a \vee b$ without the need to also satisfy its subset $\neg a \vee b$; this subclause can be false and still A is true. In the second model the values of x_i and e_i are opposite to each other for every i, but the clause $f_2 \in F$ is false; c_1 can take any value, including false; this again allows A to be true even if $\neg a \vee b$ is false. In the third model, the variables x_i and e_i are all opposite to each other and all clauses of F true; all t_i and c_i are forced to be true, making $\neg a \vee b$ the only way to satisfy the clause $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee \neg a \vee b$. When removing the intermediate variables t_i and c_i, all that matters is whether $\neg a \vee b$ was allowed to be false for some values of the removed variables or not. This is the case for the first two models but not the third, where $\neg a$ and b are necessary.

Minimality. The minimality of A is proved applying Lemma 5 to remove some clauses so that the remaining ones do not resolve and Lemma 3 applies. Lemma 2 proves A minimal since it only contains superirredundant clauses.

Substituting the variables a, b with false removes $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee \neg a \vee b$ and $a \vee \neg b$ from A. The remaining clauses contain x_i, e_i only negative and t_i, c_j only positive. Therefore, these clauses do not resolve. Since they do not contain each other, Lemma 3 proves them superirredundant. They are also superirredundant in A by Lemma 5 since A does not contain any of their supersets.

The superirredundancy of the remaining two clauses is proved by substituting all x_i, e_i with false. This substitution removes the clauses $\neg x_i \vee \neg e_i$, $\neg x_i \vee t_i$, $\neg e_i \vee t_i$, $\neg x_i \vee c_j$ and $\neg e_i \vee c_j$ because they all contain either $\neg x_i$ or $\neg e_i$. The two remaining clauses are $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee \neg a \vee b$ and $a \vee \neg b$. They have opposite literals, but resolving them results in tautologies. As a result, $F = \text{ResCn}(F)$. Since none of the two entails the other, they are irredundant in $\text{ResCn}(F)$ and are therefore superirredundant. By Lemma 5, they are superirredundant in A as well since A does not contain a superset of them.

Formula F is unsatisfiable. Forgetting all variables except X_C from A is expressed by $B = \{\neg x_i \vee \neg e_i \mid x_i \in X\} \cup \{a \vee \neg b\}$, a Horn formula of the required variables $X_C = X \cup E \cup \{a, b\}$ and size $||B|| = 2 \times n + 2 = k$.

Theorem 1 proves that B expresses forgetting if every set of literals S that contains exactly all variables $X_C = X \cup E \cup \{a, b\}$ is satisfiable with A if and only if it is satisfiable with B. Two cases are possible.

$\{x_i, e_i\} \subseteq S$ **for some** i ; the clause $\neg x_i \vee \neg e_i$ in both A and B is falsified by S; both $A \cup S$ and $B \cup S$ are unsatisfiable;

$\{x_i, e_i\} \subseteq S$ **for no** i ; since S contains either x_i or $\neg x_i$ for each i and either e_i or $\neg e_i$ for each i, either $\neg x_i \in S$ or $\neg e_i \in S$; as a result, all clauses $\neg x_i \vee \neg e_i$ are satisfied in $A \cup S$ and $B \cup S$, and can therefore be disregarded from this point on; the only remaining clause of B is $a \vee \neg b$;

if S contains $\neg a$ and b, then B is not satisfied; but A contains the same clause $a \vee \neg b$, so it is not satisfied either; if S contains both a and b or both $\neg a$ and $\neg b$, then B is satisfied, and A is also satisfied by setting all variables t_i and c_j to true; therefore, the only sets S that may differ when added to A and B are those containing a and $\neg b$; these sets are consistent with B; they make the clause $a \vee \neg b$ of A redundant, and resolve with the clause $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee \neg a \vee b$ making it subsumed by $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m$.

Two subcases are considered:

$\{\neg x_i, \neg e_i\} \subseteq S$ **for some** i the remaining clauses of A are satisfied by setting t_i to false, all t_z with $z \neq i$ to true and all c_j to true; in particular, the clause $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m$ is satisfied because of $t_i = \text{false}$;

$\{\neg x_i, \neg e_i\} \subseteq S$ **for no** i ; at this point, also $\{x_i, e_i\} \subseteq S$ for no i; as a result, S contains either $\{x_i, \neg e_i\}$ or $\{\neg x_i, e_i\}$, which means that it implies $x_i \not\equiv e_i$; the clauses $\neg e_i \vee c_j$ are therefore equivalent to $x_i \vee c_j$; by assumption, at least a clause of F is false for every possible value of the variables X; let f_j be such a clause for the only truth evaluation on X that satisfies S; by setting all variables t_i and all c_z with $z \neq j$ to true and c_j to false, this assignment satisfies all clauses; in particular, the clauses $\neg x_i \vee c_j$ and $x_i \vee c_j$ are satisfied even if c_j is false because f_j is false in S, which implies that all literals x_i and $\neg x_i$ it contains are false; the clause $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m$ is satisfied because of $c_j = \text{false}$.

All of this proves that B is the result of forgetting all variables except X_C from A.

Minimal number of literals. Every CNF formula B that expresses forgetting all variables except $X_C = X \cup E \cup \{a, b\}$ from A contains at least $k = 2 \times n + 2$ literal occurrences regardless of the satisfiability of F.

This is proved by showing that B contains the literals $\neg x_i$, $\neg e_i$, a and $\neg b$. This is in turn proved by Lemma 9: for each of them l, a set S is shown consistent with A while $S \backslash \{l\} \cup \{\neg l\}$ is not.

For the literals $\neg x_i$ and a the set S contains all $\neg x_i$, all e_i, a and b. It is consistent with A because both are satisfied by the model that sets all x_i to false and all e_i, t_i, c_i, a and b to true. Replacing $\neg x_i$ with x_i makes S inconsistent with $\neg x_i \vee \neg e_i$. Replacing a with $\neg a$ makes S inconsistent with $a \vee \neg b$.

For the literals $\neg e_i$ and $\neg b$, the set S contains all x_i, all $\neg e_i$, $\neg a$ and $\neg b$. It is consistent with A because both are satisfied by the model that assigns all x_i, t_i and c_i to true and all e_i, a and b to false. Replacing $\neg e_i$ with e_i makes S inconsistent with $\neg x_i \vee \neg e_i$. Replacing $\neg b$ with b makes it inconsistent with $a \vee \neg b$.

This proves that every formula obtained by forgetting all variables except X_C from A contains all the $k = 2 \times n + 2$ literals $\neg x_i$, $\neg e_i$, a and $\neg b$.

Formula F is satisfiable. If this is the case, every CNF formula B that expresses forgetting all variables except X_C from A contains the literals $\neg a$ and b. These two literals are in addition to the k literals of the previous point, raising the minimal number of literals to $k + 2$.

That B contains $\neg a$ is proved by exhibiting a set of literals S that is consistent with A while $S\backslash\{l\} \cup \{\neg l\}$ is not where $l = \neg a$. This implies that B contains $\neg a$ by Lemma 9. A similar set with $l = b$ shows that B also contains b.

Let M be a model of F. The set of literals S contains x_i or $\neg x_i$ depending on whether M satisfies x_i; it contains e_i or $\neg e_i$ depending on whether M falsifies x_i; it also contains $\neg a$ and $\neg b$. This set is consistent with A because they are both satisfied by the model that extends M by setting each e_i opposite to x_i, all t_i and c_i to true and a and b to false. In particular, the clause $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee \neg a \vee b$ is satisfied because it contains $\neg a$.

Replacing $\neg a$ with a makes S no longer consistent with A. Let $S' = S\backslash\{\neg a\} \cup \{\neg\neg a\}$. This set has the same literals over x_i and e_i of S. Since M satisfies F, for each of its clauses f_j at least a literal in f_j is true in M. If this literal is x_i, then S' contains x_i; since x_i is in f_j, formula A contains the clause $\neg x_i \vee c_j$; therefore, $S' \cup A \models c_j$. If the literal of f_j that is true in M is $\neg x_i$, then S' contains e_i; since $\neg x_i$ is in f_j, formula A contains $\neg e_i \vee c_j$; therefore, $S' \cup A \models c_j$. This proves that regardless of whether the literal of f_j that is true in M is positive or negative, if F is consistent then $S' \cup A$ implies c_j. This is the case for every j since M satisfies all clauses of F. Since S' contains either x_i or e_i for every i and A contains both $\neg x_j \vee t_j$ and $\neg e_j \vee t_j$ for every j, $S' \cup A$ also implies all variables t_j. Since $S' = S\backslash\{\neg a\} \cup \{a\}$ also contains a and $\neg b$, it is inconsistent with $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee \neg a \vee b$. This proves that $\neg a$ is in B.

The similar set S that contains a and b leads to the same point where all variables c_j and t_i are implied, making $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee \neg a \vee b$ consistent with $\{a, b\}$ but not with $\{a, \neg b\}$. This proves that b is also in B. $\square$

Lemma 11. *There exists a polynomial algorithm that turns a CNF formula F into a minimal-size Horn formula A, a subset $X_C \subseteq Var(A)$ and a number k such that forgetting all variables except X_C from A is expressed by a Horn formula of size k if F is satisfiable and only by Horn formulae of size greater than k otherwise.*

Proof. Let the formula be $F = \{f_1, \ldots, f_m\}$ and $X = \{x_1, \ldots, x_n\}$ its variables. The formula A is built over an extended alphabet comprising the variables $X = \{x_1, \ldots, x_n\}$ and the additional variables $O = \{o_1, \ldots, o_n\}$, $E = \{e_1, \ldots, e_n\}$, $P = \{p_1, \ldots, p_n\}$, $T = \{t_1, \ldots, t_n\}$, $C = \{c_1, \ldots, c_m\}$, $R = \{r_1, \ldots, r_n\}$, $S = \{s_1, \ldots, s_n\}$ and q.

