Reconstructing a single-head formula to facilitate logical forgetting

Paolo Liberatore

December 21, 2020 — 1:52

Abstract

Logical forgetting may take exponential time in general, but it does not when its input is a single-head propositional definite Horn formula. Single-head means that no variable is the head of multiple clauses. An algorithm to make a formula single-head if possible is shown. It improves over a previous one by being complete: it always finds a single-head formula equivalent to the given one if any.

1   Introduction

Logical forgetting is removing some variables from a logical formula while preserving its information regarding the others [LLM03]. Seen from a different angle, it is restricting a formula to keep only its information about some of its variables [Del17].

Each of the two interpretations has its own applications. Removing information allow reducing memory requirements [EKI19], simplifying reasoning [DW15, EF07, WSS05] and clarifying the relationship about variables [Del17]. Restricting a formula on some variables allows formalizing the limited knowledge of agents [FHMV95, RHPT14], ensuring privacy [GKLW17] and removing inconsistency [LM10].

Initially studied in first-order logic [LR94, ZZ11], logical forgetting has found its way into many logics such as propositional logic [Boo54, Moi07, Del17], answer set programming [WZZZ14, GKL16], description logics [KWW09, EIS+06], modal logics [vDHLM09], logics about actions [EF07, RHPT14], temporal logic [FAS+20], defeasible logic [AEW12] and belief revision [NCL07].

Contrary to intuition, forgetting may increase size instead of decreasing it [Lib20b]. This phenomenon is harmful to the many application scenarios where size reduction is important if not the sole aim of forgetting: increasing efficiency of reasoning [DW15, EF07, WSS05], summarizing, reusing and clarifying information [Del17], dealing with information overload [EKI19], tailoring information to specific applications [EIS+06], dealing with memory bounds [FHMV95] or more generally with limitations in the ability of reasoning [RHPT14].

To complicate matters, forgetting is not uniquely determined: a specific algorithm for forgetting may produce a large formula that is equivalent to a small one. The latter is as good as the former at representing information, but better in terms of size. The question is not whether forget increases size, but whether every possible mechanism for forgetting does.

1

Several equivalent formulae express forgetting the same variables from the same formula F. The question is whether one of them is smaller than F, or than a target size bound.

Giving an answer to this question is harder than the typical problems in propositional reasoning: it is in Σ_3^p and is D^{p}2-hard. The Horn restriction lowers complexity of one level, to membership in Σ_2^p and hardness in D^p [Lib20b].

Such a simplification in complexity is typical to the propositional Horn restriction [Mak87]. It is one of the reasons that makes it of interest, the other being its expressivity. Consistency, inference and equivalence only take polynomial time. At the same time, commonly-used statements like "if a, b and c then d" and "e, g, and f are impossible at the same time" are allowed. The Horn restriction is studied in, for example, description logic [KRH13], default reasoning [EL00], fuzzy logic [BV06], belief revision [ZP12], transaction logic [BK94] and nonmonotonic reasoning [Got92].

Restricting to the Horn case lowers the complexity of many operations. Is this the case for forgetting?

A way of forgetting a variable is to resolve all pairs of clauses over that variable and to remove the resolved clauses. The required space may be exponential in the number of variables to forget, even if the final result is polynomially large.

Something better can be done in the definite Horn case: replace the variable with the body of a clause with the variable in its head. This nondeterministic algorithm only requires polynomial working space [Lib20a]. It extends from the definite to the general Horn case [Lib20a].

A restriction that makes this method not only polynomial in space but also in time is the single-head condition. If clauses do not share heads, the choice of the clause to replace a variable is moot. What was in general nondeterministic becomes deterministic. Complexity decreases from nondeterministic polynomial time to deterministic polynomial time. The problem is tractable.

Whether a formula is single-head or not is easy to check by scanning the clauses and storing their heads; if one of them is found again, the formula is not single-head.

It is easy, but misses the semantical nature of forgetting.

Forgetting variables is restricting information to the other variables, and information is independent on the syntax. Forgetting the same variables from two equivalent formulae gives two equivalent formulae. The initial information is the same, what to forget is the same. What results should be the same, apart from its syntactic form. Forgetting c from $\{a \rightarrow b, b \rightarrow c, c \rightarrow d\}$ and forgetting it from $\{a \rightarrow b, b \rightarrow c, c \rightarrow d, a \rightarrow c\}$ produces equivalent formulae since the two formulae are equivalent. Yet, only the former is single-head. Only the former takes advantage of polynomiality in time. This needs not to be, since the result is the same. The second formula can be translated into the first, making forgetting polynomial.

Every formula that is equivalent to a single-head formula is amenable to this procedure: make it single-head, then forget.

The second step, forgetting, is easy since the formula is single-head. What about the first?

While checking whether a formula is single-head is straightforward, checking whether it is equivalent to a single-head formula is not. Many formulae are equivalent to a given one and are therefore to be checked for multiple heads. The problem would simplify if it amounted to checking a simple condition on the formula itself or its models rather than all its equivalent

formulae, but finding such a condition proved difficult [Lib20c].

A polynomial algorithm that often turns a formula in single-head form if possible is to select the minimal bodies for each given head according to a certain order. It is polynomial but incomplete: some single-head equivalent formulae are not recognized as such [Lib20c].

The present article restarts the quest from a different direction: instead of fixing the heads and searching for the bodies, it fixes the bodies and searches for the heads. The overall algorithm is complete: it turns a formula in single-head form if possible. The price is its running time, which may be exponential.

This may look discouraging since the target is an efficient algorithm for forgetting. Two points are still in favor of the new algorithm: first, it is exponential only when it works on equivalent sets of variables; second, each step can be stopped at any time, making the algorithm incomplete but fast.

The first point derives from the structure of the algorithm: it is a loop over the bodies of the formula; for each, its equivalent sets of variables are candidate bodies for clauses of the single-head form. The overall loop comprises only a polynomial number of steps. Each step is exponential because a formula may make a set of variables equivalent to exponentially many others. If not, the algorithm is polynomial in time.

The second point is a consequence of what each step is required to do: change some clauses of the original formula to make each head only occurs once. If such an endeavor turns out to be too hard, the original clauses can be just left unfiltered. Still better, some original clauses may be added to complete the best set found so far. This part of the output formula will not be single-head, but the rest may be. The forgetting algorithm may still benefit from the change, since it is exponential in the number of duplicated heads.

The article is organized as follows. Section 2 restates the definitions and results obtained in previous articles [Lib20a, Lib20c]. Section 3 presents the overall algorithm. It repeatedly calls a function implemented in Section 4. This completes the algorithm, which is correct and complete but not always polynomial-time. Given that a previous algorithm was incomplete but polynomial, the question is whether the problem itself is tractable; this is investigated in Section 5. The Python implementation of the program is described in Section 6.

2   Previously

A summary of the past work on single-head equivalence is in the Introduction. This section contains the technical definitions and results of previous articles [Lib20a, Lib20c] used in this article.

The first one is a quite obvious lemma that proves that entailing a variable is only possible via a clause that has that variable as its head [Lib20a].

Lemma 1 *If F is a definite Horn formula, the following three conditions are equivalent, where $P' \rightarrow x$ is not a tautology ($x \notin P'$).*

1. *$F \models P' \rightarrow x$;*

2. *$F^x \cup P' \models P$ where $P \rightarrow x \in F$;*

3. *$F \cup P' \models P$ where $P \rightarrow x \in F$.*

An order over the sets of variables [Lib20a] is central to the algorithm presented in the present article.

Definition 1 *The ordering $\leq_F$ that compares two sets of variables where F is a formula is defined by $A \leq_F B$ if and only if $F \models B \to A$.*

The set $BCN(B, F)$ contains all consequences of the set of variables B according to formula F [Lib20a].

Definition 2 *The set $BCN(B, F)$, where B is a set of variables and F is a formula, is defined as $BCN(B, F) = BCN(B, F) = \{x \mid F \cup B \models x\}$.*

The intended meaning of $BCN(B, F)$ is the set of consequences of B. Formally, $BCN(B, F)$ meets its aim. Informally, a consequence is something that follows from a premise. Some deduction is presumed involved in the process. Yet, $BCN(B, F)$ may contain variables that do not follow from B, they are just in B. No inference ever derives it from B.

Excluding all variables of B is not a correct solution: $BCN(B, F) \backslash B$ is too small. It does not include the variables of B that follow from B thanks for some derivation.

For example, both x and y are in $BCN(B, F)$ if $B = \{x, y\}$ and $F = \{y \to z, z \to y\}$. Yet, x is in $BCN(B, F)$ only because it is in B. Instead, y can be deduced from B via $y \to z$ and $z \to y$. This distinction proved important to the previous, incomplete algorithm for single-head equivalence [Lib20c].

Definition 3 *For every set of variables B and formula F, the* real consequences *of F are $RCN(B, F) = \{x \mid F \cup (BCN(B, F) \backslash \{x\}) \models x\}$.*

The set of all consequences of $B = \{x, y\}$ is $BCN(B, F) = \{x, y, z\}$. The difference between x and y is that x cannot be recovered once removed from it while y can. In other words, B derives something that derives y back; it does not for x. Therefore, y a real consequence, x is not: $RCN(B, F) = \{y, z\}$.

This example also shows how to determine $RCN(B, F)$ when F is a definite Horn formula: start from B and add to it all variables x such that $B' \to x \in F$ with $B' \subseteq B$. Only the variables that are added form $RCN(B, F)$. The algorithm is presented in full and proved correct in the same article where $RCN(B, F)$ is introduced [Lib20c]. The algorithm also determines the set of clauses used in the process: $\{B' \to x \in F \mid F \models B \to B'\}$. The variant of this set where tautologies are excluded is defined in the present article as $UCL(B, F)$. Since this set only contains the clauses of F, if F does not contain tautologies the difference disappears.

The only variables that may be consequences of a formula without being real consequences of it are the variables of B. This is formally stated by the following lemma [Lib20c].

Lemma 2 *For every formula F and set of variables B, it holds $BCN(B, F) = A \cup RCN(B, F)$.*

3   The reconstruction algorithm

A formula may have multiple clauses with the same head and still be equivalent to one that has not. Such a formula deceives the algorithm based on replacing heads with their bodies by sending it on a wild goose chase across multiple nondeterministic branches that eventually produce the same clause. Turning the formula into its single-head form wherever possible avoids such a waste of time.

The approximate algorithm shown in a previous article [Lib20c] produces a single-head formula that may be equivalent to F. It gives no guarantee, in general: it may fail even if the formula is single-head equivalent.

The algorithm presented in this section does not suffer from this drawback. When run on formula that has a single-head equivalent form, it finds it. The trade-off is efficiency: it does not always finish in polynomial time.

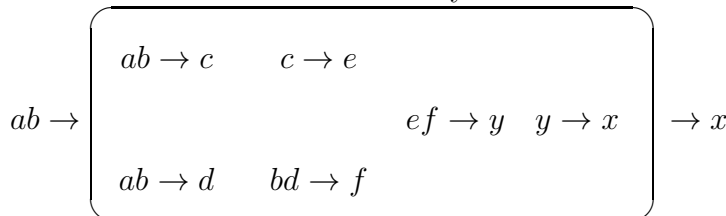## 3.1   Bird's-eye view of the reconstruction algorithm

The algorithm tries to reconstruct the formula clause by clause. It starts with an empty formula and adds clauses to it until it becomes equivalent to the input formula. Not adding clauses with the same head makes the resulting formula single-head.

An example tells which clauses are added. The aim is to make the formula under construction equivalent to the input formula. If the input formula contains $ab \rightarrow x$, the formula under construction has to entail it to achieve equivalence. This is the same as x being implied by $\{a, b\}$.

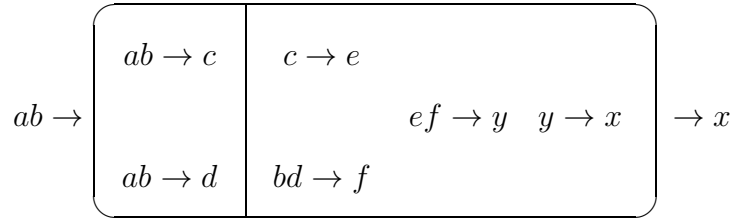$$ab \rightarrow \boxed{} \rightarrow x$$
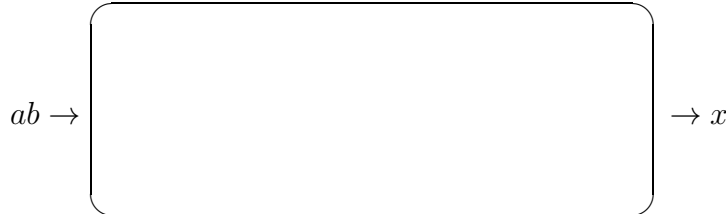
What is in the rectangle? The implication of x from $\{a, b\}$ is required in the formula under construction. This formula aims at replicating the input formula. It cannot contain clauses not entailed by that. The answer is that the rectangle contains clauses entailed by the input formula.

Since all clauses are definite Horn, this implication is realized by forward chaining: a and b derive some variables that derive some others that derive others that derive x. For example, a and b derive c and d, which derive e and f which derive y which derives x. Each individual derivation is realized by a clause.

$$ab \rightarrow \boxed{\begin{array}{llcc} ab \rightarrow c & c \rightarrow e & & \\ & & ef \rightarrow y & y \rightarrow x \\ ab \rightarrow d & bd \rightarrow f & & \end{array}} \rightarrow x$$

The first step of such a derivation has a and b imply another variable by means of a clause like $ab \rightarrow c$. Without such a clause, forward chaining does not even start. Nothing derives from $\{a, b\}$ without. Not even a single variable to derive others.

Some clauses in the rectangle are like $ab \to c$ and $ab \to d$, with $\{a, b\}$ as their body. Maybe not two, but at least one. Not zero. At least a clause in the rectangle has body $\{a, b\}$. Maybe a subset like $\{a\}$, but this case will be discussed later. For the moment, some clauses on the left edge of the rectangle have $\{a, b\}$ as their bodies.

$$ab \to \boxed{\begin{array}{c|c} ab \to c & c \to e \\[1em] & \qquad ef \to y \quad y \to x \qquad \\[1em] ab \to d & bd \to f \end{array}} \to x$$

A first stage contains $ab \to c$ and $ab \to d$; a second contains the following clauses used in the derivation of x. The key is induction. The formula under construction is assumed to already imply the clauses in the second stage. Adding $ab \to c$ and $ab \to d$ completes the derivation.

The inductive assumption is that the formula under construction implies all clauses in the second stage. The addition of $ab \to c$ and $ab \to d$ makes it imply $ab \to x$ as well. Such an induction is only possible if these two clauses are added only when the others are already implied. Adding them makes $ab \to x$ entailed. This entailment is the induction claim. Which is the induction assumption for the next step.

The induction step that makes $ab \to x$ entailed requires $c \to e$ to be entailed, which is only possible if the induction step that makes $c \to e$ entailed is in the past. The same for the other clauses in the second stage: when the clauses of body ab are added, all clauses of body c, bd, etc. are already. Key to induction is the order of addition of clauses.

How the single-head property is achieved is still untalked of. In the example, once $c \to e$ is in, all other sets of variables entailing e should do so by first entailing c. For example, if $ab \to e$ is in the input formula, it is entailed by the formula under construction thanks to $ab \to c$ and $c \to e$; no need for duplicated heads. If ab entailed e but not c, the input formula is not single-head equivalent.

All of this works flawlessly because of the simplicity of the input formula. Not in general.

Induction works because the algorithm tries to entail $ab \to x$ only when all clauses whose bodies are entailed by ab are already entailed. Such an assumption requires a caveat: these bodies are entailed by ab but do not entail it. It does not work on bodies that are not just entailed by ab but also equivalent to it. Extending the induction assumption to them makes it circular rather than inductive. On the example: if gh is equivalent to ab, the induction step for ab follows that of gh; but the step for gh follows that of ab for the same reason.

Equivalent sets cannot be worked on inductively. Even worse, they cannot be worked on separately as well. When the step that adds $ab \to x$ is over, the following steps rely on its inductive claim. A following step for the body ij may require $ab \to x$ to be entailed, if ij entails ab. Because of equivalence, ij also entails gh. The induction assumption also requires the clauses of body gh to be entailed. The clauses of head ab and gh to be found together.

This also tells what would happen if the first stage of derivation contained $a \to c$ instead of $ab \to c$, a strict subset of ab instead of ab itself. By monotonicity, a is entailed by ab; if it does not entail it, $a \to c$ is already in the formula under construction thanks to the inductive assumption. Otherwise, ab and a are equivalent. The induction step where $ab \to x$ is achieved also achieves $a \to c$.

Equivalent bodies also require special care to ensure the single-head property. Searching for the clauses of body ab and gh at the same time may result in two clauses such as $ab \rightarrow c$ and $gh \rightarrow c$. Such a solution may as well satisfy the inductive claim, but is not acceptable. As a previous article illustrates for pages and pages [Lib20c], just adding $gh \rightarrow c$ and hoping for equivalence to give $ab \rightarrow c$ does not work; this very clause $ab \rightarrow c$ may be necessary to ensure equivalence.

The algorithm in this section turns to a mysterious function to produce the necessary clauses of body ab—or equivalent. The next section unveils the mystery.

3.2   Worm's eye view of the reconstruction algorithm

The key to induction is the entailment between sets of variables. The required conditions are long to write: "a set variables entails another without being entailed by it". Sentences involving one of these segments may still be readable, but two or three make them unparsable. An order between sets used in a previous article [Lib20c] shortens wording: $A \leq_F B$ if $F \models B \rightarrow A$. If A entails B according to F but is not entailed by it, then $A <_F B$. If they entail each other, then $A \equiv_F B$.

The algorithm sorts the bodies of F according to this order, neglecting their heads. The minimal bodies are considered first. The set of clauses with these bodies are turned into single-head form. With this set fixed, the bodies that are minimal among the remaining ones are processed in the same way. The procedure ends when no body is left.

If A is the only minimal body, the first step finds all clauses $A \rightarrow x$ entailed by A. If the second-minimal body is B, the second step finds some clauses $B \rightarrow y$; not all of them, since some may be entailed thanks to the clauses produced in the first step. The same is done in the remaining steps.

No backtracking is required. The clauses generated by a step are in the final formula regardless of what happens in the following steps. This is not the reason why the algorithm may take exponential time.

The reason is that each step may take exponential time. When $A \equiv_F B$, the heads for A and B have to be generated in the same step. Sometimes this is easy, sometimes it is not. In the worst case, the possibilities increase exponentially with the number of equivalent sets.

Technically, the algorithm reconstructs the formula starting from the clauses of minimal bodies:

- the formula is initially empty;

- a body A that only entails equivalent bodies is chosen;

- enough clauses $P \rightarrow x$ with $P \equiv_F A$ are selected so that the reconstructed formula entails all clauses of the given formula with a body equivalent to A;

- a body B that only entails bodies entailed by A or equivalent to B is chosen;

- enough clauses $P \rightarrow x$ with $P \equiv_F B$ are selected so that the reconstructed formula entails all clauses of the given formula with a body equivalent to B;

- a body C that only entails bodies entailed by A and B or equivalent to C is chosen;

- ...

Since bodies are considered in increasing order, when a body B is considered all bodies $A <_F B$ have already been. This implies that all clauses $P \to x$ with $P \equiv_F A$ are already entailed by the formula under construction.

3.3 The construction lemma

The reconstruction algorithm looks like a typical backtracking mechanism: some clauses $P \to x$ with $P \equiv_F A$ are chosen; the same for B, C, and so on; choosing clauses for a following set D may prove impossible, showing that the choice for A was wrong. Backtracking to A is necessary.

It is not.

The reason is that each step generates enough clauses to entail all clauses with a body equivalent to the given one. Different choices may be possible, but the result always entails all these clauses. To entail all these clauses, their heads are required to be in the chosen clauses. Different choices differ in their bodies, but not in their heads and consequences. If a following choice proves impossible, backtracking is of no help since it can only replace a set of clauses with an equivalent one, with the same heads. The same impossible choice is faced again.

Lemma 3 *If F is a single-head formula such that $F \models A \to x$ with $x \notin A$ and $F \not\models A \to B$, then F does not contain $B \to x$.*

Proof. Since $F \models A \to x$ and $x \notin A$, Lemma 1 implies that F contains a clause $P \to x$ such that $F \models A \to P$. The set P differs from B since F entails $A \to P$ but not $A \to B$. As a result, $B \to x$ is not the same as $P \to x$. Since F contains the latter clause and is single-head, it does not contain the former. $\qquad\square$

Why does this lemma negate the need for backtracking? Backtracking kicks in when adding $B \to x$ is necessary but another clause $A \to x$ is already in the formula under construction. Because of the order of the steps, B is either strictly larger than A or incomparable to it. The condition $F \not\models A \to B$ holds in both. The lemma proves that $B \to x$ is not in any single-head formula equivalent to F. Adding this clause to the formula under construction is not only useless, it is impossible if that formula has to be single-head. Which clauses the step for A added is irrelevant.

3.4 One iteration of the reconstruction algorithm

An iteration of the reconstruction algorithm picks a body B only when all bodies $A <_F B$ have already been processed by a previous iteration. As a result, all clauses $A' \to x$ with $A' \equiv_F A$ are already entailed by the formula under construction. This is the fundamental invariant of the algorithm.

When looking at the iteration for a body B, the bodies A such that $A <_F B$ are called *strictly entailed bodies* to simplify wording. Since B is fixed, "all strictly entailed bodies do something" is simpler than "all bodies A such that $A <_F B$ do something".

When processing B, the formula under construction is the result of the previous iterations. The set A of a previous iteration cannot be greater or equal than B; therefore, either A is

incomparable with B or is less than it. If A is incomparable with B, then $F \not\models B \to A$; the clauses of body A or equivalent are not relevant to implications from B. If $A <_F B$ then $F \models B \to A$; the clauses of body A or equivalent to A help entailing variables from B. The construction lemma ensures that their heads x are not valid candidates as heads for B. The following definitions express these ideas.

Definition 4 *Given a formula F, the* strictly entailed bodies *of B are the sets A such that $A \to x \in F$ for some x and $A <_F B$. The clauses and free heads of such bodies are:*

clauses of strictly entailed bodies:

$$SCL(B, F) = \{A \to x \mid x \notin A, \ F \models A \to x, \ F \models B \to A, \ F \not\models A \to B\}$$

heads free from strictly entailed bodies:

$$SFREE(B, F) = \{x \mid \not\exists A \to x \in SCL(B, F)\}$$

The iteration for a body B produces some clauses $B' \to x$ with $B' \equiv B$. The construction lemma limits the choices of the heads to $x \in SFREE(B, F)$. All clauses of $SCL(B, F)$ are already entailed by the formula under construction. They are the only ones relevant to finding new clauses $B' \to x$. The goal of the iteration is to entail the following clauses.

Definition 5 *Given a set of variables B and a formula F, the* clauses of equivalent bodies *are:*

$$BCL(B, F) = \{A \to x \mid x \notin A, F \models A \to x, \ F \models B \to A, \ F \models A \to B\}$$

The algorithm progressively builds a formula equivalent to the given one. It is initially empty. Every step hinges around the body B of a clause. The main invariant is that the formula under construction already entails $SCL(B, F)$; the aim is to make it entail $BCL(B, F)$ as well. This makes the invariant true again in the following steps, since $BCL(B, F)$ is part of $SCL(C, F)$ if $B <_F C$. The $SFREE(B, F)$ set simplifies this task by restricting the possible heads of the new clauses.

Every step adds some new clauses to the formula; which ones they are is explained in a later section. This one is about the overall algorithm. To it, what matters of a single step is only what it produces, not how. What it produces is formalized as follows.

Definition 6 *A* valid iteration *is a function $ITERATION(B, F)$ that satisfies the following two conditions for every set of variables B and formula F:*

choice: *$ITERATION(B, F)$ is a set of non-tautological clauses $B' \to x$ such that*

- $F \models B' \to x$,
- $F \models B \equiv B'$ *and*
- $x \in SFREE(B, F)$;

entailment: $SCL(B, F) \cup ITERATION(B, F) \models BCL(B, F)$.

The first condition (choice) limits the possible clauses $B' \to x$ generated at each step. The second (entailment) requires them to satisfy the invariant for the next steps (entailment of $BCL(B, F)$) given the invariant for the current (entailment of $SCL(B, F)$).

A minimal requirement for the steps of the algorithm is that they succeed when the formula is already single-head. In spite of its weakness, this property is useful for proving that the algorithm works on single-head equivalent formulae.

Lemma 4 *If F is a single-head formula, $ITERATION(B, F) = \{B' \to x \in F \mid x \notin B', \ F \models B \equiv B'\}$ is a valid iteration function.*

Proof. The given function $ITERATION(B, F)$ is shown to satisfy both the choice and the entailment condition, the two parts of the definition of a valid iteration function (Definition 6).

The choice condition concerns the individual clauses of $ITERATION(B, F)$. They are not tautological by construction. The other requirements are proved one by one for each $B' \to x \in ITERATION(B, F)$.

- The first is $F \models B' \to x$. It holds because $B' \to x \in F$ is part of the definition of this function $ITERATION(B, F)$.

- The second is $F \models B \equiv B'$. It is part of the definition of this function $ITERATION(B, F)$.

- The third is $x \in SFREE(B, F)$. Its converse $x \notin SFREE(B, F)$ means that $SCL(B, F)$ contains a clause $A \to x$. The definition of $A \to x \in SCL(B, F)$ is $x \notin A$, $F \models A \to x$, $F \models B \to A$ and $F \not\models A \to B$. The latter implies $F \not\models A \to B'$ since $F \models B \equiv B'$. Lemma 3 proves that $F \models A \to x$, $x \notin A$ and $F \not\models A \to B'$ imply $B' \to x \notin F$, which contradicts $B' \to x \in F$.

The given function $ITERATION(B, F)$ also satisfies the entailment condition of Definition 6: $SCL(B, F) \cup ITERATION(B, F) \models BCL(B, F)$. This is implied by $SCL(B, F) \cup ITERATION(B, F) \models SCL(B, F) \cup BCL(B, F)$, which is defined by $SCL(B, F) \cup ITERATION(B, F) \models B' \to x$ if $x \notin B'$, $F \models B' \to x$, $F \models B \to B'$ and either $F \not\models B' \to B$ or $F \models B' \to B$. Either one choice of alternative is always true, which simplifies the condition to $SCL(B, F) \cup ITERATION(B, F) \models B' \to x$ if $x \notin B'$, $F \models B' \to x$ and $F \models B \to B'$. What is actually proved is a more general condition: for every $G \subseteq F$, $SCL(B, F) \cup ITERATION(B, F) \models B' \to x$ holds if $x \notin B'$, $G \models B' \to x$ and $F \models B \to B'$. The introduction of G allows for induction over the size of G.

If G is empty, it entails $B' \to x$ only if $x \in B'$. Such a clause does not satisfy the other requirement $x \notin B'$. The condition is met. This is the base case of induction.

The induction step requires proving the claim for an arbitrary $G \subseteq F$ assuming it for every formula smaller than G.

The claim is that $G \subseteq F$, $x \notin B'$, $G \models B' \to x$ and $F \models B \to B'$ imply $SCL(B, F) \cup ITERATION(B, F) \models B' \to x$. The induction assumption is the same for every $G' \subset G$.

The premises $x \notin B'$ and $G \models B' \to x$ allow applying Lemma 1: $G^x \models B' \to B''$ holds for some clause $B'' \to x \in G$.

A consequence of $B'' \to x \in G$ is $B'' \to x \in F$ since $G \subseteq F$. For the same reason, $G^x \models B' \to B''$ implies $F \models B' \to B''$. With the other assumption $F \models B \to B'$,

transitivity implies $F \models B \to B''$. The converse implication $B'' \to B$ may be entailed by F or not. If $F \models B'' \to B$ holds then either x is in B'' or $B'' \to x$ is in $ITERATION(B, F)$; in both cases, $B'' \to x$ is entailed by $SCL(B, F) \cup ITERATION(B, F)$. If $F \not\models B'' \to B$ then either x is in B'' or $B'' \to x$ is in $SCL(B, F)$; in both cases, $B'' \to x$ is entailed by $SCL(B, F) \cup ITERATION(B, F)$. The conclusion of this paragraph is $SCL(B, F) \cup ITERATION(B, F) \models B'' \to x$.

Induction proves that $SCL(B, F) \cup ITERATION(B, F)$ also entails $B' \to B''$. This condition is the same as $SCL(B, F) \cup ITERATION(B, F) \models B' \to b$ for every $b \in B''$. Since G^x entails $B' \to B''$, it entails $B' \to b$ for every $b \in B''$. If $b \in B'$ then $B' \to b$ is a tautology and is therefore entailed by $SCL(B, F) \cup ITERATION(B, F)$. Otherwise, $b \notin B'$. Since $B'' \to x$ contains x, it is not in G^x; since G contains $B'' \to x$ instead, G^x is a proper subset of G. The induction assumption is the same as the induction claim with G^x in place of G, that is, $G^x \subseteq F$, $x \notin B'$, $G^x \models B' \to b$ and $F \models B \to B'$ imply $SCL(B, F) \cup ITERATION(B, F) \models B' \to b$. Since all premises hold, the claim holds as well: $SCL(B, F) \cup ITERATION(B, F) \models B' \to b$. This is the case for every $b \in B''$. A consequence is $SCL(B, F) \cup ITERATION(B, F) \models B' \to B''$.