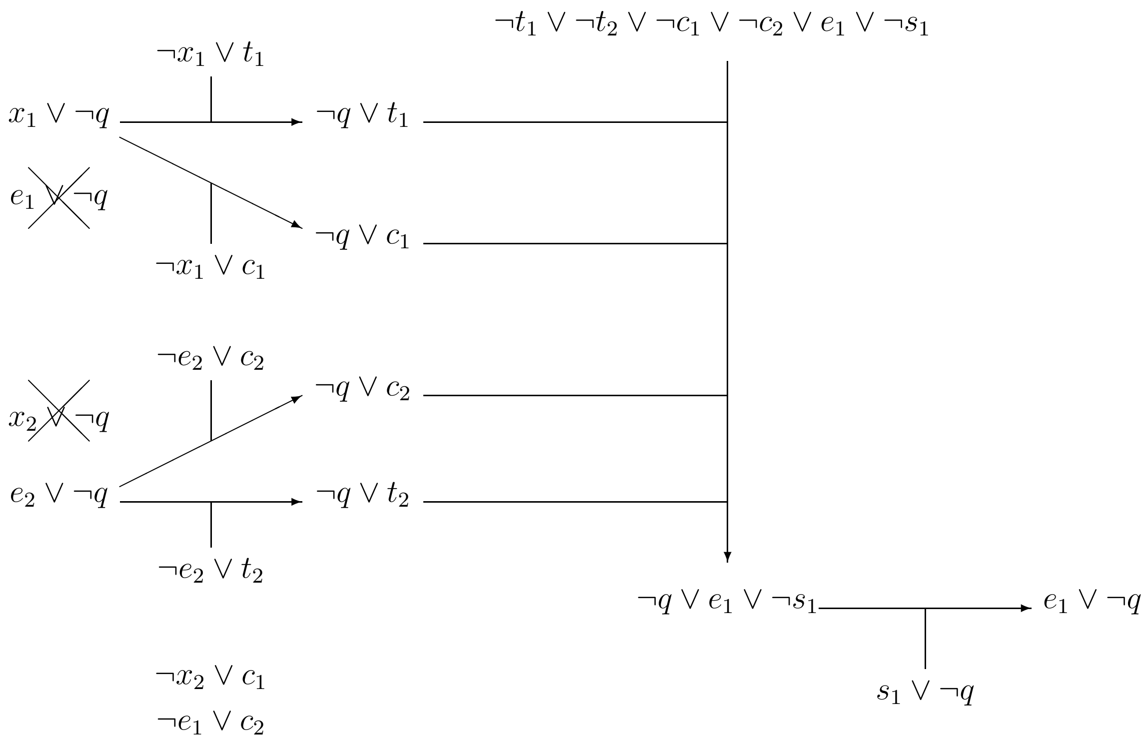
The formula A, the set of variables X_C and the integer k are as follows.

$$
\begin{aligned}
A &= A_F \cup A_T \cup A_C \cup A_B \\
A_F &= \{x_i \vee \neg o_i, o_i \vee \neg q \mid x_i \in X\} \cup \{e_i \vee \neg p_i, p_i \vee \neg q \mid x_i \in X\} \\
A_T &= \{\neg x_i \vee t_i, \neg e_i \vee t_i \mid x_i \in X\} \\
A_C &= \{\neg x_i \vee c_j \mid x_i \in f_j,\ f_j \in F\} \cup \{\neg e_i \vee c_j \mid \neg x_i \in f_j,\ f_j \in F\} \\
A_B &= \{\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee x_i \vee \neg r_i, r_i \vee \neg q \mid x_i \in X\} \cup \\
&\quad\ \{\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee e_i \vee \neg s_i, s_i \vee \neg q \mid x_i \in X\}
\end{aligned}
$$

$$X_C = X \cup E \cup T \cup C \cup R \cup S \cup \{q\}$$
$$k = 2 \times n + ||A_T|| + ||A_C|| + ||A_B||$$

Before formally proving that the reduction works, a short summary of why it works is given. The variables to forget are $O \cup P$. A way to forget them is to turn A_F into $A_R = \{x_i \vee \neg q, e_i \vee \neg q \mid x_i \in X\}$. The other clauses of A are superirredundant: Lemma 1 proves they belong to all minimal equivalent formulae. The bound k allows only one clause of A_R for each i. Combined with the clauses of A_T they entail $t_i \vee \neg q$. If F is satisfiable, they also combine with the clauses A_C to imply all clauses $c_j \vee \neg q$. Resolving these clauses with A_B produces all clauses $x_i \vee \neg q$ and $e_i \vee \neg q$, including the ones not in the formula. This way, a formula that contains one clause of A_R for each index i implies all of A_R, but only if F is satisfiable.

The following figure shows how $e_1 \vee \neg q$ is derived from $x_1 \vee \neg q$ and $e_2 \vee \neg q$, when the formula is $F = \{f_1, f_2\}$ where $f_1 = x_1 \vee x_2$ and $f_2 = \neg x_1 \vee \neg x_2$. These clauses translate into $A_C = \{\neg x_1 \vee c_1, \neg x_2 \vee c_1, \neg e_1 \vee c_2, \neg e_2 \vee c_2\}$. The rest of the formula A does not depend on the specific clauses of F but only on the number of variables and clauses it contains.



For each index i, at least one among $x_i \vee \neg q$ and $e_i \vee \neg q$ is necessary for deriving $\neg q \vee t_i$, which is required for these derivations to work. Alternatively, $\neg q \vee t_i$ may be selected. Either way, for each index i at least a two-literal clause is necessary.

The claim is formally proved in four steps: first, a non-minimal way to forget all variables except X_C is shown; second, its superirredundant clauses are identified; third, an equivalent formula of size k is built if F is satisfiable; fourth, the necessary clauses in every equivalent formula are identified; fifth, if F is unsatisfiable every equivalent formula is proved to have size greater than k.

Effect of forgetting.

Theorem 5 proves that forgetting all variables not in X_C, which are $O \cup P$, is expressed by resolving out these variables. Since o_i occurs only in $x_i \vee \neg o_i$ and $o_i \vee \neg q$, the result is $x_i \vee \neg q$. The same holds for p_i. The resulting clauses are denoted A_R:

$$A_R = \{x_i \vee \neg q, e_i \vee \neg q \mid x_i \in X\}$$

Superirredundancy.

The claim requires A to be minimal, which follows from all its clauses being superirredundant by Lemma 2. Most of them survive forgetting; the reduction is based on these being superirredundant. Instead of proving superirredundancy in two different but similar formulae, it is proved in their union.

In particular, the clauses $A_F \cup A_T \cup A_C \cup A_B$ are shown superirredundant in $A_F \cup A_R \cup A_T \cup A_C \cup A_B$. Lemma 4 implies that they are also superirredundant in its subsets $A_F \cup A_T \cup A_C \cup A_B$ and $A_R \cup A_T \cup A_C \cup A_B$, the formula before and after forgetting.

To be precise, the latter is just one among the formulae expressing forgetting. Yet, its superirredundant clauses are in all minimal CNF formulae equivalent to it as proved by Lemma 1. Therefore, all minimal CNF formulae expressing forgetting contain them.

Superirredundancy is proved via Lemma 5: a substitution simplifies $A_F \cup A_R \cup A_T \cup A_C \cup A_B$ enough to prove superirredundancy easily, for example because its clauses do not resolve and Lemma 3 applies.

- Replacing all variables x_i, e_i, t_i and c_j with true removes from $A_F \cup A_R \cup A_T \cup A_C \cup A_B$ all clauses in $A_R \cup A_T \cup A_C$, all clauses of A_F but $o_i \vee \neg q$ and $p_i \vee \neg q$ and all clauses of A_B but $r_i \vee \neg q$ and $s_i \vee \neg q$. The remaining clauses contain only the literals o_i, p_i, r_i, s_i and $\neg q$. Therefore, they do not resolve. Since none is contained in another, they are all superirredundant by Lemma 3. This proves the superirredundancy of all clauses $o_i \vee \neg q$, $p_i \vee \neg q$, $r_i \vee \neg q$ and $s_i \vee \neg q$.
- Replacing all variables q, o_i, p_i, r_i and s_i with false removes from $A_F \cup A_R \cup A_T \cup A_C \cup A_B$ all clauses but $A_T \cup A_C$. These clauses contain only the literals $\neg x_i$, $\neg e_i$, t_i and c_j. Therefore, they do not resolve. Since they are not contained in each other, Lemma 3 proves them superirredundant.
- Replacing all variables q, r_i and s_i with false and all variables t_i and c_i with true removes from $A_F \cup A_R \cup A_T \cup A_C \cup A_B$ all clauses but $x_i \vee \neg o_i$ and $e_i \vee \neg p_i$. They do not resolve because they do not share variables. Lemma 3 proves them superirredundant because they do not contain each other.
- Replacing all variables with false except for all variables t_i and c_j and the two variables x_h and r_h removes all clauses from $A_F \cup A_R \cup A_T \cup A_C \cup A_B$ but $\neg x_h \vee t_h$, $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee x_h \vee \neg r_h$ and all clauses $\neg x_h \vee c_j$ with $x_h \in f_j$. They only resolve in tautologies. Therefore, their resolution closure only contains them. Removing $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee x_h \vee \neg r_h$ from the resolution closure leaves only $\neg x_h \vee t_h$ and all clauses $\neg x_h \vee c_j$ with $x_h \in f_j$. They do not resolve since they do not contain opposite literals. Since $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee x_h \vee \neg r_h$ is not contained in them, it is not entailed by them. This proves it superirredundant. A similar replacement proves the superirredundancy of each $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee e_h \vee \neg s_h$.

These points prove that the clauses $A_F \cup A_T \cup A_C \cup A_B$ are superirredundant in the formula before forgetting and the clauses $A_T \cup A_C \cup A_B$ are superirredundant in the formula after forgetting. The only clauses that may be superredundant are A_R in the formula after forgetting.

Formula F is satisfiable.

Let M be a model satisfying F. Forgetting all variables except X_C is expressed by $A'_R \cup A_T \cup A_C \cup A_B$, where A'_R comprises the clauses $x_i \vee \neg q$ such that $M \models x_i$ and the clauses $e_i \vee \neg q$ such that $M \models \neg x_i$. This Horn formula has size k. It expresses forgetting because it is equivalent to $A_R \cup A_T \cup A_C \cup A_B$. This is proved by showing that it entails every clause in A_R.

Since M satisfies every clause $f_j \in F$, it satisfies at least a literal of f_j: for some x_i, either $x_i \in f_j$ and $M \models x_i$ or $\neg x_i \in f_j$ and $M \models \neg x_i$. By construction, $x_i \in f_j$ implies $\neg x_i \vee c_j \in A_C$ and $\neg x_i \in f_j$ implies $\neg e_i \vee c_j \in A_C$. Again by construction, $M \models x_i$ implies $x_i \vee \neg q \in A'_R$ and $M \models \neg x_i$ implies $e_i \vee \neg q \in A'_R$.

As a result, either $x_i \vee \neg q \in A'_R$ and $\neg x_i \vee c_j \in A_C$ or $e_i \vee \neg q \in A'_R$ and $\neg e_i \vee c_j \in A_C$. In both cases, the two clauses resolve in $c_j \vee \neg q$.

Since M satisfies either x_i or $\neg x_i$, either $x_i \vee \neg q \in A'_R$ or $e_i \vee \neg q \in A'_R$. The first clause resolves with $\neg x_i \vee t_i$ and the second with $\neg e_i \vee t_i$. The result is $t_i \vee \neg q$ in both cases.

Resolving all these clauses $t_i \vee \neg q$ and $c_j \vee \neg q$ with $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee x_i \vee \neg r_i$ and then with $r_i \vee \neg q$, the result is $x_i \vee \neg q$. In the same way, resolving these clauses with $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee e_i \vee \neg s_i$ and $s_i \vee \neg q$ produces $e_i \vee \neg q$. This proves that all clauses of A_R are entailed.

Necessary clauses

All CNF formulae that are equivalent to $A_R \cup A_T \cup A_C \cup A_B$ and have minimal size contain $A_T \cup A_C \cup A_B$ because these clauses are superirredundant, according to Lemma 1. Therefore, these formulae are $A_N \cup A_T \cup A_C \cup A_B$ for some set of clauses A_N. This set A_N is now proved to contain either $x_h \vee \neg q$, $x_h \vee \neg r_h$, $e_h \vee \neg q$, $e_h \vee \neg s_h$ or $t_h \vee \neg q$ for each index h. Let M and M' be the following models.

$$M = \{x_i = e_i = t_i = \mathsf{true} \mid i \neq h\} \cup \{x_h = e_h = t_h = \mathsf{false}\} \cup$$
$$\{c_j = \mathsf{true}\} \cup \{q = \mathsf{true}\} \cup \{r_i = \mathsf{true}, s_i = \mathsf{true}\}$$
$$M' = \{x_i = e_i = t_i = \mathsf{true} \mid i \neq h\} \cup \{x_h = e_h = t_h = \mathsf{true}\} \cup$$
$$\{c_j = \mathsf{true}\} \cup \{q = \mathsf{true}\} \cup \{r_i = \mathsf{true}, s_i = \mathsf{true}\}$$

The five clauses are falsified by M. Since the two of them $x_h \vee \neg q$ and $e_h \vee \neg q$ are in A_R, this set is also falsified by M. As a result, M is not a model of $A_R \cup A_T \cup A_C \cup A_B$. This formula is equivalent to $A_N \cup A_T \cup A_C \cup A_B$, which is therefore falsified by M. In formulae, $M \not\models A_N \cup A_T \cup A_C \cup A_B$.

The formula $A_N \cup A_T \cup A_C \cup A_B$ contains a clause falsified by M. Since $M \models A_T \cup A_C \cup A_B$, this clause is in A_N but not in $A_T \cup A_C \cup A_B$. In formulae, $M \not\models c$ for some $c \in A_N$ and $c \notin A_T \cup A_C \cup A_B$. This clause is entailed by $A_R \cup A_T \cup A_C \cup A_B$ because this formula entails all of $A_N \cup A_T \cup A_C \cup A_B$, and c is in A_N. In formulae, $A_R \cup A_T \cup A_C \cup A_B \models c$.

This clause c contains either x_h, e_h or t_h. This is proved by deriving a contradiction from the assumption that c does not contain any of these three literals. Since $M \not\models c$, the clause c contains only literals that are falsified by M. Not all of them: it does not contain x_h, e_h and t_h by assumption. It does not contain $\neg x_h$, $\neg e_h$ and $\neg t_h$ either because it would otherwise be satisfied by M. As a result, c is also falsified by M', which is the same as M but for the values of x_h, e_h and t_h. At the same time, M' satisfies $A_R \cup A_T \cup A_C \cup A_B$, contradicting $A_R \cup A_T \cup A_C \cup A_B \models c$. This contradiction proves that c contains either x_h, e_h or t_h.

From the fact that c contains either x_h, e_h or t_h, that is a consequence of $A_R \cup A_T \cup A_C \cup A_B$, and that is in a minimal-size formula, it is now possible to prove that c contains either $x_h \vee \neg q$, $x_h \vee \neg r_h$, $e_h \vee \neg q$, $e_h \vee \neg s_h$ or $t_h \vee \neg q$.

Since c is entailed by $A_R \cup A_T \cup A_C \cup A_B$, a subset of c follows from resolution from it: $A_R \cup A_T \cup A_C \cup A_B \vdash c'$ with $c' \subseteq c$. This implies $A_N \cup A_T \cup A_C \cup A_B \models c'$ by equivalence. If $c' \subset c$, then $A_N \cup A_T \cup A_C \cup A_B$ would not be minimal because it contained a non-minimal clause $c \in A_N$. Therefore, $A_R \cup A_T \cup A_C \cup A_B \vdash c$.

The only two clauses of $A_R \cup A_T \cup A_C \cup A_B$ that contain x_h are $x_h \vee \neg q$ and $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee x_h \vee \neg r_h$. They contain either $\neg q$ or $\neg r_h$. These literals are only resolved out by clauses containing their negations q and r_h. No clause contains q and the only clause that contains r_h is $r_h \vee \neg q$, which contains $\neg q$. If a result of resolution contains x_h, it also contains either $\neg q$ or $\neg r_h$. This applies to c because it is a result of resolution.

The same applies if c contains e_h: it also contains either $\neg q$ or $\neg s_i$.

The case of $t_h \in c$ is a bit different. The only two clauses of $A_R \cup A_T \cup A_C \cup A_B$ that contain t_h are $\neg x_h \vee t_h$ and $\neg e_h \vee t_h$. Since both are in A_T and $c \notin A_T$, they are not c. The first clause $\neg x_h \vee t_h$ only resolves with $x_h \vee \neg q$ or $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee x_h \vee \neg r_h$, but resolving with the latter generates a tautology. The result of resolving $\neg x_h \vee t_h$ with $x_h \vee \neg q$ is $t_h \vee \neg q$; no clause contains q. Therefore, c can only be $t_h \vee \neg q$. The second clause $\neg e_h \vee t_h$ leads to the same conclusion.

In summary, c contains either $x_h \vee \neg q$, $x_h \vee \neg r_h$, $e_h \vee \neg q$, $e_h \vee \neg s_h$ or $t_h \vee \neg q$. In all these cases it contains at least two literals. This is the case for every index h; therefore, A_N contains at least n clauses of two literals. Every minimal CNF formula equivalent to $A_R \cup A_T \cup A_C \cup A_B$ has size at least $2 \times n$ plus the size of $A_T \cup A_C \cup A_B$. This sum is exactly k. This proves that every minimal CNF formula expressing forgetting contains at least k literal occurrences. Worded differently, every CNF formula expressing forgetting has size at least k.

Formula F is unsatisfiable

The claim is that no CNF formula of size k expresses forgetting if F is unsatisfiable. This is proved by deriving a contradiction from the assumption that such a formula exists.

It has been proved that every CNF formula expressing forgetting is equivalent to $A_R \cup A_T \cup A_C \cup A_B$ and that the minimal equivalent CNF formulae are $A_N \cup A_T \cup A_C \cup A_B$ for some set A_N that contains clauses that include either $x_h \vee \neg q$, $x_h \vee \neg r_h$, $e_h \vee \neg q$, $e_h \vee \neg s_h$ or $t_h \vee \neg q$ for each index h.

If A_N contains other clauses, or more than one clause for each h, or these clauses contain other literals, the size of $A_N \cup A_T \cup A_C \cup A_B$ is larger than $k = 2 \times n + ||A_T|| + ||A_C|| + ||A_B||$, contradicting the assumption. This proves that every formula of size k that is equivalent to $A_R \cup A_T \cup A_C \cup A_B$ is equal to $A_N \cup A_T \cup A_C \cup A_B$ where A_N contains exactly one clause among $x_h \vee \neg q$, $x_h \vee \neg r_h$, $e_h \vee \neg q$, $e_h \vee \neg s_h$ or $t_h \vee \neg q$ for each index h.