The conclusion $SCL(B, F) \cup ITERATION(B, F) \models B' \to B''$ joins $SCL(B, F) \cup ITERATION(B, F) \models B'' \to x$ to prove $SCL(B, F) \cup ITERATION(B, F) \models B' \to x$. This is the induction claim. □

The specific function $ITERATION(B, F)$ of this lemma gives only clauses of F. It depends on the syntax of F. Yet, the validity of an iteration function is semantical: only depends on entailments, equivalences and the three functions $SCL(B, F)$, $SFREE(B, F)$ and $BCL(B, F)$, which are all defined in terms of entailments. Therefore, if $ITERATION(B, F)$ is valid, the same function applied to another formula $ITERATION(B, F')$ is still valid if $F \equiv F'$.

Lemma 5 *If $ITERATION(B, F)$ is a valid iteration function and $F \equiv F'$, then $ITERATION(B, F') = ITERATION(B, F)$ is a valid iteration function.*

Proof. Since $ITERATION(B, F)$ is valid iteration function, it produces a set of clauses $B' \to x$ such that $x \notin B'$, $F \models B \to x$, $F \models B \equiv B'$, $x \in SFREE(B, F)$ and all of them plus $SCL(B, F)$ entail all clauses $B' \to x$ with $F \models B \equiv B'$ that are entailed by F. In turn, $SCL(B, F)$ is defined as the set of clauses $B'' \to x$ such that $F \models B'' \to x$, $F \models B \to B''$ and $F \not\models B'' \to x$. Finally, $SFREE(B, F)$ is defined as the set of variables x such that no clause $B'' \to x \in SCL(B, F)$.

All these conditions are based on what F entails. Since F is equivalent to F', they also hold when F' is in place of F. Therefore, $ITERATION(B, F') = ITERATION(B, F)$ is also a valid iteration function. □

The two lemmas together prove that if F is single-head equivalent then it has a valid iteration function: the one for a single-head formula that is equivalent to F.

Lemma 6 *If F is equivalent to the single-head formula F', then $ITERATION(B, F) = \{B' \to x \in F' \mid F' \models B \equiv B'\}$ is a valid iteration function.*

Proof. By Lemma 4, $ITERATION(B, F')$ is a valid iteration function for F'. By Lemma 5, it is also valid for F. □

As a result, every single-head equivalent formula has a valid iteration function.

Corollary 1 *Every single-head equivalent formula has a valid iteration function.*

The proofs of the three lemmas show how to find a valid iteration function: switch from the single-head equivalent formula to a single-head formula that is equivalent to it and then extract the necessary clauses from it.

This mechanism works in theory, and is indeed used in proofs that requires a valid iteration function to exists. It does not work in practice, since the single-head formula equivalent to the given one is not known. Rather the opposite: the whole point of valid iteration functions is to be used in reconstruction algorithm to find a single-head formula equivalent to the given one. The iteration function is a means to the aim of finding a single-head formula, not the other way around.

In practice, finding a valid iteration function is no easy task. An entire following section is devoted to that. For the moment, a valid iteration function is assumed available.

3.5   The reconstruction algorithm

Assuming the availability of a valid iteration function $ITERATION(B, F)$, the reconstruction algorithm is as follows.

Algorithm 1 (Reconstruction algorithm)   *1. $G = \emptyset$*

> *2. $P = P(X)$ # set of all subsets of X*
>
> *3. while $P \neq \emptyset$:*
>
> > *(a) choose a $\leq_F$-minimal $B \in P$*
> > *(a set such that $A <_F B$ does not hold for any $A \in P$)*
> > *(b) $G = G \cup ITERATION(B, F)$*
> > *(c) $P = P \backslash \{B' \mid F \models B \equiv B'\}$*
>
> *4. return G*

The algorithm is non-deterministic, but works for every possible nondeterministic choices. It is not deterministic because many sets B may be minimal each time. The final result depends on these choices because $ITERATION(B, F)$ may differ from $ITERATION(B', F)$ even if $B \equiv_F B'$. Yet, an equivalent single-head formula is found anyway, if any.

As an example, a valid iteration function for $F = \{a \rightarrow b, b \rightarrow a, bc \rightarrow d\}$ has the values $ITERATION(\{a, c\}, F) = \{bc \rightarrow d\}$ and $ITERATION(\{b, c\}, F) = \{ac \rightarrow d\}$. The reconstruction algorithm produces either $bc \rightarrow d$ or $ac \rightarrow d$ depending on whether it chooses $\{a, c\}$ or $\{b, c\}$ as the set B of an iteration of its main loop. Either way, the result is single-head and equivalent to F.

The following lemmas state some properties of the algorithm that are independent on the nondeterministic choices. For example, proving termination means that the algorithm terminates for all possible nondeterministic choices.

Lemma 7 *The reconstruction algorithm (Algorithm 1) always terminates.*

Proof. The set P becomes smaller at each iteration, since a choice of B is always possible if P is not empty because $\leq_F$ is a partial order. As a result, at some point P becomes empty and the algorithm terminates. $\square$

The second property of the reconstruction algorithm is that it never calculates $ITERATION(B, F)$ for equivalent sets B. Still better, it calculates $ITERATION(B, F)$ exactly once for each B among its equivalent ones. This is proved by the following two lemmas.

Lemma 8 *For every set of variables B', the reconstruction algorithm (Algorithm 1) calculates $ITERATION(B, F)$ for some B such that $F \models B \equiv B'$.*

Proof. Initially, $B' \in P$ since P contains all sets of variables. The algorithm terminates by Lemma 7. It only terminates when P is empty. Therefore, B' is removed from P during the run of the algorithm. Removal is only done by $P = P \backslash \{B' \mid F \models B \equiv B'\}$. This operation is preceded by $G = G \cup ITERATION(B, F)$. $\square$

Lemma 9 *The reconstruction algorithm (Algorithm 1) does not calculate both $ITERATION(B, F)$ and $ITERATION(B', F)$ if $F \models B \equiv B'$.*

Proof. Let B and B' be two sets such that the reconstruction algorithm calculates $ITERATION(B, F)$ and $ITERATION(B', F)$. This implies that the algorithm chooses B in an iteration and B' in another. By symmetry, the iteration of B can be assumed to come first. This iteration ends with $P = P \backslash \{B' \mid F \models B \equiv B'\}$. When the iteration of B' begins, no set equivalent to B according to F is in P any longer. Since B' is in P, it is not equivalent to B. This is a contradiction. $\square$

When the reconstruction algorithm extracts a set B from P, all sets A with $A <_F B$ have been removed in a past iteration. During that, G was increased to entail all clauses $A \rightarrow x$ entailed by F, which make $SCL(B, F)$ because of $A <_F B$. That G entails $SCL(B, F)$ is the fundamental invariant of the algorithm.

Lemma 10 *If $ITERATION(B, F)$ is a valid iteration function, $G \models SCL(B, F)$ holds during the whole execution of the reconstruction algorithm (Algorithm 1).*

Proof. The proof links the values of the variables at different iterations. The values of P, B and G at the i-th iteration of the algorithm are denoted P_i, B_i and G_i. More precisely, these are the values right after Step 3a of the i-th iteration.

The claim $G \models SCL(B, F)$ is proved by induction over the number of iterations of the loop.

In the first iteration, P_1 contains all sets of variables, G_1 is the empty set and B_1 is a set of P_1 such that P_1 contains no set $A <_F B_1$. The values of P_1 and B_1 imply that no set such that $A <_F B_1$ exists. As a result, $SCL(B_1, F)$ is empty and therefore entailed even by the empty formula G_1.

The induction claim is $G_i \models SCL(B_i, F)$; the induction assumption is $G_j \models SCL(B_j, F)$ for every $j < i$.

The claim $G_i \models SCL(B_i, F)$ is $G_i \models A \to x$ if $x \notin A$, $F \models A \to x$ and $A <_F B$. The set B_i is an element of P_i such that P_i contains no element that is strictly less than B_i. Since A is strictly less than B_i, it is not in P_i. Going back to the first iteration, P_1 contains all sets of variables, including A. Therefore, A is removed at some iteration $j < i$. The only removal instruction is $P_{j+1} = P_j \backslash \{B' \mid F \models B_j \equiv B'\}$. It removes A; therefore, $F \models B_j \equiv A$ holds. With the assumptions that $A \to x$ is not tautologic and is entailed by F, it proves that $A \to x$ belongs to $BCL(B_j, F)$.

The induction assumption is $G_j \models SCL(B_j, F)$. A consequence is that $G_j \cup ITERATION(B_j, F)$ entails $SCL(B_j, F) \cup ITERATION(B_j, F)$. By definition of a valid iteration function, the latter entails $BCL(B_j, F)$, which includes $A \to x$. This clause is therefore entailed by $G_j \cup ITERATION(B_j, F)$, the same as G_{j+1}.

Since $j < i$ and G monotonically increases, $G_{j+1} \subseteq G_i$. As a result, $G_i \models A \to x$, which is the claim. $\square$

The reconstruction algorithm generates a formula that is equivalent to F. This holds for every valid iteration function and nondeterministic choices.

Lemma 11 *If $ITERATION(B, F)$ is a valid iteration function, the formula produced by the reconstruction algorithm is equivalent to F.*

Proof. The algorithm returns G, which is only added clauses of $ITERATION(B, F)$. All of them are entailed by F by the definition of valid iteration functions. This proves that the formula returned by the algorithm contains only clauses entailed by F.

It is now proved to imply all clauses $B' \to x$ entailed by F. By Lemma 8, the reconstruction algorithm calculates $ITERATION(B, F)$ for some B such that $F \models B \equiv B'$. A consequence of the invariant $G \models SCL(B, F)$ proved by Lemma 10 is that $G \cup ITERATION(B, F)$ entails $SCL(B, F) \cup ITERATION(B, F)$, which entails $B' \to x$ by the definition of valid iteration. By transitivity, $G \cup ITERATION(B, F)$ entails $B' \to x$. After $G = G \cup ITERATION(B, F)$, it is G that entails $B' \to x$. Since G is never removed clauses, at the end of the algorithm G still entails $B' \to x$. $\square$

The algorithm is correct: it always terminates and always outputs a formula equivalent to the given one. What is missing is its completeness: if the input formula is equivalent to a single-head formula, that formula is output for some valid iteration function.

Lemma 12 *If F is equivalent to a single-head formula F', for some valid iteration function the reconstruction algorithm (Algorithm 1) returns F'.*

Proof. By Lemma 6, $ITERATION(B, F) = \{B' \to x \in F' \mid F' \models B \equiv B'\}$ is a valid iteration. The algorithm always terminates thanks to Lemma 11. It returns only clauses from $ITERATION(B, F)$; since these are all clauses of F', the return value is a subset of F'.

The claim is proved by showing that all clauses of F' are returned. Let $B' \to x \in F'$. By Lemma 8, the algorithm calculates $ITERATION(B, F)$ for some B such that $F \models B \equiv B'$. By construction, $B' \to x$ is in $ITERATION(B, F)$. The algorithm returns the union of all sets $ITERATION(B, F)$ it calculates, and $B' \to x$ is in this union. $\square$

This lemma tells that every single-head formula equivalent to the input can be returned by the reconstruction algorithm using the appropriate iteration function. Yet, an inappropriate

iteration function may make it return a formula that is still equivalent to the original but is not single-head. An additional property of the iteration function is required to avoid such an outcome.

Lemma 13 *If $ITERATION(B, F)$ is a valid iteration function whose values never contain each two clauses with the same head then the reconstruction algorithm (Algorithm 1) returns a single-head formula if and only if F is equivalent to a single-head formula.*

Proof. Let F' be the formula returned by the reconstruction algorithm on F. By Lemma 11, $F \equiv F'$.

If F' is single-head, then F is single-head equivalent by definition.

Otherwise, F' is not single-head. The following proof shows that F is equivalent to no single-head formula.

Let $B \to x$ and $C \to x$ be two clauses returned by the algorithm. They were added to G in different iterations, since by assumption $ITERATION(B, F)$ does not contain two clauses with the same head. Let B' and C' be the values of B in the iterations that produce them:

$$B \to x \ \in \ ITERATION(B', F) \text{ where } F \models B \equiv B'$$
$$C \to x \ \in \ ITERATION(C', F) \text{ where } F \models C \equiv C'$$

By Lemma 9, the reconstruction algorithm does not determine both $ITERATION(B', F)$ and $ITERATION(C', F)$ if $F \models B' \equiv C'$. Therefore, $F \not\models B \equiv C$.

Let F' be a single-head formula that is equivalent to F, and $A \to x$ be its only clause with head x. Because of its equivalence with F, it entails both $B \to x$ and $C \to x$. Since these clauses are produced by a valid iteration function, they are not tautologic: $x \notin B$ and $x \notin C$. Lemma 1 tells that F' contains $B'' \to x$ and $C'' \to x$ such that $F' \models B \to B''$ and $F' \models C \to C''$. Since the only clause of F' that has x in the head is $A \to x$, both B'' and C'' coincide with A. As a result, F' entails $B \to A$ and $C \to A$.

Since F is equivalent to F', it also does. This is the same as $A \leq_F B$ and $A \leq_F C$. These orderings are not strict. Otherwise, $A <_F B$ implies $A <_F B'$, which implies $A \to x \in SCL(B', F)$, which implies $x \notin SFREE(B', F)$, which contradicts $B \to x \in ITERATION(B', F)$ since the latter includes $x \in SFREE(B', F)$. The same applies to $A <_F C$. As a result, $F \models A \equiv B$ and $F \models A \equiv C$. The consequence $F \models B \equiv C$ contradicts $F \not\models B \equiv C$, which was previously proved. □

This lemma completes the proof of soundness and correctness of the reconstruction algorithm: it terminates with a single-head formula, if any; if any exists, it returns it.

3.6   Efficiency of the reconstruction algorithm

The reconstruction algorithm works flawlessly in theory, but in practice its iterations may be exponentially many because so many are the sets of variables. A small change solves the problem: instead of all sets of variables, only process the bodies of clauses of the formula. The algorithm produces the same result because it runs like using a tie-breaking rule: if more than one set is minimal, prefer one such that $ITERATION(B, F) = \emptyset$; if none is such, choose a set that is the body of a clause of F. The following two lemmas show this rule valid.

Lemma 14 *If $ITERATION(B, F) \neq \emptyset$ then $F \models B \equiv B''$ holds for some clause $B'' \to x \in F$.*

Proof. Let $B' \to x$ be a clause of $ITERATION(B, F)$. The definition of valid iteration includes $x \notin B'$, $F \models B' \equiv B$, $F \models B' \to x$ and $x \in SFREE(B, F)$.

From $F \models B' \to x$ and $x \notin B'$ Lemma 1 implies that F contains a clause $B'' \to x$ such that $F \models B' \to B''$. The converse implication $F \models B'' \to B'$ may hold or not. If it does, then F contains a clause $B'' \to x$ with $F \models B' \equiv B''$. Since $F \models B \equiv B'$, it follows $F \models B \equiv B''$ and the claim is proved.

The other case $F \not\models B'' \to B'$ is shown to lead to contradiction. Since $F \models B \equiv B'$, the conditions $F \not\models B'' \to B'$ and $F \models B' \to B''$ extend from B' to B. This change turns them into $F \not\models B'' \to B$ and $F \models B \to B''$. This is the definition of $B'' <_F B$. Since $B'' \to x$ is in F, it is not a tautology and is entailed by F. With $B'' <_F B$, these conditions define $B'' \to x \in SCL(B, F)$. This implies $x \notin SFREE(B, F)$, which conflicts with $x \in SFREE(B, F)$. □

The choice of the minimal set B in the construction algorithm is non-deterministic. All lemmas proved so far hold for every possible nondeterministic choices. Therefore, they also hold when restricting the choices in whichever way. The following lemma tells that the candidate sets are of two kinds: those which would generate an empty set of clauses, and those equivalent to the body of a clause in the original formula.

Lemma 15 *At every step of the reconstruction algorithm (Algorithm 1) if P is not empty it contains either a $\leq_F$-minimal set B such that $ITERATION(B, F) = \emptyset$ or a $\leq_F$-minimal set B such that $B \to x \in F$.*

Proof. Since P is not empty, it has at least a minimal element B because $\leq_F$ is a partial order. If $ITERATION(B, F)$ is empty, the claim is proved. Otherwise, $ITERATION(B, F) \neq \emptyset$, and Lemma 14 applies: F contains a clause $B'' \to x$ such that $F \models B \equiv B''$. Its body B'' is $<_F$-minimal in P because it is equivalent to B, which is minimal. Remains to prove that B'' is in P.

By contradiction, $B'' \notin P$ is assumed. Since P initially contains all sets of variables, B'' was removed at some point. The only operation removing elements from P removes all sets equivalent to a given one. Since it removes B'' it also removes B since it is equivalent to B''. This contradicts the assumption that B is in P. □

If $ITERATION(B, F)$ is empty the iteration that chooses B is irrelevant to the final result since G remains the same. The sets equivalent to B could just be removed from P. The other case is that B is equivalent to a body B'' of a clause of F. Since B is minimal, also B'' is minimal. Therefore, the algorithm could as well choose B'' instead of B. This means that the algorithm may be limited to choosing sets of variables that are bodies of clauses of F. The other sets either do not change G or can be replaced by them.

Algorithm 2 (Precondition-reconstruction algorithm)   *1. $G = \emptyset$*

2. $P = \{B \mid B \to x \in F$ for some variable $x\}$

3. while $P \neq \emptyset$:

(a) choose a $<_F$-minimal $B \in P$
   (a set such that $A <_F B$ does not hold for any $A \in P$)

(b) $G = G \cup ITERATION(B, F)$

(c) $P = P \backslash \{B' \mid F \models B \equiv B'\}$

4. return G

The algorithm is almost the same as the basic reconstruction algorithm (Algorithm 1). The only difference is that P is no longer initialized with all sets of variables, but only with the bodies of the clauses of F. Its initial size is reduced from exponential to linear. Since each iterations removes at least B, it makes P strictly decrease. They are therefore linearly many. The algorithm is still not polynomial-time since $ITERATION(B, F)$ may not be polynomial.

This new, efficient version of the reconstruction algorithm is like the original one with a restriction of the nondeterministic choices: if $ITERATION(B, F) = \emptyset$ holds for some minimal sets B of P, one of them is chosen; otherwise, the body of a clause of F is chosen. Lemma 15 guarantees that P always contains a minimal set of either kind.

The difference is that the original algorithm removes all sets B' such that $F \models B \equiv B'$ when $ITERATION(B, F) = \emptyset$; the new algorithm virtually removes B only. The following lemma shows that the other sets B' also satisfy $ITERATION(B', F) = \emptyset$, and are therefore removed right after B.

Lemma 16 *If $ITERATION(B, F) = \emptyset$ and $F \models B \equiv B'$ then $ITERATION(B', F) = \emptyset$.*

Proof. The proof shows contradictory the existence of a clause $B'' \rightarrow x$ in $ITERATION(B', F)$. The definition of a valid iteration function includes $x \notin B''$, $F \models B'' \rightarrow x$, $F \models B \equiv B''$ and $x \in SFREE(B, F)$. The first three conditions define $B'' \rightarrow x \in BCL(B', F)$.

Since $BCL(B', F)$ and $SFREE(B', F)$ are defined from what F entails, they coincide with $BCL(B, F)$ and $SFREE(B, F)$ since $F \models B \equiv B'$. This implies $B'' \rightarrow x \in BCL(B, F)$ and $x \in SFREE(B, F)$.

The definition of $ITERATION(B, F)$ includes $SCL(B, F) \cup ITERATION(B, F) \models BCL(B, F)$. Since $ITERATION(B, F)$ is empty, this is the same as $SCL(B, F) \models BCL(B, F)$. As a result, $SCL(B, F)$ implies $B'' \rightarrow x$ while not containing any positive occurrence of x since $x \in SFREE(B, F)$. This is only possible if $B'' \rightarrow x$ is tautologic, but $x \in B''$ contradicts $x \notin B''$. $\square$

The restriction of the nondeterministic choices does not affect the lemmas about Algorithm 1 proved so far. While not explicitly mentioned, they do not assume anything about the nondeterministic choices. As a result, they hold regardless of them. They hold even restricting the nondeterministic choice in whichever way.

As a result, the new algorithm always terminates, always returns a formula equivalent to the given one, and that formula is single-head if and only if any is equivalent to the original, provided that $ITERATION(B, F)$ does not contain two clauses of the same head for any B.

The modified reconstruction algorithm performs the loop a number of times equal to the number of clauses of the input formula. More precisely, to the number of different bodies in the formula; still, a linear number of times. If each iteration turns out to be polynomial, establishing single-head equivalence would be proved polynomial.

The problem is indeed in $ITERATION(B, F)$. A brute-force implementation is to loop over all sets of clauses $B' \rightarrow x$ with $B' \equiv_F B$ and $x \in SFREE(B, F)$, checking if one satisfies the conditions of validity. Unfortunately, the sets B' may be exponentially many. This obvious solution is unfeasible.

Whether $ITERATION(B, F)$ can be determined in polynomial time is an open problem. The following section gives an algorithm that may run in polynomial time but is exponential-time in the worst case.

3.7   Formulae that are not single-head equivalent

All is good when the formula is single-head equivalent: an appropriate valid iteration function makes the reconstruction algorithm return a single-head formula that is equivalent to the given one. This is what happens when the formula is single-head equivalent. What if it is not?

An easy and wrong answer is that the algorithm returns a formula that is not single-head. A closer look at the lemmas shows that this is not always the case.

Lemma 11 tells that the reconstruction algorithm returns a formula that is equivalent to the given one, but has a requirement: a valid iteration function. That one exists is only guaranteed for single-head formulae, not in general. If the formula is not single-head equivalent then either:
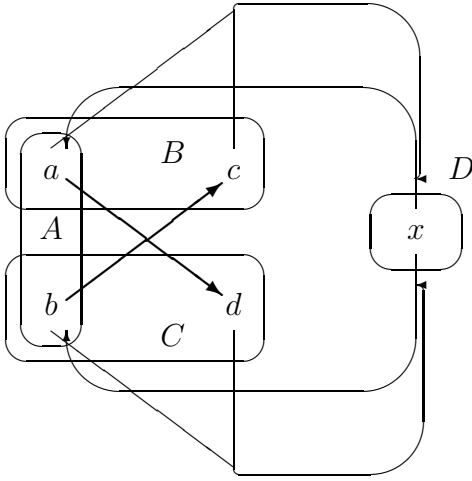
- no valid iteration function exists;

- a valid iteration function exists; it makes the algorithm generate a formula equivalent to the given one by Lemma 11 and therefore not single-head.

Lemma 11 splits the second case in two: either the iteration function has some values containing two clauses with the same head or it does not. All three cases are possible.

An example of a formula that has no valid iteration function is $F = \{a \rightarrow b, b \rightarrow c, c \rightarrow b\}$, in the following figure and in the `inloop.py` test file of the `singlehead.py` program. While $ITERATION(\{b\}, F)$ and $ITERATION(\{c\}, F)$ satisfy the definition of validity if equal to $\{b \rightarrow c, c \rightarrow b\}$, no value for $ITERATION(\{a\}, F)$ does. Indeed, $SCL(\{a\}, F) = \{b \rightarrow c, c \rightarrow b\}$, which implies $SFREE(\{a\}, F) = \{a\}$. As a result, $ITERATION(\{a\}, F)$ could only be empty. This is not possible because $BCL(\{a\}, F)$ contains $a \rightarrow c$ and $b \rightarrow c$, which are not entailed by $SCL(\{a\}, F)$.
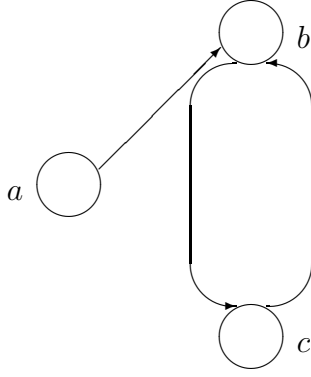
An example of a formula that has a valid iteration function but all of them have values with two clauses with the same head is $F = \{x \to a, a \to d, x \to b, b \to c, ac \to x, bd \to x\}$. This formula is in the `disjointnotsingle.py` test file of the `singlehead.py` program. The following figure shows it. A valid iteration function is $ITERATION(\{a\}, F) = \{a \to d\}$, $ITERATION(\{b\}, F) = \{b \to c\}$, $ITERATION(\{x\}, F) = ITERATION(\{a, b\}, F) = ITERATION(\{a, c\}, F) = ITERATION(\{d, b\}, F) = \{x \to a, x \to b, ac \to x, bd \to x\}$, but the last values contain two clauses with the same head x. This is always the case. In particular, if $B = \{x\}$ then $ITERATION(B, F)$ is required to entail $x \to a$, $x \to b$, $ac \to x$ and $bd \to x$, but $SCL(B, F) = \{a \to d, b \to c\}$ does not help at all. Three heads x, a and b do not suffice for four bodies.

An example of a formula that is not single-head equivalent but has a valid iteration function that never returns a set containing two clauses with the same head is $F = \{a \to c, b \to c\}$, in the following figure and in the `samehead.py` test file of the `singlehead.py` program. Its only valid iteration function is defined by $ITERATION(\{a\}, F) = \{a \to c\}$ and $ITERATION(\{b\}, F) = \{b \to c\}$. The reconstruction algorithm either processes $\{a\}$ first and $\{b\}$ second or vice versa. In both cases it adds both $ITERATION(\{a\}, F)$ and $ITERATION(\{b\}, F)$ to the resulting formula, which ends up containing two clauses with head c.
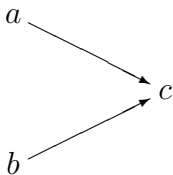
The algorithm as implemented in the `singlehead.py` program does not deal with these

three cases separately. It needs not. The optimizations in the search for an iteration function turn all three cases into the first, where no valid iteration function exists.

The reason is that $ITERATION(B, F)$ is not calculated in advance, but during the iteration for B. A number of candidate sets of clauses are tried. Excluding the ones with duplicated heads is actually a simplification. So is excluding the ones that contain some heads of another value $ITERATION(A, F)$ calculated before, which happens in two cases: when no valid iteration function exists and when the output formula would not be single-head.

4 The iteration function

The reconstruction algorithm uses a function $ITERATION(B, F)$ at each iteration. Only its properties have been given so far. If F is single-head equivalent such a function is easily proved to exist, but how to calculate it is an entirely different story. This long section is exclusively devoted to this problem.

At least, the value of the function $ITERATION(B, F)$ for B is independent of that for B'. Even if $B \equiv_F B'$, but also if $B \not\equiv_F B'$, it is not required to be the same. This is a simplification because the computation of $ITERATION(B, F)$ is only affected by the set of variables B, not by any other set B'.

The requirements on $ITERATION(B, F)$ are listed in Definition 6: is a set of non-tautological clauses $B' \rightarrow x$ such that $F \models B' \rightarrow x$, $B' \equiv_F B$, $x \in SFREE(B, F)$ and $SCL(B, F) \cup ITERATION(B, F) \models BCL(B, F)$.

The algorithm presented in this section is implemented in the Python program `singlehead.py`: it first determines the heads of the clauses; it then calculates the set of all possible bodies to be associated with them; for each association, the resulting set is checked. The first two steps automatically guarantee the first three conditions, the second checks the fourth.

The heads are uniquely determined. This is not the case for the bodies, nor it is the case for their associations with the heads. Efficiency depends on how many alternatives can be excluded in advance, and how fast they can be discarded if they are not valid.

4.1 Heads in search of bodies

The heads of the clauses of $ITERATION(B, F)$ are uniquely determined. The condition of validity implies an exact set of heads; two valid iteration functions do not differ on their heads. If $ITERATION(B, F)$ is also required not to contain duplicated heads, it must contain exactly a clause for each head.

The starting point is $BCN(B, F) = \{x \mid F \models B \rightarrow x\}$, the set of variables entailed by F and B. A valid iteration function $ITERATION(B, F)$ contains only clauses with head in $BCN(B, F)$. Not the other way around, because:

- $ITERATION(B, F)$ does not contain heads in $SFREE(B, F)$ but $BCN(B, F)$ does;

- $ITERATION(B, F)$ does not contain heads that are in all sets equivalent to B, but $BCN(B, F)$ does; if a clause $B' \rightarrow x$ is in $ITERATION(B, F)$ then $B' \equiv_F B$; if x is in all sets equivalent to B then $x \in B'$; the clause $B' \rightarrow x$ is true but tautological.