The case $x_h \vee \neg r_h \in A_N$ is excluded. It would imply

$A_R \cup A_T \cup A_C \cup A_B \models x_h \vee \neg r_h$, which implies the redundancy of $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg c_1 \vee \cdots \vee \neg c_m \vee x_h \vee \neg r_h \in A_B$ contrary to its previously proved superirredundancy. A similar argument proves $e_h \vee \neg s_h \not\in A_N$.

The conclusion is that every formula of size k that is equivalent to $A_R \cup A_T \cup A_C \cup A_B$ is equal to $A_N \cup A_T \cup A_C \cup A_B$ where A_N contains exactly one clause among $x_h \vee \neg q$, $e_h \vee \neg q$, $t_h \vee \neg q$ for each index h.

If F is unsatisfiable, all such formulae are proved to be satisfied by a model that falsifies $A_R \cup A_T \cup A_C \cup A_B$, contrary to the assumed equivalence.

Let M be the model that assigns $q = \mathsf{true}$ and $t_i = \mathsf{true}$, and assigns $x_i = \mathsf{true}$ and $e_i = \mathsf{false}$ if $x_i \vee \neg q \in A_N$ and $x_i = \mathsf{false}$ and $e_i = \mathsf{true}$ if $e_i \vee \neg q \in A_N$ or $t_i \vee \neg q \in A_N$. All clauses of A_N and A_T are satisfied by M.

This model M can be extended to satisfy all clauses of $A_C \cup A_B$. Since F is unsatisfiable, M falsifies at least a clause $f_j \in F$. Let M' be the model obtained by extending M with the assignments of c_j to false, all other variables in C to true and all variables r_i and s_i to true. This extension satisfies all clauses of A_B either because it sets c_j to false or because it sets r_i and s_i to true. It also satisfies all clauses of A_C that do not contain c_j because it sets all variables of C but c_j to true.

The only clauses that remain to be proved satisfied are the clauses of A_C that contain c_j. They are $\neg x_i \vee c_j$ for all $x_i \in f_j$ and $\neg e_i \vee c_j$ for all $\neg x_i \in f_j$. Since M' falsifies f_j, it falsifies every $x_i \in f_j$; therefore, it satisfies $\neg x_i \vee c_j$. Since M' falsifies f_j, it falsifies every $\neg x_i \in f_j$; since by construction it assigns e_i opposite to x_i, it falsifies e_i and therefore satisfies $\neg e_i \vee c_j$.

This proves that M' satisfies $A_N \cup A_T \cup A_C \cup A_B$. It does not satisfy $A_R \cup A_T \cup A_C \cup A_B$. If $x_1 \vee \neg q \in A_N$, then M' sets x_1 to true and e_1 to false; therefore, it does not satisfy $e_1 \vee \neg q \in A_R$. Otherwise, M' sets x_1 to false and e_1 to true; therefore, it does not satisfy $x_1 \vee \neg q \in A_N$.

This contradicts the assumption that $A_N \cup A_T \cup A_C \cup A_B$ is equivalent to $A_R \cup A_T \cup A_C \cup A_B$. The assumption that it has size k is therefore false. □

Lemma 12. *Checking whether forgetting some variables from a minimal-size Horn formula is expressed by a CNF or Horn formula bounded by a certain size is D^p-hard.*

Proof. For every CNF formula F, Lemma 10 ensures the existence of a minimal-size Horn formula A, a set of variables X_A and an integer k such that forgetting all variables except X_A from A is expressed by a Horn formula of size k if F is unsatisfiable and is only expressed by larger CNF formulae otherwise.

For every CNF formula G, Lemma 11 ensures the existence of a minimal-size Horn formula B, a set of variables X_B and an integer l such that forgetting all variables except X_B from B is expressed by a Horn formula of size l if G is satisfiable and is only expressed by larger CNF formulae otherwise.

The prototypical D^p-hard problem is that of establishing whether a formula F is satisfiable and another G is unsatisfiable. If the alphabets of the two formulae G and F are not disjoint, they can be made so by renaming one of them to fresh variables because renaming does not affect satisfiability. The same applies to the formulae B and A respectively build from them according to Lemma 10 and Lemma 11 because renaming does not change the minimal size of forgetting either. Lemma 7 proves that $A \cup B$ can be minimally expressed by $C \cup D$ where C minimally expresses forgetting from A and D from B. The size of these two formulae is l and k if G is unsatisfiable and F satisfiable. If G is satisfiable, then D is larger than k while C is still large at least l; the minimal expression of forgetting $A \cup B$ is therefore strictly larger than $k + l$. The same happens if F is unsatisfiable.

This proves that the problem of checking the satisfiability of a formula and the unsatisfiability of another reduces to the problem of checking the size of the minimal expression of forgetting from Horn formulae. □

Theorem 6. *Checking whether forgetting some variables from a Horn formula is expressed by a CNF or Horn formula bounded by a certain size expressed in unary is D^p-hard and in Σ_2^p, and remains hard even if the formula is restricted to be of minimal size.*

Proof. The problem belongs to Σ_2^p because it can be expressed as the existence of a formula of the given size or less that expresses forgetting the given variables from the formula. In turn, expressing forgetting is by Theorem 1 the same as the equiconsistency with a set of literals containing all variables not to be forgotten. This condition can be expressed by the following metaformula where A is the formula, Y are the variables not to be forgotten and k the size bound.

$$\exists B \, . \, ||B|| \leq k \text{ and } \forall S \, . \, Var(S) \subseteq Y \Rightarrow (S \cup A \not\models \bot \Leftrightarrow S \cup B \not\models \bot)$$

Both B and S are bounded in size: the first by k, the second by the number of variables in Y. Since consistency is polynomial for Horn formulae, this is a $\exists\forall$QBF, which proves membership to Σ_2^p.

Hardness for D^p is proved by Lemma 12. □

Lemma 13. *There exists a polynomial algorithm that turns a CNF formula F into a minimal-size CNF formula A, a subset $X_C \subseteq Var(A)$ and a number k such that forgetting all variables from A except X_C is expressed by a CNF formula of size k if $\forall X \exists Y.F$ is valid and only by CNF formulae of size $k + 2$ or greater otherwise.*

Proof. Let $F = \{f_1, \ldots, f_m\}$ and its variables be $X = \{x_1, \ldots, x_n\}$ and Y. Checking the validity of $\forall X \exists Y.F$ remains Π_2^p-hard even if F is satisfiable: if F is not satisfiable, $\forall X \exists Y.F$ can be turned into the equivalent formula $\forall X \cup \{s\} \exists Y.s \vee F$, and $s \vee F$ is satisfiable. The following assumes F satisfiable, which is proved correct by this argument.

The reduction is based on an extended alphabet with the additional fresh variables $E = \{e_1, \ldots, e_n\}$, $C = \{c_1, \ldots, c_m\}$ and $\{a, b, q, r\}$. The formula A, the set of variables X_C and the number k are:

$$A = \{f_j \vee c_j \vee q \mid f_j \in F\} \cup$$
$$\{\neg c_j \vee r \mid f_j \in F\} \cup$$

$$\{\neg r \vee \neg a \vee b \vee q\} \cup$$
$$\{a \vee \neg b \vee q\} \cup$$
$$\{x_i \vee e_i \mid x_i \in X\}$$
$$X_C = X \cup E \cup \{a, b, q\}$$
$$k = 2 \times n + 3$$

A short explanation of how the reduction works precedes its formal proof. The key is how a model over $X \cup \{q\}$ extends to a model of A, in particular its possible values of a and b. All models over $X \cup \{q\}$ that satisfy q can be extended to satisfy A: all clauses not containing q are satisfied by setting $r = \mathsf{true}$ and e_i opposite to x_i; satisfaction is not affected by the values a and b. The remaining models set $q = \mathsf{false}$. For these models, the satisfaction of a clause f_j for some values of Y makes $f_j \vee c_j \vee q$ satisfied even if $c_j = \mathsf{false}$. In turn, $c_j = \mathsf{false}$ satisfies $\neg c_j \vee r$ even if $r = \mathsf{false}$, which satisfies $\neg r \vee \neg a \vee b \vee q$ regardless of the values of a and b; the values of a and b only need to satisfy $a \vee \neg b \vee q$. Otherwise, the falsity of f_j for all values of Y imposes $c_j = \mathsf{true}$ to satisfy $f_j \vee c_j \vee q$, which makes $\neg c_j \vee r$ require $r = \mathsf{true}$, which turns $\neg r \vee \neg a \vee b \vee q$ into $\neg a \vee b \vee q$, making the literals $\neg a$ and b necessary in addition to a and $\neg b$. A key point is that the variables Y are part of the clauses f_j, whose satisfiability affects the necessity of setting c_j, but they disappear in the minimal formulae as they are to be forgotten.

The proof comprises four steps: first, A is proved minimal as required by the claim of the lemma; second, k literals that are in every formula that expresses forgetting regardless of the validity of the QBF are identified; third, a formula of size k expressing forgetting when the QBF is valid is determined; fourth, every formula expressing forgetting contains at least two further literals if the QBF is invalid.

Minimality of A.

Follows from Lemma 2 since all clauses of A are superirredundant. This is in turn proved by showing substitutions that disallow all resolutions, which proves the superredundancy of the remaining clauses by Lemma 5 and Lemma 3.

The substitution that replaces with true the variables a, b, r, all e_i and all c_j with $j \neq h$ for every given h such that $f_h \in F$ removes all clauses but $f_h \vee c_h \vee q$, which is therefore superirredundant.

The clauses $\neg c_j \vee r$ are proved superirredundant by substituting q and all variables e_i with true, which removes all other clauses. The clauses $\neg c_j \vee r$ do not resolve because they do not contain opposite literals.

Two other clauses are proved superirredundant by the substitution that replaces all variables e_i with true, all c_j with false, and $X \cup Y$ with some values that satisfy F; such values exist because F is by assumption satisfiable. This substitution removes all clauses but $\neg r \vee \neg a \vee b \vee q$ and $a \vee \neg b \vee q$, which only resolve in tautologies.