The second point may look moot. As a matter of facts, it is not. An example clarifies why $ITERATION(B, F)$ cannot include tautologies. The formula is $F = \{ab \rightarrow c, ac \rightarrow b, d \rightarrow a\}$, in the following figure and in the `equiall.py` test file. The sets $\{a, b\}$ and $\{a, c\}$ are equivalent. The clauses $ab \rightarrow a$ and $ac \rightarrow a$ are tautological. While they look harmless, useless but harmless, they are not. Once in $ITERATION(\{a, b\}, F)$, they conflict with $ITERATION(\{d\}, F)$, which needs $d \rightarrow a$. The formula returned by the reconstruction algorithm would not be single-head while F is single-head equivalent, being single-head. Variables like a are not possible heads for $\{a, b\}$ although they are in $BCN(\{a, b\}, F)$ and may be in $SFREE(\{a, b\}, F)$. Excluding tautologies excludes them.
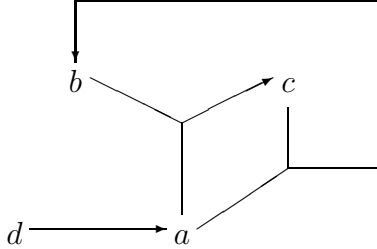
Finding such variables is easy using the real consequences [Lib20c]: $RCN(B, F) = \{x \mid F \cup (BCN(B, F) \backslash \{x\}) \models x\}$. They are the variables that follow from B because of some clauses of F, not just because they are in B.

Lemma 17 If $x \in BCN(B, F) \backslash RCN(B, F)$ and $F \models B \equiv B'$, then $x \in B'$.

Proof. The contrary of the claim is proved contradictory: $x \notin B'$.

Since x is in $BCN(B, F) \backslash RCN(B, F)$, it is in $BCN(B, F)$ but not in $RCN(B, F)$.

The first condition $x \in BCN(B, F)$ is defined as $F \cup B \models x$. With the assumption $F \models B \equiv B'$, it implies $F \cup B' \models x$.

The second condition $x \notin RCN(B, F)$ is defined as $F \cup (BCN(B, F) \backslash \{x\}) \not\models x$. The assumption $F \models B \equiv B'$ implies $F \models B \rightarrow B'$, which in turn implies $B' \subseteq BCN(B, F)$. Because of the additional assumption $x \notin B'$, this containment strengthens to $B' \subseteq BCN(B, F) \backslash \{x\}$. Since $F \cup (BCN(B, F) \backslash \{x\}) \not\models x$, the entailment $F \cup B' \not\models x$ follows by monotonicity, contradicting $F \cup B' \models x$. $\square$

This lemma tells that a variable in $BCN(B, F)$ but not in $RCN(B, F)$ belongs to all sets B' that are equivalent to B according to F. The following lemma proves something in the opposite direction: the variables in $RCN(B, F)$ may be heads of $ITERATION(B, F)$.

Lemma 18 If $x \in RCN(B, F)$ then $B' \rightarrow x \in BCL(B, F)$ for some B'.

Proof. Since $F \cup B$ implies every variable in $BCN(B, F)$ by definition, it implies $BCN(B, F)$. A consequence of the deduction theorem is $F \models B \rightarrow BCN(B, F)$. Every variable in B is entailed by $F \cup B$ and is therefore in $BCN(B, F)$. A consequence of $B \subseteq BCN(B, F)$ is $F \models BCN(B, F) \rightarrow B$. Equivalence is therefore proved: $F \models B \equiv BCN(B, F)$.

The assumption $x \in RCN(B, F)$ is defined as $F \cup (BCN(B, F) \backslash \{x\}) \models x$. Combined with the tautological entailment $F \cup (BCN(B, F) \backslash \{x\}) \models BCN(B, F) \backslash \{x\}$, it implies $F \cup (BCN(B, F) \backslash \{x\}) \models BCN(B, F) \backslash \{x\} \cup \{x\}$. This is the same as $F \cup (BCN(B, F) \backslash \{x\}) \models BCN(B, F)$, which is the same as $F \models BCN(B, F) \backslash \{x\} \rightarrow BCN(B, F)$. The converse implication is a consequence of monotony. This proves the equivalence $F \models BCN(B, F) \backslash \{x\} \equiv BCN(B, F)$. By transitivity of equivalence, $F \models B \equiv BCN(B, F) \backslash \{x\}$.

The set B' required by the claim of the lemma is $BCN(B,F)\backslash\{x\}$. The final consequence of the last paragraph is $F \models B \equiv B'$. Another entailment proved is $F \cup (BCN(B,F)\backslash\{x\}) \models x$, which is the same as $F \models B' \rightarrow x$. The last requirement for $B' \rightarrow x$ to be in $BCL(B,F)$ is $x \notin B'$, which is the case because B' is by construction the result of removing x from another set. □

This is not exactly the reverse of the previous lemma, in that it does not precisely prove that $RCN(B,F)$ contains the variables that are in some sets equivalent to B but not all. Yet, it allows for a simple way for determining the heads of the clauses in $ITERATION(B,F)$.

Lemma 19 *The heads of the clauses in $ITERATION(B,F)$ for a valid iteration function is $HEADS(B,F) = RCN(B,F) \cap SFREE(B,F)$.*

Proof. The proof comprises two parts: first, if $B' \rightarrow x \in ITERATION(B,F)$ then $x \in HEADS(B,F)$; second, if $x \in HEADS(B,F)$ then $B' \rightarrow x \in ITERATION(B,F)$ for some set of variables B'.

The first part assumes $B' \rightarrow x \in ITERATION(B,F)$. The definition of valid iteration includes $x \notin B'$, $F \models B \equiv B'$, $F \models B' \rightarrow x$ and $x \in SFREE(B,F)$. A consequence of $F \models B \equiv B'$ is that every element of B' is entailed by $F \cup B$, and is therefore in $BCN(B,F)$. This proves the containment $B' \subseteq BCN(B,F)$. Since x is not in B', this containment strengthens to $B' \subseteq BCN(B,F)\backslash\{x\}$. The assumption $F \models B' \rightarrow x$ implies $F \cup B' \models x$. Since $B' \subseteq BCN(B,F)\backslash\{x\}$, this implies $F \cup (BCN(B,F)\backslash\{x\}) \models x$. This entailment defines $x \in RCN(B,F)$. Since $x \in SFREE(B,F)$ also holds, the claim $x \in HEADS(B,F)$ follows.

The second part of the proof assumes $x \in HEADS(B,F)$. This is defined as $x \in RCN(B,F)$ and $x \in SFREE(B,F)$. By Lemma 18, the first implies $B' \rightarrow x \in BCL(B,F)$ for some B'. By the assumption of validity of the iteration function, $SCL(B,F) \cup ITERATION(B,F)$ implies $B' \rightarrow x$. By Lemma 1, $SCL(B,F) \cup ITERATION(B,F)$ contains a clause $B'' \rightarrow x$. Since $x \in SFREE(B,F)$, this clause is not in $SCL(B,F)$, and is therefore in $ITERATION(B,F)$. □

The name of this function $HEADS(B,F)$ needs a clarification. The aim of $ITERATION(B,F)$ is to reproduce $BCL(B,F)$. Since $HEADS(B,F)$ are the heads of $ITERATION(B,F)$, they are the heads needed to reproduce $BCL(B,F)$. Not.

A part of the definition of $ITERATION(B,F)$ is that its heads are all in $SFREE(B,F)$. This restriction presumes the formula single-head equivalent: it only works if variables that are not heads in $SCL(B,F)$ suffice as heads of clauses that reproduce $BCL(B,F)$. If they do not, $ITERATION(B,F)$ is unable to imply $BCL(B,F)$. The following formula exemplifies such a situation.

$$F = \{a \rightarrow x, x \rightarrow b, b \rightarrow x\}$$

The heads of $SCL(\{a\},F)$ are $\{b,x\}$. The other variables are $SFREE(\{a\},F) = \{a\}$. Therefore, $HEADS(\{a\},F) = \emptyset$ since $a \notin RCN(\{a\},F)$. The heads of $SCL(\{a\},F)$ and $ITERATION(\{a\},F)$ are disjoint by definition, this is not the problem. The problem is that the available heads $HEADS(\{a\},F) = \emptyset$ are insufficient for building clauses that entail $BCL(B,F)$.

The first step for obtaining $ITERATION(B, F)$ is generating its heads. Finding the bodies and attaching them to these heads is the hard part of the reconstruction algorithm. Finding the heads is not.

4.2 Every body on its own heads

A side effect of Lemma 19 is an equivalent condition to single-head equivalence. The following is a preliminary lemma.

Lemma 20 *If a valid iteration function is such that $ITERATION(B, F)$ does not contain multiple clauses with the same head for any B, the return value of the reconstruction algorithm is single-head if and only if $HEADS(B, F)$ and $HEADS(B', F)$ do not overlap for any B and B' such that $F \not\models B \equiv B'$.*

Proof. One direction of the claim is that if the reconstruction algorithm returns two clauses with the same head then $HEAD(B, F) \cap HEAD(B', F) \neq \emptyset$ for some B and B' such that $F \not\models B \equiv B'$. The reconstruction algorithm returns the union of all sets $ITERATION(B, F)$ it calculates. Since this union contains two clauses of the same head, one comes from $ITERATION(B, F)$ and one from $ITERATION(B', F)$ because no single set $ITERATION(B, F)$ contains multiple clauses with the same head. Lemma 9 tells that the reconstruction algorithm does not determine both $ITERATION(B, F)$ and $ITERATION(B', F)$ if $F \models B \equiv B'$. This proves $F \not\models B \equiv B'$. The heads of $ITERATION(B, F)$ and $ITERATION(B', F)$ are respectively $HEADS(B, F)$ and $HEADS(B', F)$ by Lemma 19. These sets overlap because $ITERATION(B, F)$ contains a clause with the same head as one of $ITERATION(B', F)$. This proves $HEADS(B, F) \cap HEADS(B', F) \neq \emptyset$ with $F \not\models B \equiv B'$.

The other direction of the claim is that $HEADS(B, F) \cap HEADS(B', F) \neq \emptyset$ with $F \not\models B \equiv B'$ implies that the reconstruction algorithm returns two clauses with the same head. Lemma 8 states that the reconstruction algorithm determines $ITERATION(B'', F)$ for some B'' such that $F \models B \equiv B''$. The heads of the clauses in $ITERATION(B'', F)$ are $HEADS(B'', F)$, which is the same as $HEADS(B, F)$ because $F \models B \equiv B''$. For the same reason, the reconstruction algorithm also determines $ITERATION(B''', F)$ with $F \models B' \equiv B'''$, and its heads are $HEADS(B', F)$. Let $x \in HEADS(B, F) \cap HEADS(B', F)$. Both $ITERATION(B'', F)$ and $ITERATION(B''', F)$ contain a clause of head x. The body cannot be the same because the first is equivalent to B'' and the second to B''', and these sets are respectively equivalent to B and B', which are not equivalent by assumption. As a result, these two clauses have the same head but different bodies. □

The condition that the sets $HEADS(B, F)$ are disjoint is logical, not algorithmic. It is independent of the reconstruction algorithm. It depends on the definition of $HEADS(B, F)$ which is based on $RCN(B, F)$ and $SFREE(B, F)$, both defined in terms of entailments. It turns the overall search for a single-head formula equivalent to F into the local search for the clauses of $ITERATION(B, F)$.

Lemma 21 *A formula F is equivalent to a single-head formula if and only if a valid iteration function is such that $ITERATION(B, F)$ does not contain two clauses with the same head for any B and $HEADS(B, F) \cap HEADS(B', F)$ is empty for every B and B' such that $F \not\models B \equiv B'$.*

Proof. The long statement of the lemma is shortened by naming its conditions: (a) a valid iteration function is such that $ITERATION(B,F)$ does not contain two clauses with the same head for any B; (b) $HEADS(B,F) \cap HEADS(B',F)$ is empty for every B and B' such that $F \not\models B \equiv B'$. The claim is that F is single-head equivalent if and only if both (a) and (b) hold.

The first direction of the claim assumes (a) and (b) and proves F single-head equivalent. Lemma 20 proves that the return value of the reconstruction algorithm is single-head from (a) and (b). Lemma 11 proves that the reconstruction algorithm returns a formula equivalent to F when it uses a valid iteration function, which is part of (a). This proves that F is single-head equivalent, as required.

The other direction of the claim assumes F single-head equivalent and proves (a) and (b).

Since F is equivalent to a single-head formula F', Lemma 6 applies: $ITERATION(B,F) = \{B' \to x \in F' \mid F' \models B \equiv B'\}$ is a valid iteration function.

It is proved to meet condition (a) by contradiction. Let $B'' \to x$ and $B''' \to x$ be two different clauses of $ITERATION(B,F)$. They are in F' because $ITERATION(B,F)$ only contains clauses of F'. This implies that F' is not single-head, while it is assumed to be.

Condition (b) is proved by contradiction. Its converse is $x \in HEADS(B,F)$ and $x \in HEADS(B',F)$ with $F \not\models B \equiv B'$. Lemma 19 proves that the heads of $ITERATION(B,F)$ and $ITERATION(B',F)$ are respectively $HEADS(B,F)$ and $HEADS(B',F)$. Therefore, a clause $B'' \to x$ is in $ITERATION(B,F)$ and a clause $B''' \to x$ is in $ITERATION(B',F)$. Both sets comprise only clauses of F' by construction. Therefore, both clauses are in F'. The definition of iteration function includes $F \models B \equiv B''$ and $F \models B' \equiv B'''$. The assumption $F \not\models B \equiv B'$ implies $F \not\models B'' \equiv B'''$, which implies $B'' \neq B'''$. Two clauses of the same head but different bodies are in F', which was assumed single-head. $\square$

The condition as stated by the lemma is infeasible to check since exponentially many sets of variables exist. However, only the sets that are bodies of clauses of the formula need to be checked.

Lemma 22 *A formula F is equivalent to a single-head formula if and only if there exists a valid $ITERATION(B,F)$ that never produces two clauses with the same head and $HEADS(B,F) \cap HEADS(B',F)$ is empty for every clauses $B \to x$ and $B' \to y$ of F with $F \not\models B \equiv B'$.*

Proof. If F is single-head equivalent, $HEADS(B,F) \cap HEADS(B',F)$ is empty for every pair of sets B and B' that are not equivalent according to $\equiv_F$ by Lemma 21. It is therefore empty in the particular case where B and B' are bodies of clauses of F. This is one direction of the claim.

The other direction requires proving that F is single-head equivalent from the other properties mentioned in the claim. The first condition includes the existence of a valid iteration function $ITERATION(B,F)$. The second is that $HEADS(B,F) \cap HEADS(B',F)$ is empty whenever $F \not\models B \equiv B'$ and B and B' are bodies of clauses of F. When a valid iteration function exists, the second extends to the case where B and B' that are not bodies of clauses of F. The claim follows by Lemma 21.

By Lemma 14, either $ITERATION(B,F) = \emptyset$ or $F \models B \equiv B''$ hold for some clause $B'' \to x \in F$. If $ITERATION(B,F)$ is empty so is $HEADS(B,F)$ because the latter is the

set of the heads of the former by Lemma 19. In this case, $HEADS(B, F) \cap HEADS(B', F) = \emptyset$ follows. The same holds for B' by symmetry. The claim is therefore proved if either B or B' are not each $\equiv_F$-equivalent to a body of F.

Remains to prove it if both B and B' are equivalent to a body of F each. Let B'' and B''' be these bodies. Since B and B' are $\equiv_F$-equivalent, so are B'' and B''. Since $F \models B'' \equiv B'''$ and $HEADS(B, F)$ is defined from what F entails, it coincides with $HEADS(B''', F)$. For the same reason, $HEADS(B', F)$ coincides with $HEADS(B'''', F)$. Since B'' and B''' are bodies of clauses of F, $HEADS(B'', F) \cap HEADS(B''', F) = \emptyset$ holds by assumption. It implies $HEADS(B, F) \cap HEADS(B', F) = \emptyset$, the claim. $\square$

4.3   Head-on

Lemma 19 tells the heads of the clauses that may be in $ITERATION(B, F)$, but also helps to find their bodies: since the heads are known and the clauses are consequences of F, the bodies are found by bounding resolution to the heads.

Lemma 23 *If $F \models B' \to x$ and $x \notin B'$, a clause with head x and body contained in B' is obtained by repeatedly resolving a non-tautological clause of F with clauses of F^x.*

Proof. Lemma 1 applied to $F \models B' \to x$ and $x \notin B'$ proves $F^x \models B' \to B$ for some $B \to x \in F$. This is the clause of F in the statement of the lemma. The rest of the proof shows that it is not a tautology and that it resolves with a sequence of clauses of F^x, resulting in a subclause of $B' \to x$.

The clause $B \to x$ is first proved not to be a tautology. The entailment $F^x \models B' \to B$ is the same as $F^x \cup \{B'\} \models B$. The formula $F^x \cup \{B'\}$ does not contain x by the definition of F^x and the assumption that $B' \to x$ is not a tautology ($x \notin B'$). Since it is definite Horn and does not contain x it satisfied by the model that sets all variables to true but x: all clauses are satisfied since they contain at least a positive variable, which is not x. This model does not satisfy x, proving $F^x \cup \{B'\} \models x$ impossible and disproving $x \in B$.

Remains to show how to resolve $B \to x$ with the clauses of F^x. It immediately follows from a property proved by induction.

The induction is on the size of a formula G. The property is: if G entails $B' \to C$ then $C \cup D \to x$ resolves with some clauses of G into $B'' \cup D \to x$ where B'' is a subset of B'. The clause $C \cup D \to x$ is not required to be in G.

Since induction is on the size of G, the base case is that G is empty. An empty formula implies $B' \to C$ only if $C \subseteq B'$. The claim holds with a zero-length resolution, since $C \cup D \to x$ already satisfies $C \subseteq B'$. This proves the base case of induction.

The inductive case is for an arbitrary formula G.

If C is a subset of B', the claim again holds because a zero-length resolution turns $C \cup D \to x$ into itself, which satisfies $C \subseteq B'$.

If C is not a subset of B', then $C \backslash B'$ contains some variables. Let $c \in C \backslash B'$ be one of them: $c \in C$ and $c \notin B'$. The first condition $c \in C$ makes $G \models B' \to C$ include $G \models B' \to c$. The second condition $c \notin B'$ and this conclusion are the premises of Lemma 1, which proves $G^c \models B' \to E$ for some clause $E \to c \in G$.

Since $c \in C$, this clause $E \to c$ of G resolves with $C \cup D \to x$ into $C \backslash \{c\} \cup E \cup D \to x$. The entailment $G^c \models B' \to E$ allows applying the inductive assumption: $C \backslash \{c\} \cup E \cup D \to x$ resolves with clauses of G^c into $C \backslash \{c\} \cup B''' \cup D \to x$, where $B''' \subseteq B'$.

Overall, $C \cup D \to x$ resolves with a clause of G into $C \backslash \{c\} \cup E \cup D \to x$, which then resolves with clauses of $G^b \subseteq G$ into $C \backslash \{c\} \cup B''' \cup D \to x$ where $B''' \subseteq B'$. The last clause is similar to the first $C \cup D \to x$ in that its head is the same and its body minus D is still implied by B'. Indeed, $G \models B' \to C \backslash \{c\} \cup B'''$ follows from $G \models B' \to C$ and $B''' \subseteq B'$. Yet, the variables of $C \backslash \{c\} \cup B'''$ not in B' are one less than those of C.

Applying the same mechanism to another variable of $C \backslash \{c\} \cup B'''$ not in B' if any is therefore possible, and results in another sequence of resolutions that replace that variable with variables of B'. Since the number of variables not in B' strictly decreases, the process terminates with a clause containing zero of them. The resulting clause is $B'' \cup D \to x$ with $B'' \subseteq B'$.

This proves the induction claim.

The premise of the induction claim is $G \models B' \to C$; its conclusion is that $C \cup D \to x$ resolves with clauses of G into $B'' \cup D \to x$ with $B'' \subseteq B'$.

The premise is satisfied by $F^x \models B' \to B$, obtained in the first paragraph of the proof. The conclusion is that $B \cup \emptyset \to x$ resolves with clauses of F^x into $B'' \cup \emptyset \to x$ with $B'' \subseteq B'$. Since the first paragraph of the proof also states $B \to x \in F$, all parts of the lemma are proved. □

The goal is still to find the clauses of $ITERATION(B, F)$. A previous result shows that their heads are $HEADS(B, F)$ if the formula is single-head equivalent. This lemma restricts its bodies. It does not restrict them enough, however.

For example, since $abc \to d$ is a clause of $F = \{a \to b, ac \to d, abc \to d\}$, it is the result of a zero-length resolution starting from itself. Yet, $ac \to d$ is obtained in the same way. It implies $abc \to d$, making it redundant. If a set of clauses containing $ab \to d$ fails at being $ITERATION(B, F)$, the same test with $abc \to d$ in its place fails as well.

The inclusion of $abc \to d$ among the clauses to test is a problem for two reasons:

- the bodies generated this way are associated to the heads in $HEADS(B, F)$ to form the candidates for a valid iteration function; the number of these combinations increases exponentially with the number of bodies;

- as explained later, a candidate set for $ITERATION(B, F)$ is checked using the result of this bounded form of resolution; namely, its result on F and its result on the formula under construction plus the candidate set are compared; this comparison can be done syntactically, but only if all clauses in the candidate set are minimal.

Retaining only the clauses of minimal bodies solves both problems:

$$HCLOSE(H, F) = \{B \to x \mid F \models B \to x, \ x \in H \backslash B, \ \nexists B' \subset B \ . \ F \models B' \to x\}$$

This set can be generated by removing all clauses containing others from the result of the resolution procedure hinged on H. The `reconstruct.py` program does it by an obvious function $minimal(C)$ that returns the clauses of C whose body does not strictly contain

others. The algorithm that generates $HCLOSE(B, F)$ calls it multiple times to reduce the set of clauses it works on.

Algorithm 3 $hclose(H, F)$

1. $C = \{B \rightarrow x \in F \mid x \in H\backslash B\}$

2. $C = minimal(C)$

3. $R = \emptyset$

4. $T = C\backslash R$

5. *while $T \neq \emptyset$:*

    (a) for $B \rightarrow x \in T$:

        for $B'' \rightarrow y \in F$:

            if $B' \rightarrow x \in resolve(B'' \rightarrow y, B \rightarrow x)$ and $x \notin B'$:

                $C = C \cup \{B' \rightarrow x\}$

    (b) $C = minimal(C)$

    (c) $R = R \cup T$

    (d) $T = C\backslash R$

6. *return C*

Since the main loop of this algorithm is a "while" statement on a set T that is changed in its iterations, its termination is not obvious. The following lemma proves it.

Lemma 24 *The* `hclose()` *algorithm (Algorithm 3) always terminates.*

Proof. If a clause is in T, it is added to R in the next iteration of the main loop by the instruction $R = R \cup T$ and removed from T by $T = C\backslash R$.

The clause remains in R because this set is never removed elements. It is never inserted again in T since this set is only changed by $T = C\backslash R$. This proves that T does not contain the same clause in two different iterations.

The number of clauses deriving from resolution from a fixed formula is finite. Since all clauses added to T derive from resolution, if the iterations overcome this finite number, T is empty in at least one of them. This condition terminates the loop. $\square$

Let $HCLOSEALL(H, F)$ be like $HCLOSE(H, F)$ without minimality: $HCLOSEALL(H, F) = \{B \rightarrow x \mid F \models B \rightarrow x,\ x \in H\backslash B\}$.

The minimal subclauses of $HCLOSEALL(H, F)$ equivalently define $HCLOSE(H, F)$.

Lemma 25 *The minimal subclauses of $HCLOSEALL(H, F)$ are $HCLOSE(H, F)$.*

Proof. The definition of $B \to x \in HCLOSE(H, F)$ is $F \models B \to x$, $x \in H \backslash B$ and no subclause $B' \to x$ of $B \to x$ is entailed by F. The same clause $B \to x$ is minimal in $HCLOSEALL(H, F)$ if $F \models B \to x$, $x \in H \backslash B$ and no subclause $B' \to x$ of $B \to x$ is in $HCLOSEALL(H, F)$. The definition $B' \to x \in HCLOSEALL(H, F)$ comprises $F \models B' \to x$ but also $x \in H \backslash B'$. The latter is however not a restriction: if $B' \to x$ is a subclause of $B \to x$, which satisfies $x \in H \backslash B$, it also satisfies $x \in H \backslash B'$ since $B' \subseteq B$. □

This set $HCLOSEALL(H, F)$ is not interesting by itself. It is only used as an indirect means to prove the correctness of the algorithm. The first step in that direction proves that the algorithm only produces clauses of $HCLOSEALL(H, F)$. Not all of them, though. Yet, for each clause of $HCLOSEALL(H, F)$ it produces a subclause. Since the only subclause in $HCLOSEALL(H, F)$ of a minimal clause of $HCLOSEALL(H, F)$ is itself, this proves that the algorithm generates it. Finally, the last statement $C = minimal(C)$ excludes all clauses that are not minimal.

The first step of the proof proves that Algorithm 3 only returns clauses of $HCLOSEALL(H, F)$.

Lemma 26 *Every clause produced by the* `hclose()` *algorithm (Algorithm 3) is in* $HCLOSEALL(H, F)$.

Proof. The clauses in $HCLOSEALL(H, F)$ are defined by being entailed by F, by having their head in H and not being tautologies. The claim is proved by showing the all clauses in C satisfy these conditions throughout the execution of the algorithm.

The set C is initialized by $C = \{B \to x \in F \mid x \in H \backslash B\}$, and $x \in H \backslash B$ implies $x \notin B$. It is then changed only by $C = C \cup \{B' \to x\}$, which is only executed if $x \notin B'$. This proves that C never contains tautologies.

The other two properties are proved by showing that they are initially satisfied and are never later falsified: every clause in C is entailed by F and its head is in H.

The set C is initialized by $C = \{B \to x \in F \mid x \in H \backslash B\}$. All clauses in F are entailed by F, and $x \in H \backslash B$ includes $x \in H$.

A clause $B' \to x$ is added to C only if it is the result of resolving a clause $B \to x$ of C with a clause $B'' \to y$ of F. Since all clauses of C are entailed by F, so is $B' \to x$. Since all clauses of C have head in H, the head x of $B' \to x$ is in H.

This proves that every clause in C is in $HCLOSEALL(H, F)$. Since the algorithm returns C, it only returns clauses of $HCLOSEALL(H, F)$. □

The algorithm returns clauses of $HCLOSEALL(H, F)$, but not all of them. As expected, since C is minimized. Yet, a subclause of every clause of $HCLOSEALL(H, F)$ is returned.

Lemma 27 *A subclause of every clause in* $HCLOSEALL(H, F)$ *is returned by the* `hclose()` *algorithm (Algorithm 3).*

Proof. The definition of $B' \to x \in HCLOSEALL(H, F)$ is $F \models B' \to x$ and $x \in H \backslash B'$. The latter implies $x \notin B'$. Lemma 23 proves that a subclause of $B' \to x$ is obtained by iteratively resolving a clause $B \to x \in F$ such that $x \notin B$ with clauses of F^x.

The algorithm initializes C with the non-tautological clauses of F with head in H. This includes $B \to x$ since x is in H but not in B. The algorithm then iteratively adds to C the resolvent of clauses of C with clauses of F, but also removes clauses by $C = minimize(C)$.

For each clause in the derivation, one of its subclauses is proved to be in C in the last step of the algorithm.

This property is proved iteratively: for the first clause of the derivation, and then for every resolvent.

The first clause $B \to x$ of the derivation is shown to satisfy this property. Being a clause of F such that $x \in H \backslash B$, it is added to C by $C = \{B \to x \mid F \models B \to x, \ x \in H \backslash B\}$. The only instruction removing clauses from C is $C = minimal(C)$, which only removes clauses that strictly contain other clauses of C. In turn, these contained clauses are removed only if C contains strict subclauses of them. By induction on the size of the subclause, C contains a subclause of $B \to x$ at the end of the execution of the algorithm.

An arbitrary clause $B'' \to x$ of the derivation is inductively assumed to satisfy the property: one of its subclauses $B''' \to x$ is in C at the end of execution. The claim is proved on the following clause in the derivation.

Let $D \to b$ with $b \in B''$ be the clause of F^x that resolves with $B'' \to x$. Their resolvent is $B'' \backslash \{b\} \cup D \to x$. Since $B''' \to x$ is a subclause of $B'' \to x$, its body B''' is a subset of B''. It may contain b or not.

If B''' does not contain b then $B''' \subseteq B'' \backslash \{b\}$, and $B''' \to x$ is itself a subclause of $B'' \backslash \{b\} \cup D \to x$, and is by assumption in C at the end of execution.

If B''' contains b then $B''' \to x$ resolves with $D \to b$, producing $B''' \backslash \{b\} \cup D \to x$. This is a subclause of $B'' \backslash \{b\} \cup D \to x$ since $B''' \subseteq B''$. A subclause of this resolvent is proved to be in C at the end of execution.

Since $B''' \to x$ is in C, it has been added by either $C = \{B \to x \mid F \models B \to x, \ x \in H \backslash B\}$ or $C = C \cup \{B' \to x\}$. In the first case, $B''' \to x$ is also copied to T. In the second, since R accumulates the values of T, either T already contained $B''' \to x$ at some previous point or contains it after $T = C \backslash R$. Either way, at some point $B''' \to x$ is in T. At the next iteration, it is resolved with all clauses of F, including $D \to b$. This clause does not contain x since it is in F^x. Neither does B''' since $B''' \to x$ is in C, which is never added a tautology. Their resolvent $B''' \backslash \{b\} \cup D \to x$ is therefore not a tautology and is therefore added to C. It is removed if and when C contains one of its proper subclauses. Which is only removed if and when C contains one of its proper subclauses. This inductively proves that a subclause of $B''' \backslash \{b\} \cup D \to x$, which is a subclause of $B'' \backslash \{b\} \cup D \to x$, is in C at the end of execution.

This shows that a subclause of every clause in the derivation is in the final value of C. This includes the final clause of the derivation: $B' \to x$. Since $B' \to x$ is an arbitrary clause of $HCLOSEALL(H, F)$, this proves that a subclause of every clause of $HCLOSEALL(H, F)$ is produced by the algorithm. $\square$

The rest of the proof of correctness of Algorithm 3 is easy.

Lemma 28 *The* `hclose()` *algorithm (Algorithm 3) generates exactly the minimal subclauses of $HCLOSEALL(H, F)$.*

Proof. For every clause $B' \to x$ of $HCLOSEALL(H, F)$, the algorithm generates one of its subclauses by Lemma 27. Since it is generated by the algorithm, this subclause $B'' \to x$ is in $HCLOSEALL(H, F)$ by Lemma 26. Since $B' \to x$ is minimal, none of its proper subsets is in $HCLOSEALL(H, F)$. Therefore, the subclause $B'' \to x$ of $B' \to x$ is not a proper subclause: it coincides with $B' \to x$. This proves that the algorithm generates every minimal clause of $HCLOSEALL(H, F)$.

In the other direction, since the algorithm returns all minimal clauses of $HCLOSEALL(H,F)$, it cannot return any non-minimal one. Indeed, if $B' \to x \in HCLOSEALL(H,F)$ contains another clause of $HCLOSEALL(H,F)$, it also contains a minimal clause of $HCLOSEALL(H,F)$. By Lemma 26, this minimal clause is in C at the end of the algorithm. Therefore, $B' \to x$ is removed from C by $C = minimal(C)$ and is not returned. $\square$

Since the minimal subclauses of $HCLOSEALL(H,F)$ are exactly $HCLOSE(H,F)$ by Lemma 25, the correctness of the algorithm is proved.

Corollary 2 *The* `hclose()` *algorithm (Algorithm 3) returns $HCLOSE(H,F)$.*

Having shown how $HCLOSE(H,F)$ can be calculated, it can be used to determine $ITERATION(B,F)$, which is then used within the reconstruction algorithm to determine a single-head formula equivalent to a given one if any. How it is used is the theme of the next section.

4.4 Body search

When searching for the value of $ITERATION(B,F)$, Lemma 19 provides half of the answer: its heads are exactly $HEADS(B,F)$. What about its bodies? The definition of a valid iteration function only constraints them to come from $BCL(B,F)$. It is not much of a constraint: being a set of consequences, $BCL(B,F)$ is not bounded by the size of F. Not all of it is necessary, though: if a valid iteration function exists, one such that $ITERATION(B,F)$ is contained in $HCLOSE(HEADS(B,F),UCL(B,F))$ exists as well. Proving this statement is the aim of this section.

The first two lemmas are proved with an arbitrary subset of $SCL(B,F) \cup BCL(B,F)$, which $UCL(B,F)$ is then proved to be.

Lemma 29 *For every H and every formula $F' \subseteq SCL(B,F) \cup BCL(B,F)$, it holds $HCLOSE(H,F') \subseteq SCL(B,F) \cup BCL(B,F)$.*

Proof. The condition $B' \to x \in SCL(B,F) \cup BCL(B,F)$ is the same as $B' \to x \in SCL(B,F)$ or $B' \to x \in BCL(B,F)$. This is an alternative. The first case is $x \notin B'$, $F \models B' \to x$, $F \models B \to B'$ and $F \not\models B' \to B$. The second is $x \notin B'$, $F \models B' \to x$, $F \models B \to B'$ and $F \models B' \to B$. The two alternative subconditions $F \not\models B' \to B$ and $F \models B' \to B$ cancel each other. This proves $B' \to x \in SCL(B,F) \cup BCL(B,F)$ equivalent to $x \notin B'$, $F \models B' \to x$ and $F \models B \to B'$.

The premise of the lemma $F' \subseteq SCL(B,F) \cup BCL(B,F)$ is the same as $x \notin B'$, $F \models B' \to x$ and $F \models B \to B'$ for every clause $B' \to x$ of F'. The claim is the same for every clause $B' \to x$ of $HCLOSE(H,F')$.

The definition of $B' \to x \in HCLOSE(H,F')$ includes $x \in H \backslash B'$, which implies $x \notin B'$, the first requirement for $B' \to x \in SCL(B,F) \cup BCL(B,F)$. It also includes $F' \models B' \to x$. Since F' is contained in $SCL(B,F) \cup BCL(H,F)$, which only contains clauses entailed by F, it is entailed by F. A consequence of $F \models F'$ and $F' \models B' \to x$ is $F \models B' \to x$. This is the second requirement for $B' \to x \in SCL(B,F) \cup BCL(B,F)$.

Only $F \models B \to B'$ remains to be proved. Since F entails F', which contains $B' \to x$, it also entails $B' \to x$. A subclause of $B' \to x$ is obtained by resolving clauses of F. The

claim $F \models B \to B'$ is iteratively proved on all clauses in such a derivation. The base case is proved because $F \models F'$ and $F' \subseteq SCL(B,F) \cup BCL(B,F)$ imply $F \models B \to B'$ for every clause $B' \to x \in F'$.

The induction step assumes $F \models B \to B''$ and $F \models B \to B'''$ for two clauses $B'' \to x$ and $B''' \to y$ obtained from resolution. They resolve if either $x \in B'''$ or $y \in B''$. The result is respectively $((B''\backslash\{x\}) \cup B''') \to y$ or $(B'' \cup (B'''\backslash\{y\})) \to x$. In both cases the body of the resulting clause is a subset of $B'' \cup B'''$. The claim is proved because of $F \models B \to B''$ and $F \models B \to B'''$. $\square$

A specific value for H strengthens the containment proved by the lemma.

Lemma 30 *For every formula $F' \subseteq SCL(B,F) \cup BCL(B,F)$, it holds $HCLOSE(HEADS(B,F), F') \subseteq BCL(B,F)$.*

Proof. Since $F' \subseteq SCL(B,F) \cup BCL(B,F)$, Lemma 29 proves $HCLOSE(H, F') \subseteq SCL(B,F) \cup BCL(B,F)$ for every set of variables H. The claim is proved by showing that $H = HEADS(B,F)$ forbids clauses of $SCL(B,F)$ in $HCLOSE(H, F')$.

By contradiction, $B' \to x$ is assumed to be a clause of $SCL(B,F)$ that is also in $HCLOSE(HEADS(B,F), F')$. The definition of $HCLOSE(HEADS(B,F), F')$ includes $x \in HEADS(B,F)$. Since $HEADS(B,F) = RCN(B,F) \cap SFREE(B,F)$, it holds $x \in SFREE(B,F)$. This is defined as the set of variables that are the head of no clause of $SCL(B,F)$, contradicting the assumption $B' \to x \in SCL(B,F)$. $\square$

A set contained in $SCL(B,F) \cup BCL(B,F)$ is $UCL(B,F)$, as proved by the following lemma.

Lemma 31 *For every formula F and set of variables B, it holds $UCL(B,F) = F \cap (SCL(B,F) \cup BCL(B,F))$.*

Proof. The set $UCL(B,F)$ is defined as the set of clauses $B' \to x$ of F such that $x \notin B$ and $F \models B \to B'$. The entailment defines $B' \leq_F B$, which is the same as either $B' <_F B$ or $B' \equiv_F B$. Since every clause of F is entailed by F, every clause $B' \to x$ of $UCL(B,F)$ satisfies $x \notin B'$, $F \models B' \to x$ and either $B' <_F B$ or $B' \equiv_F B$. Therefore, it is either in $SCL(B,F)$ or in $BCL(B,F)$.

In the reverse direction, every clause of $SCL(B,F)$ and every clause of $BCL(B,F)$ satisfies $x \notin B'$ and $F \models B \to B'$. This is the definition of $UCL(B,F)$ if the clause is also in F. $\square$

Lemma 30 proves $HCLOSE(HEADS(B,F), F') \subseteq BCL(B,F)$ for every subset F' of $SCL(B,F) \cup BCL(B,F)$. The last lemma proves that $UCL(B,F)$ is such a subset. Combining the two lemmas produce the following corollary.

Corollary 3 *For every formula F and set of variables B, it holds $HCLOSE(HEADS(B,F), UCL(B,F)) \subseteq BCL(B,F)$.*

This corollary proves that $HCLOSE(HEADS(B,F), UCL(B,F))$ is a subset of $BCL(B,F)$ and being a subset of $BCL(B,F)$ is a part of the definition of $ITERATION(B,F)$. The other parts are now proved to be met by $HCLOSE(HEADS(B,F), UCL(B,F))$ as well.

This claim requires some preliminary lemmas. The first is the monotonicity of $UCL(B,F)$ with respect to containment of the formula.

Lemma 32 *If $F' \subseteq F$ then $UCL(B, F') \subseteq UCL(B, F)$.*

Proof. The definition of $B' \to x \in UCL(B, F')$ is $B' \to x \in F'$, $x \notin B'$ and $F' \models B \to B'$. The first condition implies $B' \to x \in F$ because $F' \subseteq F$. The third implies $F \models B \to B'$ by monotonicity of entailment. The conditions in the definition of $B' \to x \in UCL(B, F)$ all hold. $\square$

This lemma allows proving that $UCL(B, F)$ is the part of F that matters when checking entailment from B.

Lemma 33 *The following two conditions are equivalent: $F \models B \to x$ and $UCL(B, F) \models B \to x$.*

Proof. If $x \in B$ then $B \to x$ is a tautology. It is therefore entailed by both F and $UCL(B, F)$ and the claim is proved. The rest of the proof covers the case $x \notin B$.

If $UCL(B, F)$ implies $B \to x$ also F does, since $UCL(B, F)$ is a subset of F.

Remains to prove the converse. The assumption is $F \models B \to x$ and the claim is $UCL(B, F) \models B \to x$.

This is proved by induction on the size of F.

The base case is $F = \emptyset$, where $F \models B \to x$ implies $x \in B$, a case already proved to satisfy the claim.

The induction step proves that $F \models B \to x$ implies $UCL(B, F) \models B \to x$ assuming the same implication for every F' strictly smaller than F.

The claim is already proved when $x \in B$. Otherwise, $x \notin B$ and $F \models B \to x$ imply by Lemma 1 that F contains a clause $B' \to x$ such that $F^x \models B \to B'$. If $x \in B'$ then $B' \to x$ is a tautology and is therefore entailed by $UCL(B, F)$. Otherwise, $x \notin B'$. The condition $B' \to x \in F$ implies $F \models B' \to x$. The condition $F^x \models B \to B'$ implies $F \models B \to B'$. Otherwise, all conditions for $B' \to x$ being in $UCL(B, F)$ are met. Therefore, $UCL(B, F)$ entails $B' \to x$.

By definition, $F^x \models B \to B'$ is the same as $F^x \models B \to b$ for every $b \in B'$. Since F^x is smaller than F because it does not contain $B' \to x$ at least, the induction assumption applies: $UCL(B, F^x) \models B \to b$. Since $F^x \subseteq F$, Lemma 32 tells that $UCL(B, F^x) \subseteq UCL(B, F)$. By monotonicy of entailment, $UCL(B, F) \models B \to b$ follows. This holds for every $b \in B'$, implying $UCL(B, F) \models B \to B'$.

This conclusion $UCL(B, F) \models B \to B'$ with the previous one $UCL(B, F) \models B' \to x$ implies $UCL(B, F) \models B \to x$. $\square$

Since $F \models B \to x$ defines $x \in BCN(B, F)$ and $UCL(B, F) \models B \to x$ defines $x \in BCN(B, UCL(B, F))$, an immediate consequence is the equality of these two sets.

Lemma 34 *For every formula F and set of variables B it holds $BCN(B, F) = BCN(B, UCL(B, F))$.*

Proof. By Lemma 33, $F \models B \to x$ is the same as $UCL(B, F) \models B \to x$. The first condition defines $x \in BCN(B, F)$, the second $x \in BCN(B, UCL(B, F))$. $\square$

Contrary to the other functions defined in this article, $UCL(B, F)$ only contains clauses of F. A clause that is not in F is never in $UCL(B, F)$ even if it is entailed by it. This set is the restriction of F to the case where B is true. Only the clauses in $UCL(B, F)$ matter

when deriving something from $F \cup B$. They are the only clauses that matter to $BCN(B, F)$, $RCN(B, F)$, $SCL(B, F)$ and $BCL(B, F)$. If $UCL(B, F)$ replaces F, these do not change. More generally, if something is defined in terms of what F and B entail, it is unaffected by the removal of all clauses not in $UCL(B, F)$ from F.

This claim requires a preliminary result: $UCL()$ is monotonic with respect to entailment of the set of variables.

Lemma 35 *If $F \models A \rightarrow B$ then $UCL(B, F) \subseteq UCL(A, F)$.*

Proof. By definition, $B' \rightarrow x \in UCL(B, F)$ is $x \notin B'$, $F \models B \rightarrow B'$ and $B' \rightarrow x \in F$. The second condition $F \models B \rightarrow B'$ implies $F \models A \rightarrow B'$ since $F \models A \rightarrow B$ is assumed. With $x \notin B'$ and $B' \rightarrow x \in F$, this condition defines $B' \rightarrow x \in UCL(A, F)$. $\square$

Lemma 33 generalizes from clauses whose body is equal to the given set of variables to clauses whose body is entailed by it.

Lemma 36 *If $F \models A \rightarrow B$, then $F \models B \rightarrow x$ is equivalent to $UCL(A, F) \models B \rightarrow x$.*

Proof. A consequence of $UCL(A, F) \models B \rightarrow x$ is $F \models B \rightarrow x$ since $UCL(A, F)$ is a subset of F. The first direction of the claim is proved even if $F \models A \rightarrow B$ does not hold.

The other direction is that $F \models B \rightarrow x$ implies $UCL(A, F) \models B \rightarrow x$ if $F \models A \rightarrow B$. By Lemma 33, the assumption $F \models B \rightarrow x$ implies $UCL(B, F) \models B \rightarrow x$. This is not the claim yet because it contains $UCL(B, F)$ instead of $UCL(A, F)$. However, Lemma 35 proves $UCL(B, F) \subseteq UCL(A, F)$ from $F \models A \rightarrow B$. Monotonicity implies $UCL(A, F) \models B \rightarrow x$. $\square$

This lemma proves that checking whether F entails $B \rightarrow x$ where $F \models A \rightarrow B$ can be safely restricted from F to its subset $UCL(A, F)$. This set is determined by the algorithm for $RCN(A, F)$ at almost no additional cost, and may reduce the number of clauses to consider. This reduction is dramatic on large formulae that contain only a small fraction of clauses whose bodies are entailed by A.

This result applies to the search for bodies of $ITERATION(B, F)$.

Lemma 37 *If $ITERATION(B, F)$ is a valid iteration function and $B' \rightarrow x \in ITERATION(B, F)$, then $HCLOSE(HEADS(B, F), UCL(B, F))$ contains a clause $B'' \rightarrow x$ with $B'' \subseteq B'$.*

Proof. The definition of $B' \rightarrow x \in ITERATION(B, F)$ includes $F \models B' \rightarrow x$ and $F \models B \equiv B'$. Equivalence implies entailment: $F \models B \rightarrow B'$. Lemma 36 applies: $F \models B' \rightarrow x$ implies $UCL(B, F) \models B' \rightarrow x$.

Lemma 19 proves $x \in HEADS(B, F)$.

By definition, $HCLOSE(HEADS(B, F), UCL(B, F))$ does not include a clause $B' \rightarrow x$ such that $UCL(B, F) \models B' \rightarrow x$ and $x \in HEADS(B, F)$ only if $UCL(B, F) \models B'' \rightarrow x$ with $B'' \subset B'$. Since the head of this clause is in $HEADS(B, F)$, the same applies to it: it is not in $HCLOSE(HEADS(B, F), UCL(B, F))$ if a proper subclause of it has the same property. This inductively proves that a subclause $B'' \rightarrow x$ of $B' \rightarrow x$ is in $HCLOSE(HEADS(B, F), UCL(B, F))$. $\square$

Corollary 3 prove that searching for a valid $ITERATION(B, F)$ can be restricted to the subsets of $HCLOSE(HEADS(B, F), UCL(B, F))$.

Lemma 38 *If $ITERATION(B, F)$ is a valid iteration function then another valid iteration function with the same heads in the same number of clauses is a subset of $HCLOSE(HEADS(B, F), UCL(B, F))$.*

Proof. Lemma 37 shows that $B' \rightarrow x \in ITERATION(B, F)$ implies $B'' \rightarrow x \in HCLOSE(HEADS(B, F), UCL(B, F))$ for some $B'' \subseteq B'$. This containment implies $B'' \rightarrow x \models B' \rightarrow x$. As a result, replacing $B' \rightarrow x$ with $B'' \rightarrow x$ in $ITERATION(B, F)$ results in a formula entailing $ITERATION(B, F)$, and therefore still entailing $BCL(B, F)$. This is the entailment part of the definition of a valid iteration function.

This replacement also maintains the choice part. By Corollary 3, $HCLOSE(HEADS(B, F), UCL(B, F)) \subseteq BCL(B, F)$. This proves $x \notin B'$, $F \models B' \rightarrow x$ and $F \models B \equiv B'$. Remains to prove $x \in SFREE(B, F)$. It holds because the definition of $B' \rightarrow x \in HCLOSE(HEADS(B, F), UCL(B, F))$ includes $x \in HEADS(B, F)$, which includes $x \in SFREE(B, F)$, which implies $B' \rightarrow x \notin SCL(B, F)$.

This replacement maintains the heads of the clauses because it replaces a clause with another having the same head. $\qquad \square$

This lemma proves that searching for $ITERATION(B, F)$ can be restricted to the subsets of $HCLOSE(HEADS(B, F), UCL(B, F))$. This avoids the inclusion of clauses containing others; a further minimization is shown in the next section.

4.5 Implied by strictly implied subsets

Lemma 38 bounds the search for a valid iteration function to the subsets of $HCLOSE(HEADS(B, F), UCL(B, F))$. Still, the number of such subsets may be exponential. Lemma 19 further bounds the search: the heads of $ITERATION(B, F)$ are not just in $HEADS(B, F)$, they are exactly $HEADS(B, F)$. The subsets that do not contain a head in $HEADS(B, F)$ can be skipped. So can the subsets that contain duplicated heads, since the aim is to find a single-head formula. Yet, these reductions do not always make the number of candidate subsets polynomial. Every further restriction helps.

Some considerations on $HCLOSE(HEADS(B, F), UCL(B, F))$ provide a direction. It constrains clauses to be entailed by $UCL(B, F)$, to have head in $HEADS(B, F)$ and a minimal body with respect to set containment. The first two are necessary because of the definition of validity and because of Lemma 19, but why minimality?

A non-minimal body is never necessary because $B'' \rightarrow x$ entails $B' \rightarrow x$ if $B'' \subset B'$. Everything $B' \rightarrow x$ entails is also entailed by $B'' \rightarrow x$. Both clauses are acceptable for being in $ITERATION(B, F)$, they both consume the same head x, but $B'' \rightarrow x$ entails everything $B' \rightarrow x$ entails. The latter can be disregarded since the former is a valid substitute of it in every respect.

The goal is a set of clauses that entails $BCL(B, F)$ with $SCL(B, F)$. If $B'' \subset B'$, then $SCL(B, F) \cup IT \cup \{B'' \rightarrow x\}$ implies $SCL(B, F) \cup IT \cup \{B' \rightarrow x\}$. What really matters is this implication. When it holds, $BCL(B, F)$ is entailed by the implicant if it is entailed by the implicate. The implication is a consequence of set containment, but is not equivalent to it. It may hold even if set containment does not.

The implication involves the set IT, but is used to restrict the possible clauses to include in IT, when IT is not fixed yet. What can be checked then is whether $SCL(B, F) \cup \{B'' \rightarrow x\}$ implies $SCL(B, F) \cup \{B' \rightarrow x\}$. This is the same as $SCL(B, F) \cup \{B'' \rightarrow x\} \models B' \rightarrow x$,

which holds if $SCL(B, F) \cup B' \models B''$. A body B' can be replaced by another B'' it entails with $SCL(B, F)$.

Lemma 39 *If $ITERATION(B, F)$ is a valid iteration function, $B' \rightarrow x \in ITERATION(B, F)$, $B'' \rightarrow x \in BCL(B, F)$ and $SCL(B, F) \cup B' \models B''$ hold, another valid iteration function is $ITERATION'(B, F) = ITERATION(B, F) \backslash \{B' \rightarrow x\} \cup \{B'' \rightarrow x\}$.*

Proof. All clauses in $ITERATION'(B, F)$ but $B'' \rightarrow x$ are in $ITERATION(B, F)$. They satisfy the condition of choice of the definition of validity, which is now proved for $B'' \rightarrow x$. The assumption $B'' \rightarrow x \in BCL(B, F)$ implies $F \models B'' \rightarrow x$ and $F \models B \equiv B''$. The condition $x \in SFREE(B, F)$ follows from $B' \rightarrow x \in ITERATION(B, F)$. This proves the condition of choice.

Remains to prove the entailment condition: $SCL(B, F) \cup ITERATION'(B, F) \models BLC(B, F)$. By assumption, $SCL(B, F) \cup B' \models B''$. As a result, $SCL(B, F) \cup ITERATION'(B, F) \cup B'$, being equal to $SCL(B, F) \cup ITERATION(B, F) \backslash \{B' \rightarrow x\} \cup \{B'' \rightarrow x\} \cup B'$, entails $SCL(B, F) \cup ITERATION(B, F) \backslash \{B' \rightarrow x\} \cup \{B'' \rightarrow x\} \cup B''$, which entails x. By the deduction theorem, $SCL(B, F) \cup ITERATION'(B, F) \models B' \rightarrow x$. Since $SCL(B, F) \cup ITERATION'(B, F)$ implies the only clause of $ITERATION(B, F)$ it does not contain, it implies it all: $SCL(B, F) \cup ITERATION'(B, F) \models SCL(B, F) \cup ITERATION(B, F)$. Since $ITERATION(B, F)$ is a valid iteration function, it satisfies $SCL(B, F) \cup ITERATION(B, F) \models BCL(B, F)$. By transitivity, $SCL(B, F) \cup ITERATION'(B, F) \models BCL(B, F)$. $\square$

If $B' \rightarrow x$ and $B'' \rightarrow x$ are both in $HCLOSE(HEADS(B, F), UCL(B, F))$, the first can be disregarded if $SCL(B, F) \cup B' \models B''$. This reduces the set of candidates subsets of $HCLOSE(HEADS(B, F), UCL(B, F))$.

This is similar to what Lemma 38 allows when $B'' \subset B'$, but with an important difference: the reverse containment $B' \subset B''$ is not possible, the reverse implication from B'' to B' is. If $SCL(B, F) \models B' \equiv B''$, Lemma 39 allows $B'' \rightarrow x$ to replace $B' \rightarrow x$ but also the other way around. Such an equivalence is not possible under set containment: $B' \subset B''$ contradicts $B'' \subset B'$. If a body is minimal according to set containment, no other body can take its place; a body that is minimal according to implication can be replaced by an equivalent one. While uniquely defining $HCLOSE(H, F)$ is possible, extending it to minimality by entailment is not. Not uniquely. This is why the following definition only gives a condition of minimality rather than uniquely defining a set like $HCLOSE(H, F)$.

Definition 7 *$MINBODIES(U, S)$ is an arbitrary function that returns a subset R of U such that $B' \rightarrow x \in U$ implies $S \cup B' \models B''$ for some clause $B'' \rightarrow x \in R$.*

The function $minbodies(U, S)$ in `singlehead.py` meets this definition. It works on whole clauses $B' \rightarrow x$ and $B'' \rightarrow x$ instead of their bodies B' and B'' only. It starts from a clause of U and repeatedly checks whether its body and S entail the body of another clause of U with the same head. The trace of clauses in this and in all runs avoid loops and repeated search.

The function is called with $U = HCLOSE(HEADS(B, F), UCL(B, F))$ and $S = UCL(B, F) \cap SCL(B, F)$. The resulting subset of $HCLOSE(HEADS(B, F), UCL(B, F))$ may still include some clauses that could be removed for two reasons.

How inference is checked. Lemma 39 allows neglecting a clause if its body and $SCL(B, F)$ entails another body. But the function $minbodies(U, S)$ in `singlehead.py` does not check $SCL(B, F) \cup B' \models B''$ for every set B''. For simplicity, it only checks whether a single variable in B' resolves with a single clause of $UCL(B, F) \cap SCL(B, F)$ to produce B''.

Using resolution instead of entailment is not a limitation because the only difference is that resolution may produce a subclause of an entailed clause. This difference disappears since $B' \to x$ is subset-minimal: none of its subsets are entailed.

Whether resolving a variable in B' with a clause in $UCL(B, F) \cap SCL(B, F)$ once is sufficient is another story. A preliminary analysis suggests that this is not a limitation, and a resolution derivation of B'' from $SCL(B, F) \cup B'$ can always be restructured so that the last step is always the resolution of a body of $HCLOSE(HEADS(B, F), UCL(B, F))$ with a clause of $UCL(B, F) \cap SCL(B, F)$. This would prove by iteration that such a derivation is a sequence of steps like the ones the program follows. Yet, this is unproven.

How minimal elements are searched for. The function $minbodies(U, S)$ in `singlehead.py` follows implications among bodies, avoiding the ones it already visited. The following example (`minbodies.py`) shows that it may return a non-minimal body this way.

$$F = \{bh \to c, ch \to b, ch \to d, x \to h, bde \to x\}$$

The first four clauses make $UCL(B, F) \cap SCL(B, F)$ when $B = \{b, d, e\}$. They imply $bhe \to che$, $che \to bhe$ and $che \to cde$. These three bodies bhe, che and cde are equivalent according to F, but cde is only entailed by the other two according to $UCL(B, F) \cap SCL(B, F)$.
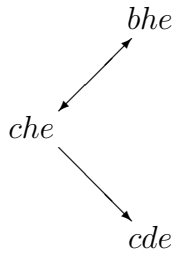
The only necessary clause is $cde \to x$, since its body cde is implied by the other two che and bhe but not the other way around. Yet, `minbodies(U, S)` may also produce $bhe \to x$.

If searching starts from che, it moves to bhe and stops there because the only other body it directly entails it is che, which it skips as already analyzed. The clause $bhe \to x$ is therefore produced even if its body entails cde and not the other way around.

The problem is still due to single resolutions in place of inference, but not only. The only necessary body cde could be found from bhe even using a single resolution at time by not avoiding the bodies already visited like che. This is not done for its computational costs.

These issues may make the function return some clauses that are not minimal. This only affects the running time of the algorithm, not its correctness. It increases the number of candidates for $ITERATION(B, F)$ and ultimately the overall running time, but since all necessary clauses are included the result is correct anyway.

4.6 The algorithm, at last

The overall algorithm for checking whether a formula is single-head equivalent assembles the pieces built so far. It is the reconstruction algorithm (Algorithm 3) where $ITERATION(B, F)$ is not given but calculated employing $HEADS()$, $HCLOSE()$ and $MINBODIES()$.

The reconstruction algorithm iterates over the bodies of the clauses in the formula, sorted by entailment. It starts from the bodies that entail no other and continues to the ones entailing only the ones already processed. For every body B, it calculates $ITERATION(B, F)$ and adds it to the formula under construction G. The only missing bit of this algorithm is $ITERATION(B, F)$. It is found by iterating over the sets of clauses it may contain until a set that entails $BCL(B, F)$ with $SCL(B, F)$ is found.