Finally, the clauses $x_h \vee e_h$ are proved superirredundant by replacing q and r with true, which removes all other clauses. Since the clauses $x_h \vee e_h$ only contain positive literals, they do not resolve.

Necessary literals.

Regardless of the validity of $\forall X \exists Y.F$, the literals $X \cup E \cup \{a, \neg b, q\}$ are necessary in every CNF formula that expresses forgetting all variables except X_C from A. This is proved by Lemma 9, exhibiting a set of literals S such that $S \cup A$ is consistent, but $S \backslash \{l\} \cup \{\neg l\} \cup A$ is not for every $l \in X \cup E \cup \{a, \neg b, q\}$.

The first set is $S = \{x_i, \neg e_i, a, b, \neg q\}$, which is consistent with A because of the model that satisfies S and assigns r and all variables c_j to true. Changing x_i to $\neg x_i$ violates the clause $x_i \vee e_i$. Changing a to false violates $a \vee \neg b \vee q$. This proves that a and all variables x_i are necessary by Lemma 9.

The second set is $S = \{\neg x_i, e_i, \neg a, \neg b, \neg q\}$, which is consistent with A because of the model that satisfies S and assigns r and all variables c_j to true. Changing e_i to $\neg e_i$ violates $x_i \vee e_i$, changing b to true violates $a \vee \neg b \vee q$. This proves that e_i and $\neg b$ are necessary by Lemma 9.

The third set is $S = \{x_i, e_i, \neg a, b, q\}$, which is consistent with A because of the model that satisfies S and assigns r and all variables c_j to true. Changing q to false violates the clause $a \vee \neg b \vee q$, proving that q is necessary.

In summary, all literals in $X \cup E \cup \{a, \neg b, q\}$ occur in every CNF formula expressing forgetting all variables except X_C from A. These literals are $2 \times n + 3$. This is a part of the claim: no CNF formula expressing forgetting is smaller than $2 \times n + 3$.

Forgetting when $\forall X \exists Y.F$ is valid

If $\forall X \exists Y.F$ is valid, forgetting is expressed by $B = \{a \vee \neg b \vee q\} \cup \{x_i \vee e_i \mid x_i \in X\}$, which has the required size $k = 2 \times n + 3$ and variables $X_C = X \cup E \cup \{a, b, q\}$. Theorem 1 proves that this formula expresses forgetting: every set S of literals of X_C that contains all variables of X_C is consistent with B if and only if it is consistent with A.

Since B only contains clauses of A, every set of literals S that is consistent with A is also consistent with B. The claim follows from proving the converse for every set of literals S over X_C that mentions all variables of X_C.

The assumption is that $S \cup B$ is consistent; the claim is that $S \cup A$ is consistent. Since $S \cup B$ is consistent, it has a model M. Let M_X be its restriction to the variables X and M'_Y to Y. By assumption, $\forall X \exists Y.F$ is valid. Therefore, $M_X \cup M_Y$ satisfies F for some truth evaluation M_Y over Y. Since S is satisfied by M and does not mention any variable Y, it is also satisfied by $M \backslash M'_Y \cup M_Y$. The truth evaluation $M_C = \{c_j = \mathsf{false} \mid f_j \in F\} \cup \{r = \mathsf{false}\}$ satisfies all clauses $\neg c_j \vee r$ and $\neg r \vee \neg a \vee b \vee q$. Since $M_X \cup M_Y$ satisfies all clauses $f_j \in F$, the union $M \backslash M'_Y \cup M_Y \cup M_C$ satisfies all clauses $f_j \vee c_j \vee q$ of A. This proves that $M \backslash M'_Y \cup M_Y \cup M_C$ satisfies all clauses of A that B does not contain.

Forgetting when $\forall X \exists Y.F$ is invalid

All CNF formulae that express forgetting have been proved to mention $X \cup E \cup \{a, \neg b, q\}$. If $\forall X \exists Y.F$ is invalid, they all mention $\neg a$ and b as well.

This is proved by Lemma 9: a set of literals S over X_C is shown to be consistent with A while $S \backslash \{\neg a\} \cup \{a\}$ is not. A similar set is shown for b.

Since $\forall X \exists Y.F$ is invalid, for some interpretation M_X over X the interpretation $M_X \cup M_Y$ falsifies F for every interpretation M_Y over Y. The required set S is built from M_X: it contains the literals over x_i that are satisfied by M_X and $\neg a$, $\neg b$ and $\neg q$.

$$S = \{x_i \mid M_X \models x_i\} \cup \{\neg x_i \mid M_X \models \neg x_i\} \cup \{\neg a, \neg b, \neg q\}$$

By construction, M_X satisfies the first part of S. The model $M_O = \{a = \mathsf{false}, b = \mathsf{false}, q = \mathsf{false}\}$ satisfies the second. Therefore, $M_X \cup M_O$ satisfies S.

The consistency of $S \cup A$ is shown by proving that $M_X \cup M_O$ can be extended to the other variables to satisfy A. This extension is $M_X \cup M_Y \cup M_O \cup M_N \cup M_C$, where M_Y is an arbitrary model over Y, M_N assigns every e_i opposite to x_i in M_X and M_C is $\{c_j = \mathsf{true} \mid f_j \in F\} \cup \{r = \mathsf{true}\}$. The clauses $f_j \vee c_j \vee q$ are satisfied because c_j is true, the clauses $\neg c_j \vee r$ because r is true, the clause $\neg r \vee \neg a \vee b \vee q$ because a is false, $a \vee \neg b \vee q$ because b is false, the clauses $x_i \vee e_i$ because $M_N \models e_i$ if $M_X \not\models x_i$.

This proves that $M_X \cup M_O \cup M_N \cup M_C$ satisfies $S \cup A$, which is therefore satisfiable.

The claim is a consequence of $S' = S \backslash \{\neg a\} \cup \{a\}$ being inconsistent with A.

$$S' = \{x_i \mid M_X \models x_i\} \cup \{\neg x_i \mid M_X \models \neg x_i\} \cup \{a, \neg b, \neg q\}$$

This is proved by contradiction: a model M' is assumed to satisfy $S' \cup A$. Since M' satisfies S', it assigns the variables x_i the same as M_X. Let M_Y be the restriction of M' to the variables Y. By assumption, M_X is a model over X that cannot be extended to Y to satisfy F. As a result, $M_X \cup M_Y \not\models F$. Therefore, M'

falsifies at least a clause $f_j \in F$. Since M' satisfies $f_j \vee c_j \vee q$ but falsifies both f_j and q, it satisfies c_j. It also satisfies r because it satisfies $\neg c_j \vee r$ and falsifies c_j. Since M' satisfies S' it satisfies a and falsifies b and q. The conclusion is that all literals of $\neg r \vee \neg a \vee b \vee q \in A$ are false, contrary to the assumption that M' satisfies A.

A similar set S with a and b in place of $\neg a$ and $\neg b$ proves that expressing forgetting also requires b. □

Lemma 14. *There exists a polynomial algorithm that turns a DNF formula $F = f_1 \vee \cdots \vee f_m$ over variables $X \cup Y$ into a minimal-size CNF formula A, a subset $X_C \subseteq Var(A)$ and a number k such that forgetting all variables except X_C from A is expressed by a CNF formula of size k if $\exists X \forall Y.F$ is valid, and only by larger CNF formulae otherwise.*

Proof. Let $F = f_1 \vee \cdots \vee f_m$ be the DNF formula over variables $X \cup Y$. The reduction employs additional variables: $O = \{o_i \mid x_i \in X\}$, $E = \{e_i \mid x_i \in X\}$, $P = \{p_i \mid x_i \in X\}$, $T = \{t_i \mid x_i \in X\}$, $D = \{d_j \mid f_j \in F\}$, $R = \{r_i \mid x_i \in X\}$, $S = \{s_i \mid x_i \in X\}$ and q. The formula A, the alphabet X_C and the number k are as follows. The formula looks Horn when using $\neg q$ in place of q, but is not: $\neg(f_j[e_i/\neg x_i]) \vee d_j$ replaces all negative occurrences of x_i with e_i, but does not touch the negative occurrences of y_i. This clause is Horn when Y is empty. This makes the lemma imply the analogous lemma for the Horn case only when Y is empty, and therefore proves the NP-hardness of that restriction and not its Σ_2^p-hardness.

$$\begin{aligned}
A &= A_F \cup A_T \cup A_D \cup A_B \\
A_F &= \{x_i \vee \neg o_i, o_i \vee q \mid x_i \in X\} \cup \{e_i \vee \neg p_i, p_i \vee q \mid x_i \in X\} \\
A_T &= \{\neg x_i \vee t_i,\ \neg e_i \vee t_i \mid x_i \in X\} \\
A_D &= \{\neg(f_j[e_i/\neg x_i]) \vee d_j \mid f_j \in F\} \\
A_B &= \{\neg t_1 \vee \cdots \vee \neg t_n \vee \neg d_j \vee x_i \vee \neg r_i, r_i \vee q \mid x_i \in X\ , f_j \in F\} \cup \\
&\qquad \{\neg t_1 \vee \cdots \vee \neg t_n \vee \neg d_j \vee e_i \vee \neg s_i, s_i \vee q \mid x_i \in X,\ f_j \in F\} \\
X_C &= X \cup E \cup Y \cup T \cup D \cup R \cup S \cup \{q\} \\
k &= 2 \times n + ||A_T \cup A_D \cup A_B||
\end{aligned}$$

The reduction works because every minimal CNF formula that expresses forgetting contains at least one among $x_h \vee q$, $e_h \vee q$ and $t_h \vee q$ for each h, and all of $A_T \cup A_D \cup A_B$. This proves the lower bound k. If the QBF is valid, for some evaluation over X the formula F is true regardless of Y. Choosing the clauses $x_h \vee q$, $e_h \vee q$ or $t_h \vee q$ that correspond to this model, some clause $\neg(f_j[e_i/\neg x_i]) \vee d_j$ of A_D implies $q \vee d_j$ because $f_j[e_i/\neg x_i]$ is true for all values of Y. This allows A_B to entail all remaining clauses. If the QBF is not valid, no clause $q \vee d_j$ is entailed.

The formal proof requires five steps: first, every formula expressing forgetting is equivalent to a certain formula $A_R \cup A_T \cup A_D \cup A_B$; second, A is a minimal CNF formula and the clauses of $A_T \cup A_D \cup A_B$ are in all minimal CNF formulae equivalent to $A_R \cup A_T \cup A_D \cup A_B$; third, forgetting is expressed by a formula of size k if the QBF is valid; fourth, every minimal CNF formula expressing forgetting contains either $x_h \vee q$, $e_h \vee q$ or $t_h \vee q$ for each h; fifth, if the QBF is invalid then forgetting is only expressed by formulae larger than k.

Effect of forgetting.

The variables to forget are $O \cup P$. Each is contained only in two clauses of A, with opposite signs. Resolving them produces the clauses in the following set A_R.

$$A_R = \{x_i \vee q, e_i \vee q \mid x_i \in X\}$$

By Theorem 5, forgetting is expressed by $A_R \cup A_T \cup A_D \cup A_B$. Therefore, all formulae that express forgetting are equivalent to this formula.

Superirredundancy.

All clauses of $A_F \cup A_T \cup A_D \cup A_B$ are proved superirredundant in $A_F \cup A_R \cup A_T \cup A_D \cup A_B$. Both A and $A_R \cup A_T \cup A_D \cup A_B$ are subsets of this formula; therefore, the superirredundant clauses are superirredundant in both formulae by Lemma 4. Since A comprises exactly them, it is minimal thanks to Lemma 2. Since all formulae expressing forgetting are equivalent to $A_R \cup A_T \cup A_D \cup A_B$, where $A_T \cup A_D \cup A_B$ are superirredundant, these clauses are in all formulae expressing forgetting, according to Lemma 1.

Superirredundancy is proved applying a substitution to the formula so that the resulting clauses do not resolve and are not contained in one another. This condition proves them superirredundant by Lemma 3. Lemma 5 implies their superirredundancy in the original formula.

Replacing all variables X, E, T and D with true removes from the formula $A_F \cup A_R \cup A_T \cup A_D \cup A_B$ all clauses but $o_i \vee q$, $p_i \vee q$, $r_i \vee q$ and $s_i \vee q$. These clauses do not resolve because they only contain positive literals. None is contained in another.

Replacing all variables R and S with false and all variables T, D and q with true removes from the formula $A_F \cup A_R \cup A_T \cup A_D \cup A_B$ all clauses but the clauses $x_i \vee \neg o_i$ and $e_i \vee \neg p_i$. They are not contained in one another; they do not resolve because they do not contain opposite literals.

Replacing all variables O, P, R and S with false and D and q with true removes all clauses but $\neg x_i \vee t_i$ and $\neg e_i \vee t_i$. These clauses do not resolve because they do not contain opposite literals; they are not contained in one another.

Replacing all variables O, P, R and S with false and T, $D \backslash \{d_h\}$ and q with true removes all clauses but $(\neg f_h[e_i/\neg x_i]) \vee d_h$, which is therefore superirredundant.

The last substitution replaces all variables $X \backslash \{x_h\}$, E, O, P, $R \backslash \{r_h\}$ and S with false, all variables $D \backslash \{d_l\}$ and q with true, all variables y_i such that $y_i \in \neg f_l[e_i/\neg x_i]$ to true and all such that $\neg y_i \in \neg f_l[e_i/\neg x_i]$ to false. This substitution removes all clauses but $\neg x_h \vee t_h$, $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg d_l \vee x_h \vee \neg r_i$ and possibly $\neg(f_l[e_i/\neg x_i]) \vee d_l$. The latter clause is removed if it contains some variable y_i. It is removed if it contains some literal $\neg x_i$ with $i \neq h$. It is removed if it contains some literal $\neg e_i$. The only other literals it may contain are $\neg x_h$ and d_l; it contains both: d_l by construction, $\neg x_h$ because otherwise f_l would be empty. The remaining clauses are therefore $\neg x_h \vee t_h$, $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg d_l \vee x_i \vee \neg r_i$ and possibly $\neg x_h \vee d_h$. These clauses only resolve in tautologies, which proves the second superirredundant. A similar argument holds for $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg d_l \vee e_i \vee \neg s_i$.

Validity of $\exists X \forall Y . F$.

Let M be a model over variables X that makes F true regardless of the values of Y. Let $A'_R \subseteq A_R$ be the set of clauses $x_i \vee q$ such that $M \models x_i$ and $e_i \vee q$ such that $M \models \neg x_i$. This set has size $2 \times n$. Therefore, $A'_R \cup A_T \cup A_D \cup A_B$ has size $k = 2 \times n + ||A_T \cup A_D \cup A_B||$. This formula expresses forgetting if it is equivalent to $A_R \cup A_T \cup A_D \cup A_B$, which is the case if $A'_R \cup A_T \cup A_D \cup A_B \models A_R$. The claim is proved by showing that $A'_R \cup A_T \cup A_D \cup A_B$ entails A_R.

Either $x_h \vee q$ or $e_h \vee q$ is in A'_R for every h and these clauses respectively resolve with $\neg x_h \vee t_h$ and $\neg e_h \vee t_h$, producing $t_h \vee q$ in both cases. Each clause

$\neg t_1 \vee \cdots \vee \neg t_n \vee \neg d_j \vee x_h \vee \neg r_h$ resolve with them and with $r_h \vee q$ to $\neg d_j \vee x_h \vee q$. This clause further resolves with $\neg(f_j[e_i/\neg x_i]) \vee d_j$ to produce $\neg(f_j[e_i/\neg x_i]) \vee x_h \vee q$. This proves that $A'_R \cup A_T \cup A_D \cup A_B$ implies every clause $\neg(f_j[e_i/\neg x_i]) \vee x_h \vee q$ with $f_j \in F$. The following equivalence holds.

$$\begin{aligned}\{\neg(f_j[e_i/\neg x_i]) \vee x_h \vee q | f_j \in F\} &\equiv \left(\bigwedge \{\neg(f_j[e_i/\neg x_i]) \mid f_j \in F\}\right) \vee x_h \vee q \\ &\equiv \neg\left(\bigvee \{f_j[e_i/\neg x_i] \mid f_j \in F\}\right) \vee x_h \vee q\end{aligned}$$

$$\equiv \neg F[e_i/\neg x_i] \vee x_h \vee q$$

Since $A'_R \cup A_T \cup A_D \cup A_B$ implies the first set, it implies the last formula: $A'_R \cup A_T \cup A_D \cup A_B \models \neg F[e_i/\neg x_i] \vee x_h \vee q$.

Since M satisfies F regardless of Y, it follows that $\{x_i \mid M \models x_i\} \cup \{\neg x_i \mid M \models \neg x_i\} \models F$. Replacing each $\neg x_i$ with e_i in both sides of this entailment turns it into $\{x_i \mid M \models x_i\} \cup \{e_i \mid M \models \neg x_i\} \models F[e_i/\neg x_i]$. Disjoining both terms with q results into $A'_R \models F[e_i/\neg x_i] \vee q$.

This entailment and the previously proved $A'_R \cup A_T \cup A_D \cup A_B \models \neg F[e_i/\neg x_i] \vee x_h \vee q$ imply $A'_R \cup A_T \cup A_D \cup A_B \models x_h \vee q$.

The same holds for $e_h \vee q$ by symmetry. Therefore, $A'_R \cup A_T \cup A_D \cup A_B$ implies every clause of A_R.

Necessary clauses.

All formulae that express forgetting are equivalent to $A_R \cup A_T \cup A_D \cup A_B$ and therefore contain all its superirredundant clauses $A_T \cup A_D \cup A_B$, as Lemma 1 proves. As a result, they have the form $A_N \cup A_T \cup A_D \cup A_B$ for some set of clauses A_N. It is now shown that all equivalent CNF formulae of minimal size contain a clause that include either $x_h \vee q$, $x_h \vee \neg r_h$, $e_h \vee q$, $e_h \vee \neg s_h$, or $t_h \vee q$ for each h.

Since $A_N \cup A_T \cup A_D \cup A_B$ is equivalent to $A_R \cup A_T \cup A_D \cup A_B$, it entails $x_h \vee q \in A_R$. This clause is not satisfied by the following model.

$$M = \{x_i = e_i = t_i = \mathsf{true} \mid i \neq h\} \cup \{x_h = e_h = t_h = \mathsf{false}\} \cup$$
$$\{d_j = \mathsf{true} \mid f_j \in F\} \cup \{r_i = s_i = \mathsf{true}\} \cup \{q = \mathsf{false}\}$$

This model satisfies all clauses of $A_T \cup A_D \cup A_B$. If A_N also satisfied it, $A_N \cup A_T \cup A_D \cup A_B$ would have a model that falsifies $x_h \vee q$, which it instead entails. As a result, A_N contains a clause c that M falsifies. Since $A_N \cup A_T \cup A_D \cup A_B$ is a formula of minimal size, it entails no proper subset of c. By equivalence, the same applies to $A_R \cup A_T \cup A_D \cup A_B$.

$$M \not\models c$$
$$A_R \cup A_T \cup A_D \cup A_B \models c$$
$$A_R \cup A_T \cup A_D \cup A_B \models c' \text{ implies } c' \not\subset c$$

If c contains neither x_h, e_h nor t_h, it would still be falsified by the model that is the same as M except that it assigns x_h, e_h and t_h to true. This model satisfies $A_R \cup A_T \cup A_D \cup A_B$. As a result, $A_R \cup A_T \cup A_D \cup A_B \cup \neg(c)$ is consistent, contradicting $A_R \cup A_T \cup A_D \cup A_B \models c$. This proves that c contains either x_h, e_h or t_h.