Algorithm 4 (Singlehead-autoreconstruction algorithm)   *1. $F = \{B \to x \in F \mid x \notin B\}$*

   2. $G = \emptyset$

   3. $P = \{B \mid \exists x.B \to x \in F\}$

   4. while $P \neq \emptyset$

       (a) choose a $<_F$-minimal $B \in P$
       (a set such that $A <_F B$ does not hold for any $A \in P$)

       (b) $H = HEADS(B, F)$

       (c) $M = HCLOSE(H, UCL(B, F))$

       (d) $T = MINBODIES(M, UCL(B, F) \cap SCL(B, F))$

       (e) $E = \mathsf{false}$

       (f) for all $IT = \{C_1 \to h_1, \ldots, C_m \to h_m \mid H = \{h_1, \ldots, h_m\}, \forall i \,.\, h_i \notin C_i, \exists y_1, \ldots, y_m \,.\, C_1 \to y_1, \ldots, C_m \to y_m \in T\}$

         i. if $SCL(B, F) \cup IT \models BCL(B, F)$

           A. $E = \mathsf{true}$

           B. break

       (g) if $\neg E$

         i. fail

       (h) $G = G \cup IT$

       (i) $P = P \backslash \{B' \mid F \models B \equiv B'\}$

5. return G

The overall structure is the same as the reconstruction algorithm (Algorithm 2): a single-head formula G aims at replicating F by accumulating $ITERATION(B,F)$ during a loop over the bodies B of F sorted by $\leq_F$. The difference is that $ITERATION(B,F)$ is not given but found by iterating over sets of clauses IT.

This search is the main contributor to complexity in the algorithm, since exponentially many such sets may exists. This is why a number of hacks for efficiency are implemented:

- instead of checking all subsets of $BCL(B,F)$, only the subsets of $HCLOSE(HEADS(B,F),UCL(B,F))$ are considered;

- this set is further reduced by $MINBODIES()$;

- rather than looping over sets of clauses, each variable in $HEADS(B,F)$ is attached a body coming from $MINBODIES()$; this is correct because all these bodies are equivalent; it is convenient because it automatically excludes all sets of clauses with missing or duplicated heads;

- $RCN(B,F)$ and $UCL(B,F)$ are determined and stored for all bodies of F once at the beginning.

The algorithm is further improved by other optimizations described in separate sections because of the length of their proofs. The rest of this section is about the correctness of Algorithm 4 itself.

Lemma 40 *All non-tautological clauses with head in $HEADS(B,F)$ and body in $HCLOSE(HEADS(B,F),UCL(B,F))$ are in $BCL(B,F)$.*

Proof. The premise of the lemma is that $B' \to x$ is a clause such that $x \notin B'$, $x \in HEADS(B,F)$ and $B' \to y \in HCLOSE(HEADS(B,F),UCL(B,F))$ for some variable y.

Corollary 3 proves that every clause of $HCLOSE(HEADS(B,F),UCL(B,F))$ is in $BCL(B,F)$. The definition of $B' \to y \in BCL(B,F)$ includes $B' \equiv_F B$, which is also part of the definition of $B' \to x \in BCL(B,F)$. Another part is $x \notin B'$, which holds because $B' \to x$ is by assumption not a tautology.

The only missing part is $F \models B' \to x$. The definition of $x \in HEADS(B,F)$ includes $x \in RCN(B,F)$, which implies $x \in BCN(B,F)$ by Lemma 2: $F \models B \to x$. Since the previously proved fact $B' \equiv_F B$ is defined as $F \models B' \equiv B$, this entailment is the same as $F \models B' \to x$, the missing part of the definition of $B' \to x \in BCL(B,F)$. $\square$

Since IT comprises only clauses meeting the condition of the previous lemma, they satisfy its consequence.

Lemma 41 *All sets IT in the autoreconstruction algorithm (Algorithm 4) are contained in $BCL(B,F)$.*

Proof. By construction, IT is made of non-tautological clauses with heads in $HEADS(B,F)$ and bodies in $MINBODIES(HCLOSE(HEADS(B,F),UCL(B,F)),UCL(B,F) \cap SCL(B,F))$. The latter is a subset of its first argument $HCLOSE(HEADS(B,F),UCL(B,F))$ by definition. Therefore, all clauses of IT have heads in $HEADS(B,F)$ and bodies in $HCLOSE(HEADS(B,F),UCL(B,F))$ and are not tautologies. Lemma 40 prove that they are in $BCL(B,F)$. □

Since all sets IT are subsets of $BCL(B,F)$, its clauses meet most of the choice part of the definition of a valid iteration function: $x \notin B'$, $F \models B' \rightarrow x$ and $F \models B \equiv B'$. They also meet the remaining one, $x \in SFREE(B,F)$, since $x \in HEADS(B,F)$.

The entailment part of the definition of validity is not meet in general, but only if the iteration of the external loop ends.

Lemma 42 *At the end of every iteration of the external loop (Step 4) of the autoreconstruction algorithm (Algorithm 4), IT satisfies the conditions for a valid iteration function $ITERATION(B,F)$.*

Proof. Lemma 41 proves that IT only contains clauses of $BCL(B,F)$. By construction, the heads of IT are in $HEADS(B,F)$, a subset of $SFREE(B,F)$. The choice condition for IT being a valid iteration function is met.

The entailment condition is $SCL(B,F) \cup IT \models BCL(B,F)$. This may or may not happen for a specific set IT. However, if no set IT satisfies $SCL(B,F) \cup IT \models BCL(B,F)$ in the internal loop, the algorithm fails. Therefore, the entailment condition for IT holds if the iteration of the external loop ends. □

While IT is a valid iteration function if the algorithm does not fail, the same could be achieved by making the algorithm fail for all sets IT. It is also essential that the algorithm does not fail if a valid iteration function exists.

Lemma 43 *If F has a valid iteration function such that $ITERATION(B,F)$ is single-head, then IT is equal to the value of a valid iteration function $ITERATION(B,F)$ at some iteration of the internal loop (Step 4f) of the autoreconstruction algorithm (Algorithm 4).*

Proof. Lemma 38 proves that if F has a valid iteration function it also has one such that $ITERATION(B,F)$ is a subset of $HCLOSE(HEADS(B,F),UCL(B,F))$ and has the same heads in the same number of clauses. Lemma 39 further restricts this property to the subsets of $MINBODIES(HCLOSE(HEADS(B,F),UCL(B,F)),UCL(B,F) \cap SCL(B,F))$. Since this is a valid iteration function, its heads are $HEADS(B,F)$ by Lemma 19.

This proves that if F has a valid iteration function such that $ITERATION(B,F)$ is single-head, also has one such that $ITERATION(B,F)$ have bodies from $MINBODIES(HCLOSE(HEADS(B,F),UCL(B,F)),UCL(B,F) \cap SCL(B,F))$ and single-head heads in $HEADS(B,F)$. This other one is still a valid iteration function; therefore, it contains no tautology. Since the internal loop (Step 4f) of the algorithm tries all tautology-free associations of these heads with these bodies, it finds it. □

The following lemma has an awkward statement but is necessary to avoid repeating the same argument in multiple proofs.

Lemma 44 *If the autoreconstruction algorithm (Algorithm 4) terminates, the sequence of values of G is the same as in the reconstruction algorithm (Algorithm 1) for some nondeterministic choices and the following valid iteration function.*

$$ITERATION(B, F) = \begin{cases} IT & \text{at the end of the iteration of } B' \\ & \text{if } B \text{ is equivalent to a body } B' \text{ in } F \\ \emptyset & \text{otherwise} \end{cases}$$

Proof. The iteration function is valid: Lemma 42 proves the conditions of validity when B is a body in F, and these conditions carry over to equivalent sets because they are semantical; Lemma 14 applied in reverse proves that the value of every valid iteration function $ITERATION(B, F)$ is empty if B is not equivalent to a body in F.

The allowed choices for the precondition B in Algorithm 4 are the same as in Algorithm 2. Lemma 42 proves that IT is the value of some valid iteration function $ITERATION(B, F)$ at the end of every iteration of Algorithm 4. Since G is initially empty and is added respectively IT or $ITERATION(B, F)$ at every step, the values of G are the same in the two algorithms.

Lemma 15 proves that Algorithm 1 can always choose a set B such that $ITERATION(B, F) = \emptyset$ if P does not contain a $<_F$-minimal body of F. Therefore, Algorithm 1 can always choose a set B so that it does not change G when it cannot choose a set B like Algorithm 2 does. The sequence of values of G is the same because the choices of the first kind do not change it and the choices of the second kind change it in the same way. $\square$

The correctness of the algorithm can now be proved.

Theorem 1 *The autoreconstruction algorithm (Algorithm 4), returns a single-head formula equivalent to the input formula if and only if the input formula has any.*

Proof. The proof comprises three steps: first, if F is single-head equivalent, the algorithm does not fail; second, the output formula (if any) is equivalent to F; third, if F is single-head equivalent the output formula is single-head. These three facts imply that the algorithm outputs a single-head version of F if any. The second implies that if F is not single-head equivalent the output formula is not single-head, if the algorithm does not fail.

The first step is that the algorithm does not fail if F is single-head equivalent. If F is single-head equivalent, it has a valid iteration function whose values are all single-head by Lemma 6. This is the precondition of Lemma 43, which proves that IT holds the value of a valid iteration function at some iteration of the internal loop of the algorithm. The definition of validity includes $SCL(B, F) \cup IT \models BCL(B, F)$. This entailment makes E true, avoiding failure.

The second step is that the output formula is equivalent to the input formula. Lemma 44 proves that the values of G in the algorithm are the same as the values in Algorithm 1 for a valid iteration function and some nondeterministic choices. This includes the last value, which is returned. Algorithm 1 returns a formula equivalent to F by Lemma 11.

The third step is that the output formula (if any) is single-head if the input formula is single-head equivalent. The algorithm has been proved to terminate two paragraphs above; this is the precondition of Lemma 44, which proves the validity of a certain iteration function. Its possible values are $\emptyset$ and the value of IT at the end of some iteration of the algorithm. Both are single-head: $\emptyset$ because it contains no clause, IT by construction. Since the output

formula is the same as that of Algorithm 1 with this iteration function by Lemma 44, and this iteration function has only single-head values, Lemma 13 applies: the output is single-head since F is by assumption equivalent to a single-head formula. $\qquad\square$

If the formula F is single-head equivalent the algorithm terminates and outputs a single-head formula equivalent to it. Otherwise, the algorithm does not output a single-head formula; it may output a formula that is not single-head or nothing at all because it fails.

4.7   Tautologies

The autoreconstruction algorithm (Algorithm 4) exploits $RCN(B, F)$ and $UCL(B, F)$. A previous article shows an algorithm `rcnucl()` for calculating $RCN(B, F)$ and $\{B' \to x \in F \mid F \models B \to B'\}$. The second is almost $UCL(B, F)$ but does not exclude tautologies. Since F is removed all tautologies in the first step of the autoreconstruction algorithm, this missing condition is moot as implied by $B' \to x \in F$.

Some optimizations below exploit $UCL(B, G \cup IT)$. The `rcnucl()` algorithm determines it if $G \cup IT$ does not contain tautologies, which is proved by the following lemma.

Lemma 45 *The set $G \cup IT$ does not contain tautologies during the execution of the autoreconstruction algorithm (Algorithm 4).*

Proof. The sets assigned to IT do not contain tautology because their definition includes $h_i \notin C_i$. Since G is initially empty, is only changed by the instruction $G = G \cup IT$ and IT does not contain tautology, it does not either. $\qquad\square$

4.8   Wild-goose chase

The algorithm requires $SCL(B, F)$ and $BCL(B, F)$. Being sets of consequences of F, they may be exponentially large. Their direct use is best avoided. The implementation of the algorithm in `singlehead.py` exploits three variants to this aim:

1. in place of $HEADS(B, F)$, whose definition involves the set of heads of $SCL(B, F)$, it uses the difference between $RCN(B, F)$ and the heads of G, the formula under construction;

2. in place of $UCL(B, F) \cap SCL(B, F)$ it uses $UCL(B, F) \cap U$ where U is the union of $UCL(B, F)$ for all bodies B of F used in the previous iterations of the algorithm;

3. $SCL(B, F) \cup IT \models BCL(B, F)$ is checked as $HCLOSE(RCN(B, G \cup IT), UCL(B, G \cup IT)) = HCLOSE(RCN(B, F), UCL(B, F))$.

In all three optimizations G takes the place of $SCL(B, F)$. This is correct as long as the algorithm mimics the reconstruction algorithm with a valid iteration function.

Lemma 46 *At every iteration of the autoreconstruction algorithm (Algorithm 4), $G \models SCL(B, F)$.*

Proof. By Lemma 44, the values of G in the autoreconstruction algorithm are the same as in Algorithm 1 for some nondeterministic choices and a valid iteration function. Lemma 10 proves $G \models SCL(B, F)$. ◻

The algorithm works by iteratively rebuilding the formula from the ground up—from a body A that do not entail any other to a body B that only entails A and so on. Each floor is built over the lower ones. If they are not, it cannot. They have to be finished and solid. In logical terms, processing B requires A to have already been processed, and its consequences already being entailed.

The first lemma states that the order of processing of bodies is correct.

Lemma 47 *If B is chosen in a given iteration of the autoreconstruction algorithm (Algorithm 4) and A' is the body of a clause in F such that $A' <_F B$, then a set A such that $A \equiv_F A'$ was chosen in a previous iteration.*

Proof. The algorithm only chooses sets of variables that are minimal in P. This means that when B is chosen, the set A' is not in P. Since A' is the body of a clause in F, it was initially in P. As a result, A' has been removed in a previous iteration. Let A be the set of variables chosen in that iteration. Since P is only removed elements by the statement $P = P \backslash \{B' \mid B \cup F \models B \equiv B'\}$, it holds $F \models A \equiv A'$, which is the same as $A \equiv_F A'$. ◻

If the algorithm is erecting a building, it needs all floors below B before building B. Floor B is not made in mid-air, it is made over them. The next lemma proves the converse: no floor that should be over B is built before it.

Lemma 48 *If A is chosen in an iteration of the autoreconstruction algorithm (Algorithm 4) that precedes the iteration in which B is chosen, then $B \not\leq_F A$.*

Proof. Since B is chosen after A, it is in P when A is chosen and at the end of that iteration. Since the algorithm only chooses minimal elements of P, it does not choose A if $B <_F A$. Since B is still in P at the end of the iteration, after $P = P \backslash \{B' \mid F \models B \equiv B'\}$, then $A \equiv_F B$ does not hold either. Since B is not strictly less than A and not equivalent to it, it is not less than or equal to it: $B \not\leq_F A$. ◻

Adding IT to G makes G entail every clause in $BCL(B, F)$ at the end of each iteration. As expected and wanted. What would be neither expected nor wanted is going overboard and adding clauses that are not in $BCL(B, F)$, or even worse not entailed by F. After all, the final aim is to reconstruct F in a controlled way. Controlled means single-head, but equivalence with F is essential: not producing anything not entailed by F.

This unwanted situation is first excluded for G.

Lemma 49 *At every iteration of the autoreconstruction algorithm (Algorithm 4) G is a subset of $\cup \{BCL(A, F) \mid B \not\leq_F A\}$.*

Proof. Let $A' \rightarrow x$ a clause of G. The set G is initially empty, and it is only changed by $G = G \cup IT$ at the end of every iteration of the external loop. Therefore, $A' \rightarrow x$ is in IT at the end of an iteration. Let A be the element of P chosen in Step 4a at the beginning of that iteration. Lemma 48 proves $B \not\leq_F A$. Lemma 41 proves $IT \subseteq BCL(A, F)$. ◻

Since G is not too large and IT is built appropriately as shown by Lemma 41, their union $G \cup IT$ is also correct.

Lemma 50 *At every iteration of the algorithm, $F \models G \cup IT$ holds.*

Proof. Lemma 41 proves that IT is a subset of $BCL(B, F)$, which is a set of consequences of F. Lemma 49 proves that G is contained in the union of some sets $BCL(A, F)$, which again only contain consequences of F. As a result, all clauses of $G \cup IT$ are entailed by F. $\square$

4.9 Head back

The autoreconstruction algorithm uses $HEADS(B, F) = RCN(B, F) \cap SFREE(B, F)$ as the heads of the clauses of IT. Since $SFREE(B, F)$ is defined in terms of $SCL(B, F)$, which is a set of consequences of F, it is expensive to determine. The implemented algorithm replaces it with $RCN(B, F)$ minus the heads of the clauses in G.

This is not always correct. Formula $F = \{a \to x, b \to x\}$ is a counterexample. The set B chosen in the first iteration may be $\{a\}$, and the algorithm sets $G = \{a \to x\}$. The second set B is then $\{b\}$. By definition, $SFREE(B, F)$ is the set of all variables that are not heads of $SCL(B, F)$, the clauses entailed by F whose body are less than B; the only variable $B = \{b\}$ entails is x, and no clause in F contains x negative; therefore, $SCL(B, F) = \emptyset$ and $SFREE(B, F) = \{a, b, x\}$. While $HEADS(B, F) = RCN(B, F) \cap SFREE(B, F) = \{x\} \cap \{a, b, x\} = \{x\}$, the heads of IT in the implemented algorithm are $RCN(B, F) \backslash \{x \mid \exists B.B \to x \in G\} = \emptyset$.

Granted, the counterexample formula is not single-head equivalent. And not by chance: the actual and computed heads only differ on formulae that are not single-head equivalent. Rather than insisting on the correct set of heads, efforts are better spent on proving that the final outcome of each iteration is correct: even when the algorithm uses the wrong sets of heads, it still disproves the formula single-head equivalent. This is exactly what happens for the counterexample formula: the set of heads for IT is empty while $BCL(B, F)$ contains $b \to x$. The iteration ends in failure, as required. This holds in general, not just in the counterexample.

The first step in this direction is that the heads of $RCN(B, F)$ minus the heads of G are all in $HEADS(B, F)$.

Lemma 51 *During the execution of the reconstruction algorithm (Algorithm 4) $H = RCN(B, F) \backslash \{x \mid \exists B'.B' \to x \in G\}$ is a subset of $HEADS(B, F)$.*

Proof. The claim is proved by showing that every variable not in $HEADS(B, F)$ is not in H either.

If x is not in $HEADS(B, F)$ then it is either not in $RCN(B, F)$ or not in $SFREE(B, F)$. In the first case, x is not in H because H is a subset of $RCN(B, F)$. In the second case, x is proved not to be in H by showing that it is the head of a clause of G.

The definition of $x \in SFREE(B, F)$ is that a clause $A' \to x$ is in $SCL(B, F)$. Being in $SCL(B, F)$ it is not a tautology: $x \notin A'$. By Lemma 46, G entails all of $SCL(B, F)$, including $A' \to x$. This entailment and $x \notin A'$ are the preconditions of Lemma 1, which tells that G contains a clause $A'' \to x$. Since x is the head of a clause of G, it is not in H. $\square$

This lemma proves that H is a subset of $HEADS(B, F)$. An expected consequence is that the sets IT built over H are also sets that can be built over $HEADS(B, F)$. This is however not the case because $MINBODIES(U, S)$ is not necessarily monotonic with respect

to its first argument. Yet, the change would preserve the property proved by Lemma 50: F entails these sets.

Lemma 52 *At every step of the autoreconstruction algorithm (Algorithm 4), F implies every clause with head in $H = RCN(B,F) \backslash \{x \mid \exists B'.B' \rightarrow x \in G\}$ and body in $MINBODIES(HCLOSE(H,UCL(B,F)), UCL(B,F) \cap SCL(B,F))$.*

Proof. By Lemma 51, H is a subset of $HEADS(B,F)$. By definition, $MINBODIES(HCLOSE(H,UCL(B,F)), UCL(B,F) \cap SCL(B,F))$ is a subset of its first argument $HCLOSE(H,UCL(B,F))$. The definition of this set includes H only as $x \in H \backslash B$, which implies $x \in HEADS(B,F) \backslash B$ since $H \subseteq HEADS(B,F)$: every clause of $HCLOSE(H,UCL(B,F))$ is in $HCLOSE(HEADS(B,F),UCL(B,F))$. This implies that $MINBODIES(HCLOSE(H,UCL(B,F)), UCL(B,F) \cap SCL(B,F))$ is a subset of $HCLOSE(HEADS(B,F),UCL(B,F))$. Lemma 40 proves that all non-tautological clauses with heads in $HEADS(B,F)$ and bodies in $HCLOSE(HEADS(B,F),UCL(B,F))$ are contained in $BLC(B,F)$ and are therefore entailed by F; the tautological ones are entailed because they are tautologies. $\qquad\square$

These two lemmas allow proving the main property of this section: $RCN(B,F)$ minus the heads of G is a correct replacement of $HEADS(B,F)$ in the autoreconstruction algorithm. More precisely, if they are not the same the autoreconstruction algorithm gives the correct answer anyway.

Lemma 53 *If $H = RCN(B,F) \backslash \{x \mid \exists B'.B' \rightarrow x \in G\}$ is not equal to $HEADS(B,F)$, the formula F is not single-head equivalent and the autoreconstruction algorithm (Algorithm 4) running with this modified definition of H fails.*

Proof. The modified and the original algorithms are identical as long as $H = RCN(B,F) \backslash \{x \mid \exists B'.B' \rightarrow x \in G\}$ and $HEADS(B,F)$ coincide. Up to the first moment these sets differ, the two algorithms are the same. Therefore, they possess the same properties. For example, $F \models G$ as proved in Lemma 50. At the first iteration where H and $HEADS(B,F)$ differ, the modified algorithm is proved to fail, but this is correct because the formula is not single-head equivalent. The rest of the proof is about this first iteration where H and $HEADS(B,F)$ differ.

The starting point is $H \subseteq HCLOSE(H,F)$ as proved by Lemma 51. The proof comprises four steps that follow from the assumption that H and $HEADS(B,F)$ differ:

1. some variables x of $HEADS(B,F)$ are the head of some clauses of G

2. if x is a variable of $HEADS(B,F)$ but not of H, and $A' \rightarrow x \in G$ for some A' then both $F \not\models A' \rightarrow B$ and $F \not\models B \rightarrow A'$ hold;

3. the algorithm fails;

4. F is not equivalent to any single-head formula.

The first step proves that if $HEADS(B,F)$ and H differ then some variables of $HEADS(B,F)$ are the head of some clauses of G.

44

Since H is subset of $HEADS(B, F)$, these two sets can only differ if $HEADS(B, F)$ contains a variable not in H. Being in $HEADS(B, F)$, such a variable is in $RCN(B, F)$. Since it is not H, which is $RCN(B, F)$ minus the heads of G, it is the head of a clause of G.

The second step proves that if x is a variable of $HEADS(B, F) \backslash H$, then all clauses $A' \to x \in G$ satisfy $F \not\models A' \to B$ and $F \not\models B \to A'$.

By Lemma 49, G is a subset of $\cup \{BCL(A, F) \mid B \not\leq_F A\}$. Therefore, $A' \to x$ is in $BCL(A, F)$ for some A such that $B \not\leq_F A$. By definition of $BCL(A, F)$ the equivalence $A \equiv_F A'$ holds. With $B \not\leq_F A$, it proves $B \not\leq_F A'$. This is the first property to prove: $F \not\models A' \to B$.

Since x is in $HEADS(B, F)$, it is in $SFREE(B, F)$. By definition, no clause $A' \to x$ with head x is in $SCL(B, F)$ exists. The definition of $A' \to x \in SCL(B, F)$ is $x \notin A'$, $F \models A' \to x$ and $A' <_F B$. If the first two conditions are true, the third is false. Since $A' \to x$ is in G, it is in $\cup \{BCL(A, F) \mid B \not\leq_F A\}$ by Lemma 49; since $BCL(A, F)$ does not include tautologies by definition, x is not in A'. Since $A' \to x \in G$ and $F \models G$ by Lemma 50, $F \models A' \to x$. As a result, $A' <_F B$ is false. Since $A' <_F B$ means $A' \leq_F B$ and $B \not\leq_F A'$, its negation is either $A' \not\leq_F B$ or $B \leq_F A'$. The second cannot because $B \not\leq_F A'$, as proved above. Therefore, $A' \not\leq B$. This implies $F \not\models B \to A'$, the second property to prove.

The third step proves that if $HEADS(B, F)$ and H differ the algorithm fails if it uses H in place of $HEADS(B, F)$.

As proved in the first step, if $HEADS(B, F)$ and H differ then a variable x in $HEADS(B, F)$ is the head of a clause of G. Since $HEADS(B, F)$ is a subset of $RCN(B, F)$, this variable x is in $RCN(B, F)$. By Lemma 18, a clause $B' \to x$ is in $BCL(B, F)$.

By contradiction, the algorithm is assumed not to fail. Since the algorithm fails if E is false, and E is set true only when $SCL(B, F) \cup IT \models BCL(B, F)$ holds, this entailment is the case for some IT. Since G entails $SCL(B, F)$ by Lemma 46, this implies $G \cup IT \models B' \to x$. The other premise $x \notin B'$ of Lemma 1 holds since $B' \to x$ is in $BCL(B, F)$, which does not contain tautologies by definition. The lemma proves the existence of a clause $B'' \to x \in G \cup IT$ such that $G \cup IT \models B' \to B''$. The former can be rewritten as: either $B'' \to x \in G$ or $B'' \to x \in IT$. The alternative $B'' \to x \in IT$ is false since the heads of IT are exactly H, and H does not contain any head of G by construction. As a result, $B'' \to x \in G$. As proved in the previous step, for all clauses $B'' \to x \in G$ it holds $F \not\models B \to B''$. Since $B' \to x \in BCL(B, F)$ includes $F \models B \equiv B'$, this is the same as $F \not\models B' \to B''$. Since F entails G by Lemma 50 and all clauses of IT by Lemma 52, the previously proved property $G \cup IT \models B' \to B''$ implies $F \models B' \to B''$.

This is a contradiction. The assumption was that the algorithm does not fail. Therefore, the algorithm fails.

This failure would be a problem if the formula were single-head equivalent, but this is proved not to be the case. This is the fourth and final step of the proof.

By assumption, $HEADS(B, F)$ and H differ. The first step of the proof proves $x \in HEADS(B, F)$ and $A' \to x \in G$ for a clause $A' \to x$. The second step proves $F \not\models B \to A''$ and $F \not\models A'' \to B$ for all clauses $A'' \to x \in G$ with head in $HEADS(B, F)$, including $A' \to x$. Since G does not contain tautologies by Lemma 45, x is not in A'. Since F entails G by Lemma 50, it entails its clause $A' \to x$.

Since x is in $HEADS(B, F)$, it is also in $RCN(B, F)$ by definition. By Lemma 18, $BCL(B, F)$ contains a clause $B' \to x$. This is defined as $x \notin B'$, $F \models B' \to x$ and $F \models B \equiv B'$.

The proof is by contradiction: F' is assumed to be a single-head formula equivalent to F. Because of equivalence, everything entailed by F is entailed by F'. In particular, F' entails $B' \to x$ and $A' \to x$. Since x is neither in B' nor in A', Lemma 1 applies: F contains two clauses $B'' \to x$ and $A'' \to x$ such that $F' \models B' \to B''$ and $F' \models A' \to A''$. Since F is removed tautologies in the first step of the algorithm, these are not tautologies: $x \notin A''$ and $x \notin B''$. Since F' is single-head, A'' and B'' are the same. Let $C = A'' = B''$. What holds for A'' or B'' also holds for C. Namely, $x \notin C$, $C \to x \in F'$, $F' \models B' \to C$ and $F' \models A' \to C$. Membership implies entailment: $F' \models C \to x$. By equivalence, F entails the same formulae: $C \to x$, $B' \to C$ and $A' \to C$.

If $F \models C \to B$, since $F \models A' \to C$ and $F \models B \equiv B'$, then $F \models A' \to B$, which is not the case. As a result, $F \not\models C \to B$. Since $F \models B \equiv B'$, the entailment $F \models B' \to C$ implies $F \models B \to C$. All conditions for $C \to x \in SCL(B, F)$ are met: $x \notin C$, $F \models C \to x$, $F \models B \to C$ and $F \not\models C \to B$.

Since G implies $SCL(B, F)$ by Lemma 46, it implies $C \to x$. Since x is not in C, Lemma 1 proves that G contains a clause $C' \to x$ such that $G \models C \to C'$, which implies $F \models C \to C'$ because F implies G by Lemma 50. Since $F \models B' \to C$ and $F \models B \equiv B'$, this implies $F \models B \to C'$. This is a contradiction since $x \in HEADS(B, F)$ and $C' \to x \in G$ imply $F \not\models B \to C'$ as proved in the second step of the proof. $\square$

In summary, even if H differs from $HEADS(B, F)$ the algorithm is still right: it fails, which is what it should do since the formula is not single-head equivalent. This allows using H instead of $HEADS(B, F)$, saving the calculation of $SCL(B, F)$.

A positive side effect of the change is that success is no longer ambiguous. If F is not single-head equivalent, the original algorithm may fail or return a formula that is not single-head. The modified version fails, period.