Since these three variables are negative in M and $M \not\models c$, they are positive in c. In other words, c contains either x_h, e_h or t_h unnegated.

Since c is entailed by $A_R \cup A_T \cup A_D \cup A_B$, but none of its proper subsets does, it follows from resolution: $A_R \cup A_T \cup A_D \cup A_B \vdash c$.

If c contains x_h, it also contains either q or $\neg r_h$. This is proved as follows. Since c is the root of a resolution tree and contains x_h, this literal is also in one of the leaves of resolution. The only clauses of $A_R \cup A_T \cup A_D \cup A_B$ containing x_h are $x_h \vee q$ and all clauses $\neg t_1 \vee \dots \vee \neg t_n \vee \neg d_j \vee x_h \vee \neg r_h$. The first does not resolve over q because the formula does not contain $\neg q$. The other clauses only resolve over r_h with $r_h \vee q$, which introduces q, which again cannot be removed by resolution. Since c is obtained by resolution, if it contains x_h it also contains either $\neg r_h$ or q.

By symmetry, if c contains e_h it also contains either $\neg s_h$ or q.

The other case is that c contains t_h. The only clauses of $A_R \cup A_T \cup A_D \cup A_B$ that contain t_h are $\neg x_h \vee t_h$ and $\neg e_h \vee t_h$. These clauses are satisfied by M while c is not, therefore c is not one of them. The first clause $\neg x_h \vee t_h$ only resolves over x_h with $x_h \vee q$ and all clauses $\neg t_1 \vee \dots \vee \neg t_n \vee \neg d_j \vee x_h \vee \neg r_h$, but resolving with the latter only generates tautologies. Therefore, the first step of resolution is necessarily $\neg x_h \vee t_h, x_h \vee q \vdash t_h \vee q$. Since none of the involved clauses contains $\neg q$, every clause obtained from resolution that contains t_h also contains q. This also includes c. The same holds by symmetry for $\neg e_h \vee t_h$.

This proves that every minimal-size CNF formula expressing forgetting contains a clause that includes either $x_h \vee q$, $x_h \vee \neg r_h$, $e_h \vee q$, $e_h \vee \neg s_h$, or $t_h \vee q$ for each h.

Falsity of $\exists X \forall Y.F$.

The falsity of $\exists X \forall Y.F$ contradicts the existence of a minimal-size CNF formula of size k expressing forgetting. The relevant results proved so far are: every CNF formula expressing forgetting has size k or more and is equivalent to $A_R \cup A_T \cup A_D \cup A_B$; the minimal-size such formulae are $A_N \cup A_T \cup A_D \cup A_B$ where A_N contains, for each h, a clause that includes either $x_h \vee q$, $x_h \vee \neg r_h$, $e_h \vee q$, $e_h \vee \neg s_h$, or $t_h \vee q$.

A formula $A_N \cup A_T \cup A_D \cup A_B$ of size k expressing forgetting, if any, is minimal since no smaller formula expresses forgetting. Therefore, A_N includes, for each h, a clause containing one of the five disjunctions. Since these are not in $A_T \cup A_D \cup A_B$, the size of such formulae is $k = 2 \times n + ||A_T \cup A_D \cup A_B||$ if every clause of A_N is exactly one of the above disjunctions for each h. If A_N contains more than one clause for some h or the clause for some h contains more than two literals or A_N contains other clauses, the formula is not minimal.

The case $x_h \vee \neg r_h \in A_N$ can be excluded: it makes $\neg t_1 \vee \cdots \vee \neg t_n \vee \neg d_j \vee x_h \vee \neg r_h \in A_N$ redundant in $A_N \cup A_T \cup A_D \cup A_B$, contradicting the minimality of this formula. The case $e_h \vee \neg s_h \in A_N$ is excluded in the same way.

These exclusions leave A_N to contain exactly one among $x_h \vee q$, $e_h \vee q$, and $t_h \vee q$ for each h and nothing else.

The final step of the proof is that no such A_N makes $A_N \cup A_T \cup A_D \cup A_B$ equivalent to $A_R \cup A_T \cup A_D \cup A_B$ if $\exists X \forall Y.F$ is invalid. Nonequivalence is proved by exhibiting a model of the first formula that does not satisfy the second.

Let M_X be the model over X that contains $x_i = \mathsf{true}$ if $x_i \vee q \in A_N$ and $x_i = \mathsf{false}$ otherwise. Let M_N be the model that assigns every e_i opposite to x_i and $M_T = \{t_i = \mathsf{true} \mid t_i \in T\}$. By construction, $M_X \cup M_N \cup M_T \cup \{q = \mathsf{false}\}$ satisfies all clauses of A_N. It also falsifies either $x_i \vee q$ or $e_i \vee q$ for each i because it assigns false to q and to either x_i or e_i. It therefore falsifies A_R.

Since $\exists X \forall Y.F$ is invalid, every interpretation over X falsifies F with an interpretation over Y. Let M_Y be the interpretation over Y such that $M_X \cup M_Y \models \neg F$. Since $F = f_1 \vee \cdots \vee f_m$, it holds $M_X \cup M_Y \models \neg f_j$ for every $f_j \in F$. It follows $M_X \cup M_Y \cup M_N \models \neg f_j[e_i/\neg x_i]$ since M_N assigns every e_i opposite to x_i.

Merging the results proved in the preceding two paragraphs, $M_X \cup M_T \cup \{q = \mathsf{false}\} \cup M_N \cup M_Y$ satisfies both A_N and $\neg f_j[e_i/\neg x_i]$ for every $f_j \in F$.

This model can be extended to a model of $A_N \cup A_T \cup A_D \cup A_B$ by adding $M_O = \{d_j = \mathsf{false}\} \cup \{r_i = \mathsf{true}\} \cup \{s_i = \mathsf{true}\}$. The clauses of A_N are already proved satisfied. The clauses $\neg x_i \vee t_i \in A_T$ are satisfied because M_T contains $t_i = \mathsf{true}$. The clauses $(\neg f_j[e_i/\neg x_i]) \vee d_j$ are satisfied because $\neg f_j[e_i/\neg x_i]$ is. The clauses of A_B are satisfied because each contains either $\neg d_j$, r_i or s_i, and these literals are true in M_O.

This proves that $M_X \cup M_T \cup \{q = \mathsf{false}\} \cup M_N \cup M_Y \cup M_O$ satisfies $A_N \cup A_T \cup A_D \cup A_B$. It does not satisfy A_R, which means that it falsifies $A_R \cup A_T \cup A_D \cup A_B$. This proves that $A_N \cup A_T \cup A_D \cup A_B$ is not equivalent to $A_R \cup A_T \cup A_D \cup A_B$.

In summary, assuming that the QBF is not valid and that a CNF formula of size k expresses forgetting, it is proved that the formula does not express forgetting. This contradiction shows that no formula of size k expresses forgetting if the QBF is not valid. □

Lemma 15. *Checking whether forgetting a given set of variables from a minimal-size CNF formula is expressed by a CNF formula bounded by a certain size is D_2^p-hard.*

Proof. For every ∀QBF Lemma 13 ensures the existence of a minimal-size CNF formula A, a set of variables X_A and an integer k such that forgetting all variables except X_A from A is expressed by a CNF formula of size k if the QBF is valid and is only expressed by larger CNF formulae otherwise.

For every ∃QBF Lemma 14 ensures the existence of a minimal-size CNF formula B, a set of variables X_B and an integer l such that forgetting all variables except X_B from B is expressed by a CNF formula of size l if the QBF is valid and is only expressed by larger CNF formulae otherwise.

A D_2^p-hard problem is that of establishing whether an ∃QBF and a ∀QBF are both valid. If their alphabets are not disjoint, they can be made so by renaming one of them to fresh variables since renaming does not affect validity. The same applies to the formulae B and A respectively build from them according to Lemma 13 and Lemma 14 because renaming does not change the minimal size of forgetting either. Lemma 7 proves that forgetting from $A \cup B$ is expressed by $C \cup D$ where C expresses forgetting from A and D from B. The minimal size of two such CNF formulae is respectively k and l. If the QBFs are both valid, they are exactly k and l large. Otherwise, they are strictly larger than either k or l. The sum is $k + l$ if both QBFs are valid and is larger than $k + l$ otherwise. □

Theorem 7. *Checking whether forgetting some variables from a CNF formula is expressed by a CNF formula of a certain size expressed in unary is D_2^p-hard and in Σ_3^p, and remains hard even if the CNF formula is restricted to be of minimal size.*

Proof. Membership to Σ_3^p is proved first. The problem is the existence of a CNF formula of the given size or less that expresses forgetting the given variables from the formula. Theorem 1 reformulates forgetting in terms of equiconsistency with a set of literals containing all variables not to be forgotten. Forgetting withing a certain size is formalized by the following metaformula where A is the formula, Y the variables not to be forgotten and k the size bound.

$$\exists B \ . \ Var(B) \subseteq Y, \ ||B|| \leq k \text{ and } \forall S \ . \ Var(S) \subseteq Y \Rightarrow (\exists M \ . \ M \models S \cup A \Leftrightarrow \exists M' \ . \ M' \models S \cup B)$$

All four quantified entities are bounded in size: B by k, S and M' by the number of variables in Y and M by the number of variables in A. This is therefore a ∃∀∃QBF, which proves membership to Σ_3^p.