Lemma 54 *If F is not equivalent to any single-head formula, Algorithm 4 with the variant $H = RCN(B, F) \backslash \{x \mid \exists B'.B' \to x \in G\}$ fails.*

Proof. A difference between $RCN(B, F) \backslash \{x \mid \exists B'.B' \to x \in G\}$ and $HEADS(B, F)$ makes the modified algorithm fail, as shown by Lemma 53. The claim is therefore proved in this case. Remains to prove it when $RCN(B, F) \backslash \{x \mid \exists B'.B' \to x \in G\}$ always coincides with $HEADS(B, F)$.

This is proved by contradiction: the algorithm is assumed not to fail and $RCN(B, F) \backslash \{x \mid \exists B'.B' \to x \in G\}$ to always coincide with $HEADS(B, F)$; the input formula is proved single-head equivalent.

Since $RCN(B, F) \backslash \{x \mid \exists B'.B' \to x \in G\}$ is the same as $HEADS(B, F)$, the modified algorithm is the same as the original (Algorithm 4). Since the modified algorithm does not fail, the original does not fail either. By Theorem 1, the output formula of the original algorithm is equivalent to F. This is therefore also the output formula of the modified algorithm. The input formula is proved single-head equivalent by showing that the output formula is single-head.

The claim is proved by inductively showing that G never contains two clauses with the same head x. It initially holds because G is empty. It is inductively assumed at the beginning

of every iteration and proved at the end. Two cases are considered: G does not contain a clause with head x at the beginning of the iteration, or it does.

The first case is that G does not contain a clause of head x. The only instruction that changes G is $G = G \cup IT$. Since G does not contain any clause of head x and IT may at most contain one by construction, $G \cup IT$ may at most contain one clause of head x. This is the inductive claim.

The second case is that G contains a clause of head x. It implies that x is not in $H = RCN(B, F) \backslash \{x \mid \exists B'.B' \rightarrow x \in G\}$. Since H is the set of heads of IT, this set does not contain any clause of head x. The instruction $G = G \cup IT$ does not add any clause of head x to G, which still contains only one clause of head x after it. $\square$

In the other way around, if the algorithm succeeds the input formula is single-head equivalent. The converse is proved by Theorem 1. In the modified version of the algorithm, success and failure alone tell whether the input formula is single-head or not.

4.10 Old bodies

The algorithm uses $UCL(B, F) \cap SCL(B, F)$ as the second argument of $MINBODIES()$. The first component $UCL(B, F)$ is already known: it is calculated in advance together with $RCN(B, F)$ for all bodies B in F. The second component $SCL(B, F)$ is instead a problem because it is not bounded by the size of F. The intersection can be rewritten as $\{B' \rightarrow x \in UCL(B, F) \mid B' <_F B\}$, which however requires two entailment tests for each clause: $F \models B \rightarrow B'$ and $F \not\models B' \rightarrow B$.

A more efficient way to obtain it is to accumulate all clauses $UCL(B, F)$ along the way in a set U. During the iteration for B, this set U contains all clauses of F with a body that is strictly less than B. It may also contain other clauses, but intersecting with $UCL(B, F)$ removes them.

The algorithm requires only small, localized changes: at the beginning, U is empty; at the end of each step, $UCL(B, F)$ is added to U by $U = U \cup UCL(B, F)$; finally, $UCL(B, F) \cap U$ replaces $UCL(B, F) \cap SCL(B, F)$.

Lemma 55 *If $B' \rightarrow x \in UCL(B, F)$, then $B' \rightarrow x \in SCL(B, F)$ if and only $B' \rightarrow x \in U$, where U is the union of $UCL(A, F)$ for all sets A used in the previous iterations of the autoreconstruction algorithm (Algorithm 4).*

Proof. The proof comprises two parts: $B' \rightarrow x \in UCL(B, F) \cap U$ implies $B' \rightarrow x \in SCL(B, F)$, and $B' \rightarrow x \in UCL(B, F) \cap SCL(B, F)$ implies $B' \rightarrow x \in U$.

The first part starts from the assumptions $B' \rightarrow x \in UCL(B, F)$ and $B' \rightarrow x \in U$. The first condition is defined as $B' \rightarrow x \in F$, $F \models B \rightarrow B'$ and $x \notin B'$, the second as $B' \rightarrow x \in UCL(A, F)$ with A chosen before B. That A was chosen before B has two consequences: first, $B <_F A$ is not the case as otherwise A would not have been minimal in P when it was chosen; second, $B \equiv_F A$ does not hold either, as otherwise B would have been removed from P in the iteration for A. Since neither $B <_F A$ nor $B \equiv_F A$ hold, $B \leq_F A$ does not hold either. By definition, $F \not\models A \rightarrow B$ follows. The condition $B' \rightarrow x \in UCL(A, F)$ implies $F \models A \rightarrow B'$. If $F \models B' \rightarrow B$ then by transitivity $F \models A \rightarrow B$, which contradicts $F \not\models A \rightarrow B$. Therefore, the converse $F \not\models B' \rightarrow B$ holds. The conditions $F \not\models B' \rightarrow B$, $F \models B \rightarrow B'$, $B' \rightarrow x \in F$ and $x \notin B'$ imply $B' \rightarrow x \in SCL(B, F)$.

The reverse direction is proved by assuming $B' \to x \in UCL(B,F)$ and $B' \to x \in SCL(B,F)$. These conditions imply $x \notin B'$, $B' \to x \in F$, $F \models B \to B'$ and $F \not\models B' \to B$. The latter two define $B' <_F B$. Since B' is the precondition of a clause of F, it is initially in P. When B is chosen it is no longer in P, as otherwise B would not be minimal in P. Therefore, B' has been removed from P in a previous step. Let A be the set of variables chosen at that step. The only instruction that removes elements from P is $P = P\backslash\{B' \mid F \models A \equiv B\}$. Therefore, $F \models A \equiv B'$. Equivalence implies entailment: $F \models A \to B'$; with $x \notin B'$ and $B' \to x \in F$, this is the definition of $B' \to x \in UCL(A,F)$. Therefore, $B' \to x$ is added to U at the end of the iteration. Since U is never removed element, it still contains that clause at the iteration when B is chosen. $\qquad\square$

The instruction $T = MINBODIES(M, UCL(B,F) \cap SCL(B,F))$ can therefore be replaced by $T = MINBODIES(M, UCL(B,F) \cap U)$ with the only additional cost of accumulating $UCL(B,F)$ in U. This is correct because the lemma proves that as long as clauses of $UCL(B,F)$ are concerned, there is no difference between $SCL(B,F)$ and U.

4.11 Cherry picking

The check $SCL(B,F) \cup IT \models BCL(B,F)$ is the most expensive operation in the inner loop of the algorithm. As a dominating operation, it should be as efficient as possible. Instead, it requires two sets that are not even bounded by the size of the original formula: $SCL(B,F)$ and $BCL(B,F)$; they contain consequences of F, not only clauses of F. This section reformulates the check to remove $SCL(B,F)$, the next $BCL(B,F)$.

Since the formula under construction G entails $SCL(B,F)$ by Lemma 46, it could be used in its place. But while G contains a subset equivalent to $SCL(B,F)$, it also contains other clauses. In particular, it contains clauses that are not in $BCL(B,G)$ and are therefore irrelevant when the precondition is B. This is a setback in efficiency when F contains many bodies that do not entail each other. Restricting to $UCL(B,G)$ excludes the extra clauses, but is not correct in general.

$$F = \{ab \to c, c \to d\}$$

The algorithm chooses $B = \{c\}$ as the first body since it is entailed by the other $\{a,b\}$ but not the other way around. The first iteration ends with $G = \{c \to d\}$. The algorithm then moves onto $B = \{a,b\}$. The set $UCL(B,G)$ contains the clauses of G whose body is entailed by B according to G. The key here is "according to G": since G does not contain $ab \to c$, the clause $c \to d$ does not have its body entailed by $B = \{b,c\}$. Since it is the only clause of G, it follows that $UCL(B,G)$ is empty. Since IT cannot contain a clause with head d, the clause $ab \to d \in BCL(B,F)$ cannot be entailed by $UCL(B,G) \cup IT$.

The problem is "according to G". The formula under construction is still too weak. Selecting clauses from G is correct, but using G for deciding body entailment is not. The clause $c \to d$ is expected to come out from this selection because its body c is entailed by $B = \{a,b\}$ and is therefore relevant to the reconstruction of the clauses whose body is entailed by B. However, c is entailed by B only according to F, not to G.

The solution is to use G as the source of clauses and F for entailment between bodies. The first results are established on this modified selection function.

$$UCL(B, G, F) = \{B' \to x \in G \mid F \models B \to B'\}$$

The condition $x \notin B'$ is not necessary because G is already proved not to contain tautologies by Lemma 45.

This set is equivalent to $SCL(B, F)$. This is proved in two parts: first, $UCL(B, G, F)$ is a subset of $SCL(B, F)$; second, it entails it.

Lemma 56 *At every iteration of the algorithm (Algorithm 4), $UCL(B, G, F) \subseteq SCL(B, F)$ holds.*

Proof. The definition of $A' \to x \in UCL(B, G, F)$ is $A' \to x \in G$ and $F \models B \to A'$. By Lemma 49, $A' \to x \in G$ implies $A' \to x \in BCL(A, F)$ for some A such that $B \not\leq_F A$. The definition of $A' \to x \in BCL(A, F)$ is $x \notin A'$, $F \models A' \to x$ and $F \models A \equiv A'$. Since $B \not\leq_F A$, the latter implies $B \not\leq_F A'$, which means $F \not\models A' \to B$. All parts of the definition of $A' \to x \in SCL(B, F)$ are proved: $x \notin A'$, $F \models A' \to x$, $F \models B \to A'$ and $F \not\models A' \to B$. □

This lemma trivially implies $SCL(B, F) \models UCL(B, G, F)$. The converse entailment requires a preliminary lemma that shows that G and $UCL(B, G, F)$ are the same when checking entailment of clauses whose body is entailed by B.

Lemma 57 *For every two formulae F and G, if $G \models A \to x$, $F \models G$ and $F \models B \to A$ then $UCL(B, G, F) \models A \to x$.*

Proof. Proved by induction on the number of variables of G. The base case is when G only contains a single variable. The only definite clauses of one variable are $x \to x$ and $\emptyset \to x$. The first is a tautology and is therefore entailed by $UCL(B, G, F)$. The second is in $UCL(B, G, F)$ since $F \models B \to \emptyset$.

The induction conclusion is that $UCL(B, G, F) \models A \to x$ follows from $G \models A \to x$, $F \models G$ and $F \models B \to A$. If x is in A then $A \to x$ is a tautology and is therefore entailed by $UCL(B, G, F)$. Otherwise, $x \notin A$ and $G \models A \to x$ are the premises of Lemma 1, which proves the existence of a clause $A' \to x \in G$ such that $G^x \models A \to A'$. Since F entails G it also entails its subset G^x and its consequences, including $A \to A'$. This entailment $F \models A \to A'$ forms with $A' \to x \in G$ the definition of $A' \to x \in UCL(B, G, F)$.

Since G^x entails $A \to A'$, it entails $A \to a$ for every $a \in A'$. The other two assumptions of the induction hypothesis have already been proved: $F \models G^x$ and $F \models B \to A$. Its conclusion is $UCL(B, G^x, F) \models A \to a$. Since this is the case for each $a \in A'$, $UCL(B, G^x, F) \models A \to A'$ follows. By construction, $UCL(B, G^x, F)$ is a subset of $UCL(B, G, F)$ since the formula F that dictates with clauses of G or G^x are selected is the same and G^x is a subset of G. As a result, $UCL(B, G, F) \models UCL(B, G^x, F)$. Therefore, $UCL(B, G, F) \models A \to A'$. Along with $A' \to x \in UCL(B, G, F)$, this implies $UCL(B, G, F) \models A \to x$. □

This lemma allows proving the converse entailment of Lemma 56.

Lemma 58 *At every iteration of the algorithm, $UCL(B, G, F) \models SCL(B, F)$.*

Proof. Let $A \to x \in SCL(B, F)$. The claim is that $UCL(B, G, F) \models A \to x$. By Lemma 46, $A \to x \in SCL(B, F)$ implies $G \models A \to x$. The definition of $A \to x \in SCL(B, F)$ includes $F \models B \to A$. Since $F \models G \cup IT$ by Lemma 50, also $F \models G$ holds. All preconditions of Lemma 57 are met. Its conclusion is $UCL(B, G, F) \models A \to x$. □

The final destination of this three lemmas is proving that $UCL(B, G, F)$ and $SCL(B, F)$ are equivalent in the algorithm. This is an obvious consequence of the above lemmas, and is formally stated in the following.

Lemma 59 *At every iteration of the autoreconstruction algorithm (Algorithm 4), $UCL(B, G, F) \equiv SCL(B, F)$ holds.*

Proof. By Lemma 56, $UCL(B, G, F) \subseteq SCL(B, F)$, which implies $SCL(B, F) \models UCL(B, G, F)$. Lemma 58 proves the other direction: $UCL(B, G, F) \models SCL(B, F)$. Implication in both directions is equivalence. □

This lemma could directly be applied to the program by checking $UCL(B, G, F) \cup IT \models BCL(B, F)$ instead of $SCL(B, F) \cup IT \models BCL(B, F)$ and modifying the function that calculates $UCL(B, F)$ by the addition of a third argument. This is however not necessary. While $SCL(B, F)$ is not equivalent to G, the whole formula $SCL(B, F) \cup IT$ can be replaced by $UCL(B, G \cup IT)$.

These formulae are not equivalent. Yet, they are the same as long as entailment of $BCL(B, F)$ is concerned. This is proved in two parts: first, $UCL(B, G \cup IT) \models BCL(B, F)$ implies $SCL(B, F) \cup IT \models BCL(B, F)$; second, the converse.

The assumption of the first part is $UCL(B, G \cup IT) \models BCL(B, F)$. Its claim is $SCL(B, F) \cup IT \models BCL(B, F)$. The first step is that IT can be factored out from $UCL(B, G \cup IT)$.

Lemma 60 *At every iteration of the autoreconstruction algorithm (Algorithm 4), $SCL(B, F) \cup IT \equiv UCL(B, G \cup IT, F)$ holds.*

Proof. An immediate consequence of the definition is $UCL(B, G \cup IT, F) = UCL(B, G, F) \cup UCL(B, IT, F)$.

By Lemma 59, $UCL(B, G, F)$ is equivalent to $SCL(B, F)$.

Since $IT \subseteq BCL(B, F)$ by Lemma 41, it holds $F \models B \to B'$ for all clauses $B' \to x$ of IT. As a result, $UCL(B, IT, F)$ contains all of them: $UCL(B, IT, F) = IT$. □

Since $SCL(B, F) \cup IT$ is equivalent to $UCL(B, G \cup IT, F)$, if the latter is equivalent to $UCL(B, G \cup IT, G \cup IT)$, the target is hit because this formula is the same as $UCL(B, G \cup IT)$.

The difference between $UCL(B, G \cup IT, F)$ and $UCL(B, G \cup IT, G \cup IT)$ is the selection criteria: $F \models B \to B'$ versus $G \cup IT \models B \to B'$ where B' is the body of a clause. These conditions differ in general. Yet, they coincide under the current assumption $UCL(B, G \cup IT) \models BCL(B, F)$.

Lemma 61 *If $UCL(B, G \cup IT) \models BCL(B, F)$ then $F \models B \to B'$ is equivalent to $G \cup IT \models B \to B'$ for every set of variables B'.*

Proof. By Lemma 50 and Lemma 52, F entails $G \cup IT$. As a result, if $G \cup IT \models B \rightarrow B'$, then $F \models B \rightarrow B'$.

The rest of the proof obtains $G \cup IT \models B \rightarrow B'$ from $F \models B \rightarrow B'$. Since $F \models B \rightarrow B'$, then $F \models B \rightarrow b$ holds for every $b \in B'$. If $b \in B$ then $B \rightarrow b$ is a tautology and is therefore entailed by $BCL(B, F)$. Otherwise, $b \notin B$ and $F \models B \rightarrow b$ with the trivially true $F \models B \equiv B$ define $B \rightarrow b \in BCL(B, F)$. Either way, $B \rightarrow b$ is entailed by $BCL(B, F)$ for every $b \in B'$. By assumption, $UCL(B, G \cup IT) \models BCL(B, F)$, which implies $G \cup IT \models B \rightarrow b$ since $UCL(B, G \cup IT)$ is defined as a subset of $G \cup IT$. This holds for every $b \in B'$, proving $G \cup IT \models B \rightarrow B'$. $\qquad\square$

The first part of the proof can now be completed.

Lemma 62 *If $UCL(B, G \cup IT) \models BCL(B, F)$ then $SCL(B, F) \cup IT \models BCL(B, F)$.*

Proof. The only difference between $UCL(B, G \cup IT)$ and $UCL(B, G \cup IT, G \cup IT)$ is that the former does not contain tautologies. The latter does not as well because it is a subset of $G \cup IT$, which does not by Lemma 45.

The set $UCL(B, G \cup IT, G \cup IT)$ contains some clauses $G \cup IT$ like $UCL(B, G \cup IT, F)$ does, but differs from it in the selection criteria: $G \cup IT \models B \rightarrow B'$ instead of $F \models B \rightarrow B'$. The assumption $UCL(B, G \cup IT) \models BCL(B, F)$ proves that F and $G \cup IT$ entail the same clauses $B \rightarrow B'$ by Lemma 61. Therefore, $UCL(B, G \cup IT, G \cup IT)$ is identical to $UCL(B, G \cup IT, F)$.

Lemma 60 proves that $UCL(B, G \cup IT, F)$ is equivalent to $SCL(B, F) \cup IT$.

Summarizing, the assumption $UCL(B, G \cup IT) \models BCL(B, F)$ implies that $UCL(B, G \cup IT)$ is the same as $UCL(B, G \cup IT, F)$, which is the same as $SCL(B, F) \cup IT$; this equality and the assumption $UCL(B, G \cup IT) \models BCL(B, F)$ imply $SCL(B, F) \cup IT \models BCL(B, F)$. $\square$

The proof looks like a sequence of equivalences. As such, it would also prove the opposite direction, saving the cost of a further proof. But this is a false impression: $UCL(B, G \cup IT)$ is not always equivalent to $SCL(B, F) \cup IT$, it is only when $UCL(B, G \cup IT) \models BCL(B, F)$ holds.

The converse direction has to be proved separately: $SCL(B, F) \cup IT \models BCL(B, F)$ implies $UCL(B, G \cup IT) \models BCL(B, F)$.

The first step in this direction is that $UCL()$ maintains certain entailments of the formula it is applied to.

Lemma 63 *For every formulae F and G, variable x and sets of variables B and B' such that $F \models G$ and $F \models B \rightarrow B'$, it holds $G \models B' \rightarrow x$ if and only if $UCL(B, G, F) \models B' \rightarrow x$.*

Proof. Since $UCL(B, G, F)$ only comprises clauses of G by definition, it is entailed by G. This proves the first direction of the claim: if $UCL(B, G, F) \models B' \rightarrow x$ then $G \models B' \rightarrow x$.

The converse direction is proved by contradiction: $G \models B' \rightarrow x$ and $UCL(B, G, F) \not\models B' \rightarrow x$ do not both hold at the same time. The latter is equivalent to the existence of a model M that satisfies $UCL(B, G, F) \cup B' \cup \{\neg x\}$.

Let M' be the model that evaluates all variables of $BCN(B, F)$ like M and all others to false: if $F \models B \rightarrow z$ then z has the same value in M and M'; otherwise, z is false in M'. This model M' is proved to satisfy every element of $G \cup B' \cup \{\neg x\}$, contradicting the

assumption $G \models B' \to x$. The proof is done separately for the elements of $G \backslash UCL(B, G, F)$, of $UCL(B, F, F)$, of B' and of $\{\neg x\}$.

Let $B'' \to y$ be a clause of G but not of $UCL(B, G, F)$. Since $UCL(B, G, F)$ contains all clauses of G except those whose body is not entailed by B, this is only possible if $F \not\models B \to B''$. As a result, $F \not\models B \to b$ for some $b \in B''$. By construction, M' evaluates b to false and therefore satisfies $B'' \to y$.

The definition of $B'' \to y \in UCL(B, G, F)$ is $B'' \to y \in G$ and $F \models B \to B''$. The latter implies $F \models B \to b$ for every $b \in B''$. Therefore, M' evaluates every $b \in B''$ in the same way as M does. Since $F \models G$ holds by Lemma 50, $B'' \to y \in G$ implies $F \models B'' \to y$. With $F \models B \to B''$, it implies $F \models B \to y$. Therefore, y is evaluated by M' in the same way as M does. This proves that all variables in every $B'' \to y \in UCL(B, G, F)$ is evaluated by M' in the same way as M does. Since M satisfies $UCL(B, G, F)$, also M' does.

Since $B' \to b$ is a tautology for every $b \in B$, it is entailed by F. Which also entails $B \to B'$. By transitivity, F entails $B \to b$ for every $b \in B'$. As a result, M' evaluates all variables of B' in the same way as M. Since M satisfies B' by assumption, also M' does.

A consequence of $F \models B \to B'$ and $G \models B' \to x$ is $F \models B \to x$ since $F \models G$. It implies that M' evaluates $\neg x$ in the same way as M, which satisfies it.

This proves that M' satisfies $G \backslash UCL(B, G, F) \cup UCL(B, G, F) \cup B' \cup \{\neg b\}$, which is the same as $G \cup B' \cup \{\neg x\}$. The satisfiabily of this set is the same as $G \not\models B' \to x$, which contradicts the assumption $G \models B' \to x$. □

This lemma tells that $UCL(B, G, F)$ is the same as G when checking entailment of clauses whose body is entailed by B according to F. It also applies to the case where the two formulae G and F are the same.

Lemma 64 *If $SCL(B, F) \cup IT \models BCL(B, F)$ then $UCL(B, G \cup IT) \models BCL(B, F)$.*

Proof. Lemma 46 proves $G \models SCL(B, F)$. With the assumption $SCL(B, F) \cup IT \models BCL(B, F)$, it implies $G \cup IT \models BCL(B, F)$.

Let $B' \to x \in BCL(B, F)$. By definition, $F \models B \to B'$. This implies $F \models B \to b$ for every $b \in B'$. If $b \in B$ then $B \to b$ is a tautology and is therefore entailed by $BCL(B, F)$. Otherwise, $b \notin B$, $F \models B \to b$ and the trivially true $F \models B \to B$ imply $B \to b \in BCL(B, F)$. Either way, $B \to b$ is entailed by $BCL(B, F)$. Since $G \cup IT$ entails this set, it also entails $B \to b$ as well. This hold for all $b \in B'$; therefore, $G \cup IT \models B \to B'$.

Lemma 63 applies to $UCL(B, G \cup IT, G \cup IT)$: since $G \cup IT \models G \cup IT$, $G \cup IT \models B \to B'$ and $G \cup IT \models B' \to x$, it follows $UCL(B, G \cup IT, G \cup IT) \models B' \to x$. This is the same as $UCL(B, G \cup IT) \models B' \to x$ since the only difference between the definitions of $UCL(B, G \cup IT, G \cup IT)$ and $UCL(B, G \cup IT)$ is that the latter does not contain tautologies, but the former does not as well since it is a subset of $G \cup IT$, which does not by Lemma 45. □

This lemma proves that $SCL(B, F) \cup IT \models BCL(B, F)$ implies $UCL(B, G \cup IT) \models BCL(B, F)$. The converse implication is Lemma 62. The conclusion is that these two conditions are the same.

Lemma 65 *The conditions $SCL(B, F) \cup IT \models BCL(B, F)$ and $UCL(B, G \cup IT) \models BCL(B, F)$ are equivalent.*

Proof. Immediate consequence of Lemma 62 and Lemma 64. □

4.12   Ubi minor major cessat

The previous section proves that the dominant operation of the algorithm $SCL(B,F) \cup IT \models BCL(B,F)$ is equivalent to $UCL(B, G \cup IT) \models BCL(B,F)$. While $SCL(B,F)$ is a set of consequences and can therefore be very large, $UCL(B, G \cup IT)$ is a subset of $G \cup IT$ and is therefore bounded by the size of the formula under construction G and the candidate set IT. The latter is in turn bounded because it is single-head: it has no more clauses than the variables in F.

The problem remains on the other side of the implication: $BCL(B,F)$ contains all clauses $B' \to x$ entailed by F such that $F \models B \to B'$. Many such clauses may be entailed even from a small formula F.

The solution is to use $UCL(B,F)$ in place of $BCL(B,F)$. This is the set of clauses of F whose body is entailed by B according to F. It is monotonic with respect to entailment of the formula, as the following lemma proves.

Lemma 66 *If $F \models F'$ then $UCL(B,F) \models UCL(B,F')$.*

Proof. Every clause $B' \to x$ of $UCL(B,F')$ satisfies $B' \to x \in F'$ and $F' \models B \to B'$ by definition. The former implies $F' \models B' \to x$. Since $F \models F'$, these two entailments carry over to the other formula: $F \models B' \to x$ and $F \models B \to B'$. By Lemma 36, these two conditions imply $UCL(B,F) \models B' \to x$. This argument shows that every clause of $UCL(B,F')$ is entailed by $UCL(B,F)$. □

Another result about $UCL()$ is that $RCN()$ is invariant with respect to replacing F with $UCL(B,F)$, as it is $BCN()$ as proved by Lemma 34.

Lemma 67 *For every formula F and set of variables B it holds $RCN(B,F) = RCN(B, UCL(B,F))$*

Proof. The first step of the proof is $F \models B \to (BCN(B,F) \backslash \{x\})$. The definition of $y \in BCN(B,F)$ is $F \models B \to y$. This is the case for every $y \in BCN(B,F)$; therefore, $F \models B \to BCN(B,F)$ follows. Since $BCN(B,F) \backslash \{x\}$ is a subset of $BCN(B,F)$, also $F \models B \to (BCN(B,F) \backslash \{x\})$ holds.

This entailment is the precondition of Lemma 36, which proves that F and $UCL(B,F)$ entail the same clauses whose body is entailed by B. In this case, $F \models B \to (BCN(B,F) \backslash \{x\})$ proves that $F \models (BCN(B,F) \backslash \{x\}) \to x$ is the same as $UCL(B,F) \models (BCN(B,F) \backslash \{x\}) \to x$. These two entailments define $x \in RCN(B,F)$ and $x \in RCN(B, UCL(B,F))$. Since they are the same, $RCN(B,F)$ and $RCN(B, UCL(B,F))$ coincide. □

This lemma allows proving $UCL(B,F)$ can replace $BCL(B,F)$ in a certain situation.

A semantical counterpart of Lemma 31 shows that $UCL(B,F)$ is equivalent to the union of the two other considered sets of clauses.

Lemma 68 *For every formula F and set of variables B, it holds $UCL(B,F) \equiv SCL(B,F) \cup BCL(B,F)$.*

Proof. Lemma 31 proves $UCL(B,F) = F \cap (SCL(B,F) \cup BCL(B,F))$. As a subset of $SCL(B,F) \cup BCL(B,F)$, the set $UCL(B,F)$ is entailed by it: $SCL(B,F) \cup BCL(B,F) \models UCL(B,F)$.

The converse implication $UCL(B, F) \models SCL(B, F) \cup BCL(B, F)$ is proved by showing that every clause $B' \rightarrow x$ of $SCL(B, F) \cup BCL(B, F)$ is entailed by $UCL(B, F)$. The definition of $B' \rightarrow x \in SCL(B, F) \cup BCL(B, F)$ includes $F \models B' \rightarrow x$ and either $B' <_F B$ or $B' \equiv_F B$. The two alternatives are equivalent to $B' \leq_F B$, which is defined as $F \models B \rightarrow B'$. Lemma 36 proves that $F \models B \rightarrow B'$ and $F \models B' \rightarrow x$ imply $UCL(B, F) \models B' \rightarrow x$. $\square$

Lemma 46 proves that G entails $SCL(B, F)$. This fact cannot be pushed from G to $UCL(B, G)$ since $UCL(B, G)$ does not contain the clauses of G whose body is not entailed by G. Adding IT and requiring $BCL(B, F)$ to be entailed fills the gap.

Lemma 69 *If $UCL(B, G \cup IT) \models BCL(B, F)$ then $UCL(B, G \cup IT) \models SCL(B, F)$.*

Proof. The claim is proved by showing that every clause $B' \rightarrow x$ of $SCL(B, F)$ is entailed by $UCL(B, G \cup IT)$.

The definition of $B' \rightarrow x \in SCL(B, F)$ includes $B' <_F B$, which includes $F \models B \rightarrow B'$. This is the same as $F \models B \rightarrow b$ for every $b \in B'$. If $b \in B$ then $B \rightarrow b$ is a tautology and is therefore entailed by $BCL(B, F)$. Otherwise, $b \notin B$ and $F \models B \rightarrow b$ and the trivially true $F \models B \equiv B$ define $B \rightarrow b \in BCL(B, F)$. Either way, $BCL(B, F)$ entails $B \rightarrow b$. Since this is the case for every $b \in B'$, it proves $BCL(B, F) \models B \rightarrow B'$. Since $UCL(B, G \cup IT)$ entails $BCL(B, F)$ by assumption, it also entails its consequence $B \rightarrow B'$. Since $UCL(B, G \cup IT)$ is by definition a subset of $G \cup IT$, monotonicity implies $G \cup IT \models B \rightarrow B'$.