Hardness to D_2^p is proved by Lemma 15 in the restriction where A is minimal. □

References

[1] G. Antoniou, T. Eiter, K. Wang, Forgetting for defeasible logic, in: Proceedings of the Eighteenth Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-18), in: Lecture Notes in Computer Science, vol. 7180, Springer, 2012, pp. 77–91.

[2] A. Artale, J.C. Jung, A. Mazzullo, A. Ozaki, F. Wolter, Living without Beth and Craig: explicit definitions and interpolants in description logics with nominals (extended abstract), in: Stefan Borgwardt, Thomas Meyer (Eds.), Proceedings of the Thirdythird International Workshop on Description Logics (DL 2020), vol. 2663, 2020.

[3] G. Ausiello, A. D'Atri, D. Saccà, Minimal representation of directed hypergraphs, SIAM J. Comput. 15 (2) (1986) 418–431.

[4] M. Berthold, R. Gonçalves, M. Knorr, J. Leite, A syntactic operator for forgetting that satisfies strong persistence, Theory Pract. Log. Program. 19 (5–6) (2019) 1038–1055.

[5] M. Bílková, Uniform interpolation and propositional quantifiers in modal logics, Stud. Log. 85 (1) (2007) 1–31.

[6] G. Boole, Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities, Walton and Maberly, 1854.

[7] E. Boros, O. Cepek, A. Kogan, P. Kucera, Exclusive and essential sets of implicates of Boolean functions, Discrete Appl. Math. 158 (2) (2010) 81–96.

[8] D. Buchfuhrer, C. Umans, The complexity of Boolean formula minimization, J. Comput. Syst. Sci. 77 (1) (2011) 142–153.

[9] O. Cepek, P. Kucera, P. Savický, Boolean functions with a simple certificate for CNF complexity, Discrete Appl. Math. 160 (4–5) (2012) 365–382.

[10] O. Čepek, P. Kučera, On the complexity of minimizing the number of literals in Horn formulae, RUTCOR Research Report RRR 11-208, Rutgers University, 2008.

[11] O. Coudert, Two-level logic minimization: an overview, Integration 17 (2) (1994) 97–140.

[12] O. Coudert, T. Sasao, Two-level logic minimization, in: Logic Synthesis and Verification, Springer, 2002, pp. 1–27.

[13] A. Darwiche, Compiling knowledge into decomposable negation normal form, in: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99), 1999, pp. 284–289.
[14] A. Darwiche, Decomposable negation normal form, J. ACM 48 (4) (2001) 608–647.
[15] M. Davis, H. Putnam, A computing procedure for quantification theory, J. ACM 7 (1960) 201–215.
[16] J.P. Delgrande, A knowledge level account of forgetting, J. Artif. Intell. Res. 60 (2017) 1165–1213.
[17] J.P. Delgrande, K. Wang, A syntax-independent approach to forgetting in disjunctive logic programs, in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015), AAAI Press, 2015, pp. 1482–1488.
[18] J.P. Delgrande, R. Wassermann, Horn clause contraction functions, J. Artif. Intell. Res. 48 (2013) 475–511.
[19] N. Eén, A. Biere, Effective preprocessing in SAT through variable and clause elimination, in: International Conference on Theory and Applications of Satisfiability Testing, 2005, pp. 61–75.
[20] T. Eiter, G. Ianni, R. Schindlauer, H. Tompits, K. Wang, Forgetting in managing rules and ontologies, in: 2006 IEEE / WIC / ACM International Conference on Web Intelligence (WI 2006), IEEE Computer Society Press, 2006, pp. 411–419.
[21] T. Eiter, G. Kern-Isberner, A brief survey on forgetting from a knowledge representation and perspective, Kuenstl. Intell. 33 (1) (2019) 9–33.
[22] T. Eiter, K. Wang, Forgetting and conflict resolving in disjunctive logic programming, in: Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI 2006), 2006, pp. 238–243.
[23] E. Erdem, P. Ferraris, Forgetting actions in domain descriptions, in: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007), AAAI Press, 2007, pp. 409–414.
[24] R. Fagin, J.Y. Halpern, Y. Moses, M. Vardi, Reasoning About Knowledge, The MIT Press, 1995.
[25] R. Feng, E. Acar, S. Schlobach, Y. Wang, W. Liu, On sufficient and necessary conditions in bounded CTL, Technical Report arXiv:2003.06492 [abs], Computing Research Repository (CoRR), 2020.
[26] P. Fišer, J. Hlavička, BOOM-a heuristic Boolean minimizer, Comput. Inform. 22 (1) (2012) 19–51.
[27] R. Gonçalves, M. Knorr, J. Leite, The ultimate guide to forgetting in answer set programming, in: Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2016), AAAI Press/The MIT Press, 2016, pp. 135–144.
[28] G. Gottlob, C.G. Fermüller, Removing redundancy from a clause, Artif. Intell. 61 (1993) 263–289.
[29] P.L. Hammer, A. Kogan, Optimal compression of propositional Horn knowledge bases: complexity and approximation, Artif. Intell. 64 (1) (1993) 131–145.
[30] P.L. Hammer, A. Kogan, Quasi-acyclic propositional Horn knowledge bases: optimal compression, IEEE Trans. Knowl. Data Eng. 7 (5) (1995) 751–762.
[31] E. Hemaspaandra, H. Schnoor, Minimization for generalized Boolean formulas, in: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI 2011), 2011, pp. 566–571.
[32] J. Lang, P. Liberatore, P. Marquis, Propositional independence — formula-variable independence and forgetting, J. Artif. Intell. Res. 18 (2003) 391–443.
[33] C.T. Lee, A completeness theorem and computer program for finding theorems derivable from given axioms, PhD thesis, Department of Electrical Engineering and Computer Science, University of California, 1967.
[34] P. Liberatore, Complexity issues in finding succinct solutions of PSPACE-complete problems, Technical Report abs/cs/0503043, CoRR, 2005.
[35] P. Liberatore, Redundancy in logic I: CNF propositional formulae, Artif. Intell. 163 (2) (2005) 203–232.
[36] P. Liberatore, Superredundancy: a tool for Boolean formula minimization complexity analysis, Electronic Colloquium on Computational Complexity, Technical Report 062, 2022.
[37] F. Lin, R. Reiter, Forget it!, in: Proceedings of the AAAI Fall Symposium on Relevance, 1994, pp. 154–159.
[38] D.W. Loveland, Part 1. Proof Theory, Princeton University Press, 2014, pp. 1–92.
[39] F. Martínez-Plumed, C. Ferri, J. Hernández-Orallo, M. Ramírez-Quintana, Knowledge acquisition with forgetting: an incremental and developmental setting, Adapt. Behav. 23 (5) (2015) 283–299.
[40] E.J. McCluskey, Minimization of Boolean functions, Bell Syst. Tech. J. 35 (6) (1956) 1417–1444.
[41] Y. Moinard, Forgetting literals with varying propositional symbols, J. Log. Comput. 17 (5) (2007) 955–982.
[42] N. Nikitina, S. Rudolph, (Non-)succinctness of uniform interpolants of general terminologies in the description logic EL, Artif. Intell. 215 (2014) 120–140.
[43] G. Nordh, B. Zanuttini, What makes propositional abduction tractable, Artif. Intell. 172 (10) (2008) 1245–1284.
[44] D. Rajaratnam, H.J. Levesque, M. Pagnucco, M. Thielscher, Forgetting in action, in: Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2014), AAAI Press, 2014.
[45] J. Robinson, A machine-oriented logic based on the resolution principle, J. ACM 12 (1) (1965) 23–41.
[46] J.R. Slagle, C.L. Chang, R. Lee, Completeness theorems for semantic resolution in consequence-finding, in: Proceedings of the First International Joint Conference on Artificial Intelligence (IJCAI'69), 1969, pp. 281–286.
[47] L.J. Stockmeyer, The polynomial-time hierarchy, Theor. Comput. Sci. 3 (1976) 1–22.
[48] S. Subbarayan, D.K. Pradhan, NiVER: non-increasing variable elimination resolution for preprocessing SAT instances, in: International Conference on Theory and Applications of Satisfiability Testing, Springer, 2004, pp. 276–291.
[49] M. Theobald, S.M. Nowick, T. Wu, Espresso-HF: a heuristic hazard-free minimizer for two-level logic, in: Proceedings of the Thirty-Third Design Automation Conference, 1996, pp. 71–76.
[50] C. Umans, The minimum equivalent DNF problem and shortest implicants, J. Comput. Syst. Sci. 63 (4) (2001) 597–611.
[51] H. van Ditmarsh, A. Herzig, J. Lang, P. Marquis, Introspective forgetting, Synthese 169 (2009) 809–827.
[52] K. Wang, A. Sattar, K. Su, A theory of forgetting in logic programming, in: Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005), AAAI Press/The MIT Press, 2005, pp. 682–688.
[53] Y. Wang, On forgetting in tractable propositional fragments, Technical Report 1502.02799, Computing Research Repository (CoRR), 2015.
[54] Y. Wang, Y. Zhang, Y. Zhou, M. Zhang, Knowledge forgetting in answer set programming, J. Artif. Intell. Res. 50 (2014) 31–70.

[55] X. Zhang, Forgetting for distance-based reasoning and repair in DL-lite, Knowl.-Based Syst. 107 (2016) 246–260.
[56] Y. Zhang, Y. Zhou, Knowledge forgetting: properties and applications, Artif. Intell. 173 (2009) 1525–1537.
[57] Y. Zhou, Polynomially bounded forgetting, in: Proceedings of the Thirteenth Pacific Rim International Conference on Artificial Intelligence (PRICAI 2014), 2014, pp. 422–434.
[58] Y. Zhou, Y. Zhang, Bounded forgetting, in: Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011), AAAI Press, 2011.