Lemma 46 proves $G \models SCL(B, F)$. Therefore, $G \models B' \rightarrow x$. By monotonicity, $G \cup IT \models B' \rightarrow x$. Since $G \cup IT \models B \rightarrow B'$ as proved in the previous paragraph, $G \cup IT \models B' \rightarrow x$ implies $UCL(B, G \cup IT) \models B' \rightarrow x$ by Lemma 36. $\square$

If $UCL(B, G \cup IT)$ implies $BCL(B, F)$, it also implies $SCL(B, F)$. If it implies the first, it implies both. Their union is $SCL(B, F) \cup BCL(B, F)$, which is semantically the same as $UCL(B, F)$. This is proved by the next lemma.

Lemma 70 *The conditions $UCL(B, G \cup IT) \models BCL(B, F)$ and $UCL(B, G \cup IT) \equiv UCL(B, F)$ are equivalent.*

Proof. Lemma 50 states $F \models G \cup IT$. This is the precondition of Lemma 66, which proves $UCL(B, F) \models UCL(B, G \cup IT)$. This is half of the second condition in the claim, $UCL(B, G \cup IT) \equiv UCL(B, F)$. It holds regardless of the first.

The claim is therefore proved if the first condition $UCL(B, G \cup IT) \models BCL(B, F)$ is the same as the other half of the second: $UCL(B, G \cup IT) \models UCL(B, F)$.

Lemma 68 proves $UCL(B, F) \equiv SCL(B, F) \cup BCL(B, F)$; therefore, the claim is that $UCL(B, G \cup IT) \models BCL(B, F)$ and $UCL(B, G \cup IT) \models SCL(B, F) \cup BCL(B, F)$ are the same. The first entailment follows from the second because its consequent is a subset of that of the second. The converse is a consequence of Lemma 69: $UCL(B, G \cup IT) \models BCL(B, F)$ implies $UCL(B, G \cup IT) \models SCL(B, F)$. Therefore, it implies $UCL(B, G \cup IT) \models SCL(B, F) \cup BCL(B, F)$. $\square$

Lemma 65 proves that $SCL(B, F) \cup IT \models BCL(B, F)$ is the same as $UCL(B, G \cup IT) \models BCL(B, F)$, which Lemma 70 proves the same as $UCL(B, G \cup IT) \equiv UCL(B, F)$. The formulae in the equivalence are both bounded in size by F and $G \cup IT$; the latter is single-head by construction, and is therefore polynomially bounded by the number of variables.

Yet, equivalence is still to be checked. It is the same as mutual inference: $UCL(B, G \cup IT)$ entails every clause of $UCL(B, F)$ and vice versa. Each inference check is linear in time, making the total quadratic.

An alternative is to trade space for time. Equivalence is equality in some cases. For example, if both A and B are deductively closed, they are equivalent if and only if they coincide. Equivalence is equality on the deductive closure of the two formulae.

Checking equality is easy, but the deductive closure may be very large. Working on the sets of prime implicants is a more efficient alternative. In the present case, a still better alternative is to use the minimal implicants with heads in H. The formula under construction G already entails $SCL(B, F)$; the target of the current iteration is to entail $BCL(B, F)$ with clauses that have heads in H. Only these clauses matter. The minimal such clauses are $HCLOSE()$.

The final step of this section is to prove that equality is enough if checked on $HCLOSE()$ when its first argument is $RCN(B, F)$ and the second is either $UCL(B, G \cup IT)$ or $UCL(B, F)$. These formulae have something in common: the premise of the following lemma.

Lemma 71 *If* $F \models B \to B'$ *holds for every clause* $B' \to x \in F$, *then* $F \equiv HCLOSE(RCN(B, F), F)$.

Proof. The definition of $B' \to x \in HCLOSE(H, F)$ includes $F \models B' \to x$: every clause of $HCLOSE(H, F)$ is entailed by F. This also happens in the particular case $H = RCN(B, F)$, and proves $F \models HCLOSE(RCN(B, F), F)$ holds even when the premise of the lemma does not.

The premise is necessary to the converse implication $HCLOSE(RCN(B, F), F) \models F$. Every clause $B' \to x$ of F is proved to be entailed by $HCLOSE(RCN(B, F), F)$.

The premise of the lemma is that $F \models B \to B'$ holds for every clause $B' \to x \in F$. It implies $B' \subseteq BCN(B, F)$. If $x \in B'$ then $B' \to x$ is tautological and is therefore entailed by $HCLOSE(RCN(B, F), F)$. Otherwise, $x \notin B'$ strengthens $B' \subseteq BCN(B, F)$ to $B' \subseteq BCN(B, F) \backslash \{x\}$. Since F contains $B' \to x$ it also entails it: $F \models B' \to x$. Monotonicity implies $F \models (BCN(B, F) \backslash \{x\}) \to x$. This is the definition of $x \in RCN(B, F)$.

The definition of $B' \to x \in HCLOSEALL(RCN(B, F), F)$ is met: $x \in RCN(B, F)$, $x \notin B'$ and $F \models B' \to x$. Lemma 25 proves that $HCLOSE(RCN(B, F), F)$ contains the minimal clauses of $HCLOSEALL(RCN(B, F), F)$. If $B' \to x$ is minimal it is in $HCLOSE(RCN(B, F), F)$; otherwise, one of its subclauses is in $HCLOSE(RCN(B, F), F)$, and a subclause always entails its superclauses. $\qquad\square$

A particular case where the premise of this lemma holds is when the formula is $UCL(B, F)$.

Lemma 72 *It holds* $UCL(B, F) \equiv HCLOSE(RCN(B, F), UCL(B, F))$.

Proof. By definition, $UCL(B, F)$ contains only clauses $B' \to x$ such that $F \models B \to B'$. This entailment is the same as $F \models B \to b$ for every $b \in B'$. Lemma 33 tells that $F \models B \to b$ is the same as $UCL(B, F) \models B \to b$. This implies $UCL(B, F) \models B \to b$ for every $b \in B'$, which is the same as $UCL(B, F) \models B \to B'$.

This condition $UCL(B, F) \models B \to B'$ holds for every $B' \to x \in UCL(B, F)$. Lemma 71 proves $UCL(B, F) \equiv HCLOSE(RCN(B, F), UCL(B, F))$. $\qquad\square$

Another step is the equality of the real consequences of F and $G \cup IT$.

Lemma 73 *If* $UCL(B, G \cup IT) \equiv UCL(B, F)$ *then* $RCN(B, F) = RCN(B, G \cup IT)$.

Proof. Since $RCN()$ is defined semantically, it is unaffected by replacing a formula with an equivalent one: $UCL(B, G \cup IT) \equiv UCL(B, F)$ implies $RCN(B, UCL(B, G \cup IT)) = RCN(B, UCL(B, F))$. The claim follows from Lemma 67, which states $RCN(B, F) = RCN(B, UCL(B, F))$ and $RCN(B, G \cup IT) = RCN(B, UCL(B, G \cup IT))$. $\qquad\square$

The latest version of the check $SCL(B, F) \cup IT \models BCL(B, F)$ was $UCL(B, G \cup IT) \equiv UCL(B, F)$. This equivalence can be turned into an equality.

Lemma 74 *The condition* $UCL(B, G \cup IT) \equiv UCL(B, F)$ *is the same as* $HCLOSE(RCN(B, F), UCL(B, F)) = HCLOSE(RCN(B, G \cup IT), UCL(B, G \cup IT))$.

Proof. Lemma 72 proves that $UCL(B, F)$ is equivalent to $HCLOSE(RCN(B, F), UCL(B, F))$ and $UCL(B, G \cup IT)$ to $HCLOSE(RCN(B, G \cup IT), UCL(B, G \cup IT))$. If $HCLOSE(RCN(B, F), UCL(B, F))$ is equal to $HCLOSE(RCN(B, G \cup IT), UCL(B, G \cup IT))$ then $UCL(B, F)$ and $UCL(B, G \cup IT)$ are equivalent to the same formula. Therefore, they are equivalent.

The other direction of the claim is now proved. Since $HCLOSE()$ is defined semantically, it has the same value when applied to two identical sets and two equivalent formulae. Since $RCN(B, F) = RCN(B, G \cup IT)$ as proved by Lemma 73 and $UCL(B, F) \equiv UCL(B, G \cup IT)$ by assumption, the claim $HCLOSE(RCN(B, F), UCL(B, F)) = HCLOSE(RCN(B, G \cup IT), UCL(B, G \cup IT))$ follows. $\qquad\square$

The chain of Lemma 65, Lemma 70 and Lemma 74 prove that the dominant operation of the algorithm $SCL(B, F) \cup IT \models BCL(B, F)$ is equivalent to $HCLOSE(RCN(B, F), UCL(B, F)) = HCLOSE(RCN(B, G \cup IT), UCL(B, G \cup IT))$. The first set can be determined from F and B alone, outside the loop over the possible sets IT.

The algorithm that calculates $UCL()$ also produces $RCN()$. As a result, $RCN(B, G \cup IT)$ is determined at no additional cost when $UCL(B, G \cup IT)$ is calculated.

A slight simplification for $HCLOSE(RCN(B, F), UCL(B, F))$ is that $HCLOSE(HEADS(B, F), UCL(B, F))$ is already known as a part of the mechanism for determining the bodies of the clauses of IT. Since $HEADS(B, F) \subseteq RCN(B, F)$ by definition, the required set $HCLOSE(RCN(B, F), UCL(B, F))$ can be calculated as $HCLOSE(HEADS(B, F), UCL(B, F)) \cup HCLOSE(RCN(B, F) \backslash HEADS(B, F), UCL(B, F))$.

4.13 Necessity is the mother of invention

The dominating operation $SCL(B, F) \cup IT \models BCL(B, F)$ is simplified as much as possible to improve the algorithm efficiency. Yet, even in the form $HCLOSE(RCN(B, F), UCL(B, F)) = HCLOSE(RCN(B, G \cup IT), UCL(B, G \cup IT))$ it is expensive as it requires computing a resolution closure, albeit bounded by the heads.

The good news is that sometimes it is not necessary. For example, if $RCN(B, F) \neq RCN(B, G \cup IT)$ then $G \cup IT$ fails at replicating F since they differ on their consequences

of B; the candidate IT can be discarded without further analysis. The additional cost of this check is only that of comparing two sets, since both $RCN(B, F)$ and $RCN(B, G \cup IT)$ are necessary anyway.

Other necessary conditions exist.

They are all proved as: $SCL(B, F) \cup IT \models BCL(B, F)$ implies something. By contraposition, not something implies not $SCL(B, F) \cup IT \models BCL(B, F)$. If that something is easy to determine, it saves from the expensive entailment test when false: IT is discarded straight away.

Any of the conditions that are equivalent to $SCL(B, F) \cup IT \models BCL(B, F)$ can be used in its place: $UCL(B, G \cup IT) \models BCL(B, F)$, $UCL(B, G \cup IT) \equiv UCL(B, F)$, and $HCLOSE(RCN(B, F), UCL(B, F)) = HCLOSE(RCN(B, G \cup IT), UCL(B, G \cup IT))$.

The trick can be repeated as many times as useful: if $SCL(B, F) \cup IT \models BCL(B, F)$ implies condition one, condition two and condition three, they are all checked; failure in any stops the iteration over IT since it fails $SCL(B, F) \cup IT \models BCL(B, F)$.

4.13.1 The first necessary condition

The first necessary condition to $SCL(B, F) \cup IT \models BCL(B, F)$ is that every variable in the bodies of $HCLOSE(RCN(B, F), UCL(B, F))$ occurs in the body of a clause of $G \cup IT$.

A formal proof is an overshooting. The condition $SCL(B, F) \cup IT \models BCL(B, F)$ is the same as the equality of $HCLOSE(RCN(B, F), UCL(B, F))$ and $HCLOSE(RCN(B, G \cup IT), UCL(B, G \cup IT))$. Equality implies that every variable in a body of the former is in a body of the latter. Since the latter can be obtained by resolving clauses of $UCL(B, G \cup IT)$ and resolution does not create literals, that variable is in a body of $UCL(B, G \cup IT)$. Since $UCL(B, G \cup IT)$ is defined as a subset of $G \cup IT$, that variable is also in a body of $G \cup IT$.

In the other way around, if $G \cup IT$ contains no body with a variable that is in a body of $HCLOSE(RC(B, F), UCL(B, F))$ then $HCLOSE(RCN(B, F), UCL(B, F)) = HCLOSE(RCN(B, G \cup IT), UCL(B, G \cup IT))$ is false.

This necessary condition can be seen as the version on bodies of the condition over the heads. But the condition on the heads is strict: the heads of $ITERATION(B, F)$ are exactly $HEADS(B, F)$. Instead, a specific body of $HCLOSE(HEADS(B, F), UCL(B, F))$ may occur in $ITERATION(B, F)$ or not. The example in `nobody.py` shows such a case: $F = \{a \rightarrow b, b \rightarrow a, c \rightarrow d, d \rightarrow c, ac \rightarrow e\}$ and $B = \{a, b\}$; it makes a equivalent to b and c to d; while $ac \rightarrow e$ is in $HCLOSE(HEADS(B, F), UCL(B, F))$, the iteration function $ITERATION(B, F) = \{bd \rightarrow e\}$ is valid in spite of not containing a body ac. The example `twobodies.py` instead shows a case when the same body occurs multiple times: $F = \{a \rightarrow b, a \rightarrow c\}$ and $B = \{a\}$. The only valid iteration function has $ITERATION(B, F)$ equal to F itself, which contains the body a twice.

The `reconstruct.py` program exploits this condition by checking whether the union of the bodies of $G \cup IT$ contains the union of the bodies of $HCLOSE(RCN(B, F), UCL(B, F))$, failing IT if it does not.

This condition involves IT: the bodies of $G \cup IT$ contain all variables in the bodies of $HCLOSE(RCN(B, F), UCL(B, F))$. Therefore, it can only be checked inside the loop over the possible sets IT. However, the bodies of IT are by construction bodies of $HCLOSE(HEADS(B, F), UCL(B, F))$. This provides a further necessary condition: the

bodies of G and $HCLOSE(HEADS(B,F), UCL(B,F))$ contain all variables in the bodies of $HCLOSE(RCN(B,F), UCL(B,F))$. This other necessary condition involves IT no longer. Therefore, it can be checked before entering the loop over the possible sets IT.

One outside the loop, one inside the loop. Both necessary conditions are based on the bodies of $G \cup IT$ containing those of $HCLOSE(RCN(B,F), UCL(B,F))$; they differ on employing the actual bodies of IT (only given inside the loop) or their upper bound $HCLOSE(HEADS(B,F), UCL(B,F))$ (known outside the loop).

Given the length of these formulae, the following shortcuts are employed: $H = HEADS(B,F)$, $R = RCN(B,F)$ and $U = UCL(B,F)$. If $BD(F)$ is the set of variables in the bodies of a formula, the conditions are:

$$BD(HCLOSE(R,U)) \subseteq BD(G) \cup BD(IT)$$
$$BD(HCLOSE(R,U)) \subseteq BD(G) \cup BD(HCLOSE(H,U))$$

They share some components. Since $H = HEADS(B,F)$ is a subset of $R = RCN(B,F)$, the latter is the union of the former and their set difference: $R = H \cup (R \backslash H)$.

This reformulation propagates to the head closure of these sets: by definition, $HCLOSE(A \cup B, F)$ comprises the clauses $B' \rightarrow x$ that obey certain conditions including $x \in A \cup B$ and not involving $A \cup B$ in any other way. As a result, $HCLOSE(A \cup B, F) = HCLOSE(A, F) \cup HCLOSE(B, F)$.

In the present case, $HCLOSE(R,U)$ is equal to $HCLOSE(H,U) \cup HCLOSE(R \backslash H, U)$. If two sets of clauses are the same, the variables in their bodies are the same: $BD(HCLOSE(R,U) = BD(HCLOSE(H,U)) \cup BD(HCLOSE(R \backslash H, U))$.

This equality allows reformulating the two necessary conditions as follows.

$$BD(HCLOSE(H,U)) \cup BD(HCLOSE(R \backslash H, U)) \subseteq BD(G) \cup BD(IT)$$
$$BD(HCLOSE(H,U)) \cup BD(HCLOSE(R \backslash H, U)) \subseteq BD(G) \cup BD(HCLOSE(H,U))$$

In the second containment, $BD(HCLOSE(H,U))$ is in both sides. Since a set is always contained in itself, it can be removed from the left-hand side.

$$BD(HCLOSE(H,U)) \cup BD(HCLOSE(R \backslash H, U)) \subseteq BD(G) \cup BD(IT)$$
$$BD(HCLOSE(R \backslash H, U)) \subseteq BD(G) \cup BD(HCLOSE(H,U))$$

The `reconstruct.py` program checks these two conditions employing the following two sets.

$$IB = BD(HCLOSE(H,U)) \backslash BD(G)$$
$$HL = BD(HCLOSE(R \backslash H, U)) \backslash BD(G) \backslash IB$$

The first set is called `inbodies` in the program; it comprises the variables that are involved in implying a variable in $HEADS(B,F)$ and are not in a body of G. The second is called

`headlessbodies`; it comprises the variables that are involved in implying a variable that is a head in G, but cannot do that because they are neither in G nor in IB.

The second set HL is used to determine the second necessary condition, the one that is independent of IT. Making its definition explicit in the definition tells why.

$$
\begin{aligned}
HL &= BD(HCLOSE(R\backslash H, U))\backslash BD(G)\backslash IB \\
&= BD(HCLOSE(R\backslash H, U))\backslash BD(G)\backslash(BD(HCLOSE(H,U))\backslash BD(G)) \\
&= BD(HCLOSE(R\backslash H, U))\backslash BD(G)\backslash BD(HCLOSE(H,U))
\end{aligned}
$$

The last step is a consequence of the equality between $A\backslash B\backslash C$ and $A\backslash B\backslash(C\backslash B)$. Since $A\backslash B$ does contain any element of B, subtracting C or subtracting C without the elements of B is the same.

This reformulation shows that the second necessary condition is the same as $HL \subseteq \emptyset$. The only subset of an empty set is itself: $HL = \emptyset$. Since HL does not depend on IT, its emptiness can be checked before entering the loop over the possible sets IT.

The first condition is also expressible in terms of IB and HL. The first step is to reformulate the union by placing the intersection in one of the two sets only: $A \cup B$ is the same as $((A\backslash B) \cup (A \cap B)) \cup ((B\backslash A) \cup (A \cap B))$; the duplication of $A \cap B$ is not necessary, making this union the same as $(A\backslash B) \cup (A \cap B) \cup (B\backslash A)$. This is the same as $A \cup (B\backslash A)$. The first necessary conditions is reformulated using this property.

$$
\begin{gathered}
BD(HCLOSE(H,U)) \cup (BD(HCLOSE(R\backslash H, U))\backslash BD(HCLOSE(H,U))) \\
\subseteq BD(G) \cup BD(IT)
\end{gathered}
$$

The union of two sets is contained in the union of $BD(G)$ and $BD(IT)$. Stated differently, the variables in the two sets that are not in $BD(G)$ are in $BD(IT)$.

$$
\begin{gathered}
(BD(HCLOSE(H,U))\backslash BD(G))\cup \\
(BD(HCLOSE(R\backslash H, U))\backslash BD(HCLOSE(H,U))\backslash BD(G)) \\
\subseteq BD(IT)
\end{gathered}
$$

The two sets in the union are exactly IB and HL. The first necessary condition becomes $IB \cup HL \subseteq BD(IT)$. Since it involves IT, it can only be checked inside the loop. But the loop is not entered at all if HL is not empty. Therefore, it could be reformulated as $IB \subseteq BD(IT)$. The program checks the condition including HL to allow disabling the other necessary condition $HL = \emptyset$ for testing.

Both necessary conditions involve $HEADS(B, F)$, but the program never calculates it. Rather, it exploits Lemma 53 by replacing $HEADS(B, F)$ with $RCN(B, F)\backslash\{x \mid \exists B'. B' \to x \in G\}$. The lemma proves that either they coincide or the autoreconstruction algorithm fails and F is not single-head equivalent. The replacing set is not always the same as the replaced set, but can take its place when checking the two necessary conditions. This is proved by looking at the two cases separately:

- if $HEADS(B, F)$ and $RCN(B, F)\backslash\{x \mid \exists B' . B' \rightarrow x \in G\}$ coincide the two conditions are unchanged by replacing the former with the latter;

- if $HEADS(B, F)$ and $RCN(B, F)\backslash\{x \mid \exists B' . B' \rightarrow x \in G\}$ differ the two conditions may change when replacing the former with the latter; this is not a problem because these conditions are only necessary; if they are made true, the program proceeds as if it did not check them, and fails as proved by Lemma 53; if they are made false they make the program fail immediately. In both cases the program fails sooner or later. This is correct behavior because F is not single-head equivalent as proved by Lemma 53.

4.13.2 The second necessary condition

The aim of the inner loop of the algorithm is to find a set IT that imitates a part of F. A necessary condition is to optimistically assume that everything that could possibly be entailed by IT is entailed: the set of its heads H. Since IT is used with G, these heads are also plugged into G for further entailments, producing $RCN(B \cup H, G)$. The union $H \cup RCN(B \cup H, G)$ bounds the variables that B can imply in $G \cup IT$. If it does not include all of $RCN(B, F)$ in spite of the optimistic assumption, $G \cup IT$ does not imitate F.

This condition only involves the heads of IT, not all of it. By construction, these heads are exactly $HEADS(B, F)$. They are the same for all sets IT. As a result, the condition can be checked before entering the loop over the possible candidate sets IT.

The complication is that $RCN()$ is not the set of variables that can be entailed. That is $BCN()$. Rather, it is the set of real consequences, the variables that are entailed not just because they are in B. This is why a formal proof is required.

Lemma 75 *If H is the set of heads of the formula E, then $RCN(B, G \cup E) \subseteq H \cup RCN(B \cup H, G)$ holds for every formula G that contains no tautology.*

Proof. Let x be a variable of $RCN(B, G \cup E)$. By definition, $G \cup E$ entails $(BCN(B, G \cup E)\backslash\{x\}) \rightarrow x$. Let $B' = BCN(B, G \cup E)\backslash\{x\}$ be the body of this clause. The entailment turns into $G \cup E \models B' \rightarrow x$, with $x \notin B'$.

By Lemma 1, it holds $G \cup E \models B' \rightarrow B''$ for some clause $B'' \rightarrow x \in G \cup E$. If this clause is in E its head x is in H by the premise of the lemma and the claim is proved.

Otherwise, $B'' \rightarrow x$ is in G. In this case, x is proved to belong to $RCN(B \cup H, G)$.

Since B' is $BCN(B, G \cup E)\backslash\{x\}$, it is a subset of $BCN(B, G \cup E)$. Therefore, all its variables are entailed by $G \cup E \cup B$. Equivalently, $G \cup E \models B \rightarrow B'$. With $G \cup E \models B' \rightarrow B''$, it implies $G \cup E \models B \rightarrow B''$.

The head of every clause of E is in H since H is the set of heads of E. As a result, $H \models E$. Together with $G \cup E \models B \rightarrow B''$, it implies $G \cup H \models B \rightarrow B''$. This entailment can be rewritten as $G \models (B \cup H) \rightarrow B''$: every variable of B'' is implied by $B \cup H$ according to G. Therefore, $B'' \subseteq BCN(B \cup H, G)$.

Since G contains no tautology by assumption, and $B'' \rightarrow x$ is one of its clauses, it is not a tautology: $x \notin B''$. The containment $B'' \subseteq BCN(B \cup H, G)$ strengthens to $B'' \subseteq BCN(B \cup H, G)\backslash\{x\}$. Since $B'' \rightarrow x$ is in G, it is also entailed by it: $G \models B'' \rightarrow x$. By monotonicity, the body B'' of this clause can be replaced by any of its supersets like $BCN(B \cup H, G)\backslash\{x\}$, leading to $G \models (BCN(B \cup H, G)\backslash\{x\}) \rightarrow x$, which defines $x \in RCN(B \cup H, G)$. □

A condition is necessary if it follows from $SCL(B,F) \cup IT \models BCL(B,F)$ or an equivalent formulation of it, like $UCL(B,F) \equiv UCL(B, G \cup IT)$.

Lemma 76 *If $UCL(B,F) \equiv UCL(B, G \cup IT)$ then $RCN(B,F) \subseteq HEADS(B,F) \cup RCN(B \cup HEADS(B,F), F)$.*

Proof. Lemma 73 proves that $UCL(B,F) \equiv UCL(B, G \cup IT)$ implies $RCN(B,F) = RCN(B, G \cup IT)$. Lemma 75 tells that the second formula of the equality $RCN(B, G \cup IT)$ is a subset of $H \cup RCN(B \cup H, G)$ where H is the set of heads of IT. The heads of IT are $HEADS(B,F)$. A consequence is $RCN(B,F) \subseteq HEADS(B,F) \cup RCN(B \cup HEADS(B,F), G)$. $\square$

When using $H = RCN(B,F) \backslash \{x \mid \exists B'.B' \to x \in G\}$ as the heads of IT, Lemma 53 applies: if F is single-head equivalent, then H coincides with $HEADS(B,F)$ by Lemma 53. As a result, $RCN(B,F) \subseteq H \cup RCN(B \cup HEADS(B,F), F)$ is another consequence of F being single-head equivalent. If it does not hold, F is not single-head equivalent and the algorithm fails as required.

The set $H \cup RCN(B \cup HEADS(B,F), G)$ is called *maxit* in the `reconstruct.py` program because it is an upper bound to the set of consequences achievable by adding whichever set of clauses IT with heads $HEADS(B,F)$ to G. Yet, it does not involve IT itself. Therefore, it can be checked once for all before entering the loop rather than once for each iteration.

4.13.3 The third necessary condition

Since $G \cup IT$ aims at imitating F on the sets B' that are equivalent to B, it must have the same consequences. A specific case is already proved: $RCN(B, G \cup IT) = RCN(B,F)$ as proved by Lemma 73. This condition extends from B to every equivalent set: $RCN(B', G \cup IT) = RCN(B,F)$ for every $B' \equiv_F B$. This is mostly proved by the following lemma.

Lemma 77 *If $B \equiv_F B'$ then $RCN(B,F) = RCN(B',F)$.*

Proof. The definition of $B \equiv_F B'$ is $F \models B \equiv B'$. This is equivalent to $F \cup B \equiv F \cup B'$. The definition $BCN(B,F) = \{x \mid F \models B \to x\}$ can be rewritten as $BCN(B,F) = \{x \mid F \cup B \models x\}$. Since $F \cup B$ is equivalent to $F \cup B'$, this set is equal to $\{x \mid F \cup B' \models x\}$, or $BCN(B',F)$. This proves $BCN(B,F) = BCN(B',F)$.

As a result, $RCN(B,F) = \{x \mid F \models (BCN(B,F) \backslash \{x\}) \to x\}$ can be rewritten as $\{x \mid F \models (BCN(B',F) \backslash \{x\}) \to x\}$, the definition of $RCN(B',F)$. $\square$

The goal is a necessary condition to the dominant operation of the loop, the check $SCL(B,F) \cup IT \models BCL(B,F)$. Lemma 65 and Lemma 70 prove it equivalent to $UCL(B, G \cup IT) \equiv UCL(B,F)$, which Lemma 73 proves to imply $RCN(B,F) = RCN(B, G \cup IT)$. Transitivity with $RCN(B', G \cup IT) = RCN(B, G \cup IT)$ would prove the claim $RCN(B', G \cup IT) = RCN(B,F)$. Unfortunately, the premise $RCN(B', G \cup IT) = RCN(B, G \cup IT)$ is only proved by Lemma 77 if $B \equiv_{G \cup IT} B'$ holds, but the assumption is $B \equiv_F B'$. The following lemma provides this missing bit.

Lemma 78 *If $UCL(B, G \cup IT) \equiv UCL(B,F)$ then $B' \equiv_F B$ is equivalent to $B' \equiv_{G \cup IT} B$.*

Proof. If $G \cup IT$ entails $B' \equiv B$ then F entails it as well because Lemma 50 proves $F \models G \cup IT$.

The converse is now proved: if $F \models B' \equiv B$ then $G \cup IT \models B' \equiv B$.

Lemma 36 applies to $F \models B \to B$ and $F \models B \to b$ for every $b \in B'$, proving $UCL(B, F) \models B \to b$ for every $b \in B'$, which implies $UCL(B, F) \models B \to B'$. It also applies to $F \models B \to B'$ and $F \models B' \to b$ for every $b \in B$, proving $UCL(B, F) \models B' \to b$, which implies $UCL(B, F) \models B' \to B$. The conclusion is $UCL(B, F) \models B \equiv B'$.

Since $UCL(B, F)$ is equivalent to $UCL(B, G \cup IT)$ by assumption, also $UCL(B, G \cup IT) \models B' \equiv B$ holds. Since $UCL(B, G \cup IT)$ is by definition a subset of $G \cup IT$, the claim $G \cup IT \models B' \equiv B$ follows. $\square$

The third necessary condition can now be proved.

Lemma 79 *If $UCL(B, G \cup IT) \equiv UCL(B, F)$, then $RCN(B', G \cup IT) = RCN(B, F)$ holds for every B' such that $B' \equiv_F B$.*

Proof. Given the premise $UCL(B, G \cup IT) \equiv UCL(B, F)$, Lemma 73 proves $RCN(B, F) = RCN(B, G \cup IT)$. Lemma 78 proves that $B' \equiv_F B$ implies $B \equiv_{G \cup IT} B'$. Lemma 77 applies: $RCN(B, G \cup IT) = RCN(B', G \cup IT)$. The claim $RCN(B, F) = RCN(B', G \cup IT)$ follows by transitivity. $\square$

Lemma 65 and Lemma 70 prove that $SCL(B, F) \cup IT \models BCL(B, F)$ is equivalent to $UCL(B, G \cup IT) \equiv UCL(B, F)$, which implies $RCN(B', G \cup IT) = RCN(B, F)$ for every B' such that $B' \equiv_F B$. This is a necessary condition. It is used like the others: in reverse. If $RCN(B, F) \neq RCN(B', G \cup IT)$ for some B' that is equivalent to B, any further processing of IT is unnecessary since $SCL(B, F) \cup IT \models BCL(B, F)$ is bound to fail.

From the computational point of view, $RCN(B, F)$ is already known from the very beginning. The second set $RCN(B', G \cup IT)$ can be easily determined given a body B' of a clause in $BCL(B, F)$. The problem is the possibly large number of such sets B'. Restricting to $HCLOSE(HEADS(B, F), UCL(B, F))$ or even to $MINBODIES(HCLOSE(HEADS(B, F), UCL(B, F)), UCL(B, F) \cap SCL(B, F))$ still leaves this condition necessary: weaker but easier to check. The `singlehead.py` program uses this restriction in the `noteq` loop.

4.14 Too many optimizations, too little time

Some further directions for improvement to the algorithm have not been investigated enough to prove their correctness and are left unimplemented.

The first undeveloped optimization is turning the third necessary condition into a sufficient condition. It looks like checking $RCN(B', G \cup IT) = RCN(B, F)$ for all $B' \in HCLOSE(HEADS(B, F), UCL(B, F) \cap SCL(B, F))$ is not just necessary to $SCL(B, F) \cup IT \models BCL(B, F)$ but also sufficient. If so, the following check $HCLOSE(RCN(B, F), UCL(B, F)) = HCLOSE(RCN(B, G \cup IT), UCL(B, G \cup IT))$ is redundant.

The second undeveloped optimization is a simplification of the check $HCLOSE(RCN(B, F), UCL(B, F)) = HCLOSE(RCN(B, G \cup IT), UCL(B, G \cup$

IT)).   A subset of $HCLOSE(RCN(B,F),UCL(B,F))$ is already known:
$HCLOSE(HEADS(B,F),UCL(B,F))$ is computed to determine the possible sets
IT to test.   Restricting the check to $HCLOSE(HEADS(B,F),UCL(B,F))$ $=$
$HCLOSE(HEADS(B,F),UCL(B,G \cup IT))$ saves time both because the first formula
is already known and because the second is easier to compute due to the smaller first
argument.

This is not correct in general, as various examples in `bnotheads.py` show. One of them
is the following.

$$F \;=\; \{ab \equiv ef, a \equiv c\}$$
$$\;=\; \{ab \to e, ab \to f, ef \to a, ef \to b, a \to c, c \to a\}$$
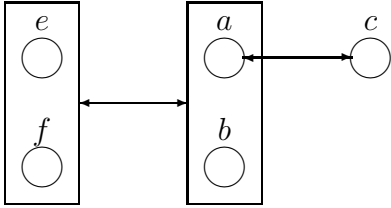
Since $SCL(B,F)$ is $\{a \to c, c \to a\}$ when $B = \{a,b\}$, the variable a is not in
$SFREE(B,F)$, and therefore not in $HEADS(B,F)$. Therefore, $ef \to a$ is never in IT.
Yet, its absence goes undetected because it is not in $HCLOSE(HEADS(B,F),UCL(B,F))$
either.

The third necessary condition catches the anomaly: a is in $RCN(B,F)$ but not in
$RCN(\{e,f\}, G \cup IT)$ since no IT contains a clause of head a. This is however specific
to this case, as the following counterexample still from `bnotheads.py` shows.

$$F \;=\; \{ab \equiv ef, a \equiv c, b \equiv d)$$
$$\;=\; \{ab \to e, ab \to f, ef \to a, ef \to b, a \to c, c \to a, b \to d, d \to b\}$$
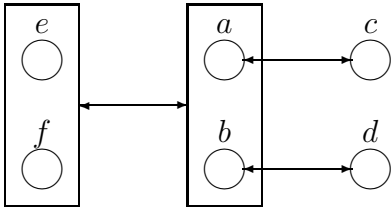
Again, a $\notin$ $HEADS(B,F)$ causes ef $\to$ a not   to   be   in
$HCLOSE(HEADS(B,F),UCL(B,F))$. However, b, c and d are not in $HEADS(B,F)$ ei-
ther, making $ef \to b$, $ef \to c$, $ef \to d$ not being in $HCLOSE(HEADS(B,F),UCL(B,F))$.
As a result, $\{e,f\}$ is not a body of $HCLOSE(HEADS(B,F),UCL(B,F))$. The equality
$RCN(\{e,f\}, G \cup IT) = RCN(B,F)$ is not even checked.

An undetected inequality like this is due to some sets B' not being recognized as equivalent
to B. In turns, this seems to be only possible if $B \cap HEADS(B,F) = \emptyset$, but this is yet to
be proved. If proved, the optimization can be employed if $B \cap HEADS(B,F) \neq \emptyset$, resorting
to the general condition $HCLOSE(RCN(B,F),UCL(B,F)) = HCLOSE(RCN(B,G \cup$
$IT),UCL(B,G \cup IT))$ otherwise.

The third undeveloped optimization is that associating a body to a variable $x \in HEADS(B, F)$ is easy if x is not in any of the possible bodies of IT. This condition is easy to check: $x \notin \bigcup T$.

All clauses in $ITERATION(B, F)$ are in $BCL(B, F)$ by definition: they are nontautological clauses $B' \to x$ entailed by F that meet $B \equiv_F B'$. When building a clause for the head x, all acceptable bodies are equivalent to B according to F, and are therefore all equivalent to each other. Since they are equivalent, they are the same. Any one will do. Even just B.

Why the requirement $x \notin \bigcup T$, then? Because $B \equiv_F B'$ is equivalence according to F, not to $G \cup IT$. Equivalence according to $G \cup IT$ is still to be achieved. The following formula (in `outloop.py`) clarifies this point.

$$F = \{a \to b, b \to c, c \to a, b \to x, c \to x\}$$

The heads for $B = \{a\}$ are $HEADS(B, F) = \{a, b, c, x\}$. The task is to produce a body for each of these variables. The simplified rule lets B be the body for all.

This is correct for x: the clause $a \to x$ is a valid choice for a clause with head x. It is in the equivalent single-head formula F', for example:

$$F' = \{a \to x, a \to b, b \to c, c \to x\}$$

The same choice does not work for b and c: no single-head formula equivalent to F contains both $a \to b$ and $a \to c$. For example, the latter cannot replace $b \to c$ in F' even if their bodies are equivalent because the equivalence $a \equiv_F b$ extends to F' only as consequence of $b \to c$.

This mechanism does not work because it relies on body equivalences that are not achieved yet. Saying that $B \to x$ is as good as $B' \to x$ because B is equivalent to B' anyway is incorrect because $B' \to x$ may be necessary to ensure this equivalence.

This is not the case for x. Visually, x is outside any loop.
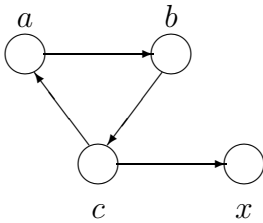
Technically, if x is not in $\bigcup T$ it cannot contribute to the equivalence of B with any other set B'. Equivalence requires going from a set to another and back. Instead, x can only be entailed. It cannot entail. Since $B' \to x$ is unnecessary to $B' \equiv B$, removing it does invalidate this equivalence. Therefore, $B' \to x$ still entails $B' \to x$, constituting a valid replacement.

This optimization would use $H' = H \cap \bigcup T$ in place of H and search for a set IT that meets $HCLOSE(RCN(B, F) \cap \bigcup T, UCL(B, F)) = HCLOSE(RCN(B, G \cup IT) \cap \bigcup T, UCL(B, G \cup IT))$ instead of $HCLOSE(RCN(B, F), UCL(B, F)) = HCLOSE(RCN(B, G \cup IT), UCL(B, G \cup IT))$. On success, a clause $B \to x$ for every $x \in H \backslash H'$ is added to G along with IT.

Most of the other examples in this article and in the test directory of the `singlehead.py` program would not benefit from this optimization. This is probably a distortion due to the nature of the technical analysis of the problem and its algorithms. What makes single-head equivalence difficult from the theoretical and implementation point of view is establishing equivalence of sets B' with B; this is why most examples revolve around this case. Variables that are entailed but not in a loop are simple to deal with. Examples that contain them are not significant.

Not significant to the technical analysis of the problem does not mean not significant to the problem itself.

To the contrary, this optimization may dramatically increase performance. For example, the algorithm is fast at telling that Formula 1 is not single-head equivalent because it only needs to test the possible associations of three heads $H = \{x, a, b\}$ with three bodies $\{b, d\}$, $\{a, c\}$ and $\{x\}$. Adding easy clauses like $x \to y_i$ with $i = 1, \ldots, n$ creates n new heads, exponentially increasing the number of associations to test. This is wasted time, since their bodies can for example be all $\{x\}$.

$$F = \{x \to a, a \to d, x \to b, b \to c, ac \to x, bd \to x\} \tag{1}$$

This formula is in the file `disjointnotsingle.py`, the variant with the two additional clauses in `insignificant.py`.

The fourth unimplemented optimization would allow establishing that a formula is not single-head equivalent without a single entailment test. It is based on a counting argument, a calculation of the number of necessary clauses. The following example illustrates its principle.
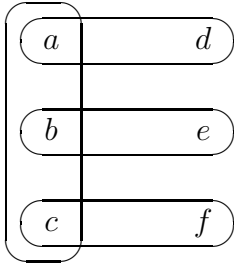
A box stands for a set of variables that is equivalent to the others: $\{a, b, c\} \equiv \{a, d\} \equiv \{b, e\} \equiv \{c, f\}$. The equivalence between $\{a, b, c\}$ and $\{a, d\}$ is for example realized by $abc \to d$, $ad \to b$ and $ad \to c$: each variable that is in a set but not in the other is implied by the other. The file for this formula is `disjointemptynotsingle.py`.

The bodies of the formula are the equivalent sets; therefore, they are all equivalent to each other. The first precondition chosen by the program could for example be $\{a, b, c\}$. Regardless, $SCL(B, F)$ is empty. The `singlehead.py` program disproves the formula single-head equivalent testing 4096 combinations of heads and bodies.

This computation can be skipped altogether by just counting the number of necessary clauses. Since $\{a, d\}$ implies the other equivalent sets and only the equivalent sets imply something, $G \cup IT$ must contain some clauses that make $\{a, d\}$ directly imply at least another equivalent set: $\{a, b, c\}$, $\{b, e\}$ or $\{c, f\}$. Each contains two variables not in $\{a, d\}$, requiring two clauses. For example, $ad \to b$ and $ad \to c$ are required to imply $\{b, c\}$ from $\{a, d\}$. They are irredundant as a alone does not imply anything in F, nor does d.

The same applies to the other three equivalent sets: two clauses each. The total is eight

clauses. They are all different: the clauses for different equivalent sets are different because the equivalent sets all differ from each other; the two clauses for the same equivalent set are different because they have different heads. The eight clauses are all different.

The variables are only six; a single-head formula contains at most six clauses. The formula is not single-head equivalent because no equivalent formula contains less than eight clauses.

This mechanism generalizes to all formulae that contain equivalent sets of variables. For each of them B', the number of variables in the minimal difference $B'' \backslash B' \backslash SFREE(B, F)$ to another equivalent set B'' is counted. The variables not in $SFREE(B, F)$ are discarded, as implicitly done on the example where $SCL(B, F) = \emptyset$. The total is the minimal number of necessary clauses. If it is larger than the number of heads, the formula is not single-head equivalent.

When determining the equivalent sets of variables, only the minimal ones count. For example, the sets that matter in $F = \{a \to b, b \to a, a \to c\}$ are only $\{a\}$ and $\{b\}$, not $\{a, c\}$ and $\{b, c\}$.

This condition is only necessary, as some formulae have less equivalent sets than heads but are not single-head equivalent anyway. An example is the formula $ab \equiv bc \equiv ca \equiv de \equiv ef \equiv fd$. Each set can imply another with a single clause. For example, $\{c, a\}$ implies $\{b, c\}$ thanks to $ca \to b$ only.
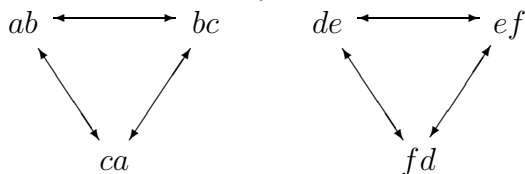
The counting condition is satisfied. Yet, the six clauses it counts as necessary only realize the two loops of equivalences. Other clauses are necessary to link the two, outnumbering the heads.

The final undeveloped variant is not an optimization of the algorithm but a way to make it produce a meaningful result even if the input formula is not single-head equivalent. At each step, the set of heads of the candidate set IT is $RCN(B, F)$ minus the heads of the formula under construction G. If the formula is single-head equivalent, the result is equal to $HEADS(B, F)$. Otherwise, the autoreconstruction algorithm fails as proved by Lemma 53.

This mechanism presumes that a single-head formula equivalent to F is the only goal; if none exists, no output is necessary. This may not be the case. A formula that contains just a pair of clauses with the same head is almost as good as one containing none: forgetting requires only one true nondeterministic choice, which only doubles the running time. Double is still polynomial.

The goal extends from finding a single-head formula to a formula containing few duplicated heads. It is achieved by adding a clause $B' \to x$ for each $x \in RCN(B, F) \backslash RCN(B, G \cup IT)$.

5   Polynomial or NP-complete?

The reconstruction algorithm requires a valid iteration function, which in turn requires testing the possible association of some bodies to some heads. These associations may be exponentially many. The algorithm takes exponential time if so.

Is this specific to the reconstruction algorithm? Or is the problem itself that requires exponential time to be solved?

While no proof of NP-hardness is found so far, at least the problem is not harder than NP. The considered decision problem is: does the formula have an equivalent formula that is single-head? Testing all equivalent formulae for both equivalence and single-headness is not feasible even in nondeterministic polynomial time since formulae may be arbitrarily large. Single-headness caps that size, since it allows at most one clause for each head. A single-head formula may at most contain a clause for each variable.

Theorem 2 *Checking whether a formula is equivalent to a single-head formula is in* NP.

Proof. Equivalence to a single-head formula can be established by a nondeterministic variant of the reconstruction algorithm, where IT is not looped over but is uniquely determined by guessing a body for each variable in $HEADS(B, F)$. Technically, the loop "for all IT..." is replaced by "nondeterministically assign an element of T to each element of H". The resulting algorithm is polynomial. □

Proving membership to NP only takes a short paragraph. Proving NP-hardness is a completely different story. The rest of this section gives clues about it.

A proof of NP-hardness is a reduction from some other NP-hard problem to that of telling whether F is single-head equivalent. Given an instance of the problem that is known NP-hard, it requires building a formula that is single-head equivalent or not depending on that instance. This is generally done by some sort of formula template, a scheme that has some parts left partially unspecified. The specific NP-hard problem instance fills these so that the resulting concrete formula is either single-head or not depending on it.

While no such reduction is known, still something can be said about the formula schemes that work and the ones that do not.

Lemma 22 tells that single-head equivalence requires two things: first, $HEADS(B, F)$ and $HEADS(B', F)$ are disjoint whenever $B \not\equiv_F B'$; second, $ITERATION(B, F)$ is single-head.

The first condition is polynomial. A reduction that only exploits it would reduce a NP-hard problem to a polynomial one. Winning the million dollar price at stake for it [Coo03] would be certainly appreciated, but seems unlikely. Rather, it suggests that a NP-hard proof is not based on the disjointness of heads.

The reduction, if any, is based on the other condition: the existence of a valid iteration function such that $ITERATION(B, F)$ is single-head. This is exactly where the autoreconstruction algorithm spends more time: testing all associations of the bodies in T to the heads in H. A reduction from the propositional satisfiability of a formula G to the single-head equivalence of F would probably hinge around a single, fixed set B:

- $HEADS(B', F)$ and $HEADS(B'', F)$ are disjoint for all $B' \not\equiv_F B''$ regardless of G;

- $SCL(B, F)$ is independent on G, or maybe only depends on the variables of G;

- B is a fixed set of variables, maybe a singleton; maybe it is a set that only depends on the variables of G;

- G is satisfiable if and only if a single-head subset of $HCLOSE(HEADS(B, F), UCL(B, F))$ entails $BCL(B, F)$ with $SCL(B, F)$.

The core of the reduction is the last point: the satisfiability of G corresponds to the existence of a single-head subset IT of $HCLOSE(HEADS(B,F),UCL(B,F))$ satisfying $SCL(B,F) \cup IT \models BCL(B,F)$, where B is fixed. The clauses of $F \cap SCL(B,F)$ may support this construction; or maybe they are just not needed, and F is entirely made of clauses of $BCL(B,F)$. Whether $SCL(B,F)$ is necessary or not might further restrict the range of possible reductions. The clauses of $SCL(B,F)$ and $BCL(B,F)$ differ in an important way when it comes to ensuring $SCL(B,F) \cup IT \models BCL(B,F)$, as explained in the next section.

5.1   What do clauses do?

While searching for a proof of NP-hardness for the single-head equivalence problem, an observation emerged: clauses in $SCL(B,F)$ and in $ITERATION(B,F)$ differ in a fundamental point: which and how many clauses of $BCL(B,F)$ they collaborate to imply.

All clauses of $BCL(B,F)$ have the form $B' \to x$ with $B' \equiv_F B$. Apart the case where B is only equivalent to itself, this means that B' entails B and all other sets equivalent to B. This is not possible if $SCL(B,F) \cup ITERATION(B,F)$ does not contain at least a clause that can be applied from B': a clause whose body is contained in B'. This clause makes B' entail another equivalent set, which in turn entail another in the same way. This clause may belong to $SCL(B,F)$ or to $BCL(B,F)$.

$B'' \to x \in SCL(B,F)$ may have a body B'' that is a subset of multiple sets B' that are equivalent to B; therefore, $B'' \to x$ may be a starting point of implication from many differing B';

$B'' \to x \in ITERATION(B,F)$ implies that B'' is exactly one of these sets that are equivalent to B; therefore, $B'' \to x$ may only be the starting point of implications from $B' = B''$ itself.

The hard task for $ITERATION(B,F)$ is to realize all equivalences $B' \equiv B$ entailed by F. As explained in the previous sections, the clauses not involved in implying such equivalences are easy to find. Equivalences are mutual entailments and are therefore broken down into entailments like $B' \to B''$ where both B' and B'' are equivalent to B according to F. The difference between a clause of $SCL(B,F)$ and one of $ITERATION(B,F)$ is that the first can be used to realize many implications $B' \to B''$, the second only the ones starting from its body.

The following example clarifies the difference: $F = \{a \equiv b, ac \to d, ad \to e, ae \to c\}$ when $B = \{a,c\}$.
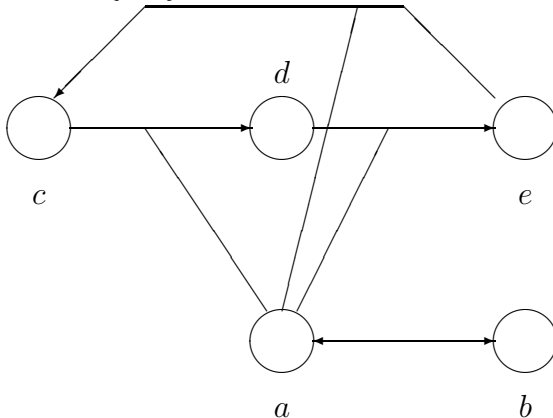
The clauses of strictly entailed bodies are $SCL(B, F) = \{a \equiv b\}$. The bodies equivalent to B are $\{a, c\}$, $\{a, d\}$, $\{a, e\}$, $\{b, c\}$, $\{b, d\}$, and $\{b, e\}$. The clause $ad \to e \in BCL(B, F)$ can only be used when $B' = \{a, d\}$ is true. It only allows the direct implication $ad \to ae$. On the other hand, the clause $a \to b \in SCL(B, F)$ entails $ac \to bc$, $ad \to bd$ and $ae \to be$.
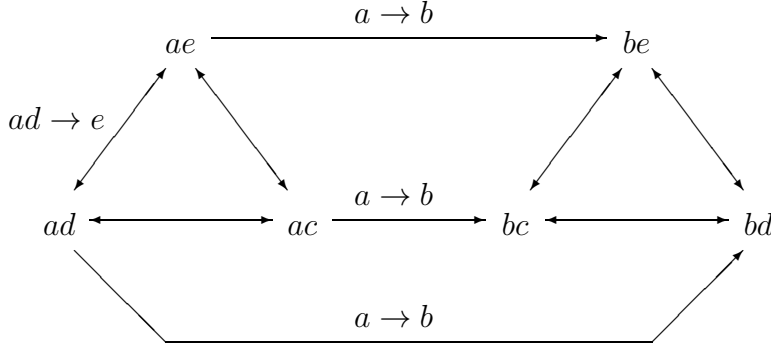

5.2 Counterexamples

Lemma 22 implies that $HEADS(B, F)$ and $HEADS(B', F)$ are always disjoint if F does not entail $B \equiv B'$ and is single-head equivalent. This is however only a necessary condition, as sufficiency also requires the existence of a valid iteration function such that $ITERATION(B, F)$ is single-head.

Since $HEADS(B, F) \cap HEADS(B', F) = \emptyset$ is easy to check for fixed B and B', a proof of NP-hardness cannot be based on it. It should instead hinge around the other condition. For example, it produces formulae that always have disjoint $HEADS(B, F)$, but have a single-head $ITERATION(B, F)$ if the given formula is satisfiable.

Such a reduction may result from generalizing one of the example formulae that have disjoint $HEADS(B, F)$ but $ITERATION(B, F)$ is not single-head.

1. A set entails two equivalent sets; achieving equivalence requires them to entail another equivalent set each, but this requires two clauses with the same head.

   Technically, $SCL(B, F)$ implies $A \to B$, $A \to C$ but no other entailment among the minimal F-equivalent sets A, B, C and D. Some other entailments are required to make the four sets equivalent. For example, to make B equivalent to the other sets, it has to imply at least one of them. Since the clauses of $SCL(B, F)$ do not suffice, some clauses from $BCL(B, F)$ are needed. Such clauses have body B because they are directly applicable from B (their body is a subset of B) and are not in $SCL(B, F)$ (their body is not a proper subset of B). At least one clause $B \to x$ is necessary. For the same reason, $C \to y$ and $D \to z$ are also necessary, but cannot be entailed from single-head clauses.
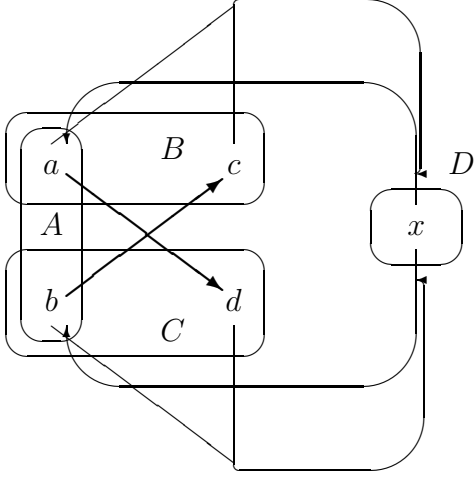
This example is in the `disjointnotsingle.py` file. As required, $SCL(B,F)$ implies $A \to B$ and $A \to C$ and nothing else. Realizing all equivalences among A, B, C and D requires completing the loops: while A already implies other equivalent sets, B, C and D do not. This can only be done with clauses of $BCL(B,F)$. Since tautologies are useless and heads in $SCL(B,F)$ are forbidden, the only possible clauses for B are $B \to b$ and $B \to x$; this respectively leaves $C \to x$ or $C \to a$ for C. Either a or b is the head of one such clause. This leaves only the other one to D, but neither $D \to a$ alone nor $D \to b$ alone is enough to make D entail the other equivalent sets. The problem can be attributed to the presence of D, since all equivalences are otherwise completed by $B \to b$ and $C \to a$.

The problem may show up even if $SCL(B,F)$ is empty. The counterexample is in `disjointemptynotsingle.py`. It makes ad, be, cf and abc equivalent. Each of them should imply another; since $SCL(B,F)$ is empty, this can only be done by clauses having them as bodies. Since each body differs from the others by two variables, the total is eight clauses. The variables are only six, insufficient for eight single-head clauses.
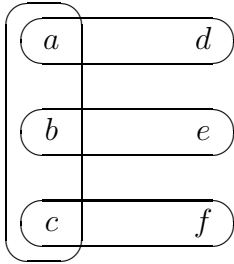
2. The same variable is necessary for two entailments among equivalent sets. For example, F entails both $A \to B$ and $C \to D$, but B and D contain the same variable. An example is $A = \{a,b\}$, $B = \{c,d\}$, $C = \{a,e\}$ and $D = \{c,f\}$ with the formula in the file `disjointemptynotsingle.py`.
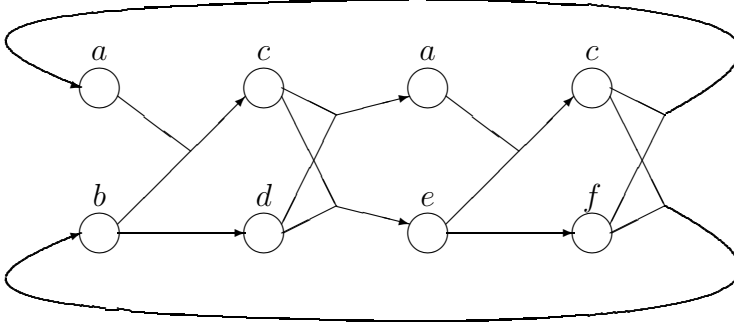
Starting from $\{a, b\}$, the $SCL(B, F)$ part of F already entails d, which is part of the equivalent set $\{c, d\}$. This calls for $ab \rightarrow c$. The same applies to $\{a, e\}$ and $ae \rightarrow c$, which has the same head.

The variables a and c and shown twice to emphasize the structure of the closed chain A, B, C and D. Without the repetition the formula looks like the following figure. The two clauses with head a can be replaced by $c \rightarrow a$.
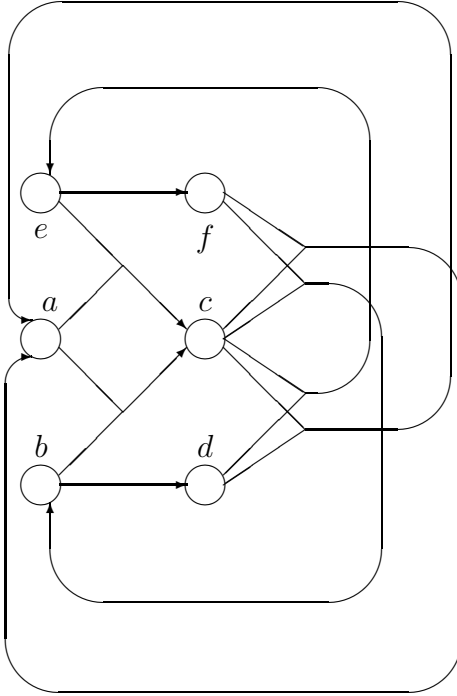
3. The formula requires connecting too many equivalent sets. All examples shown so far show some "local" issues: a path of equivalent sets that cannot be completed to form a loop. Other counterexamples allow each set to entail another, but the resulting graph of equivalent sets of variables is disconnected, like the formula in `disconnected.py`:

$$F = \{abc \equiv def, def \equiv ghi, ghi \equiv jk, jk \equiv mn, mn \equiv abc, a \rightarrow d, e \rightarrow g, i \rightarrow c, j \rightarrow m, q \rightarrow n\}$$

The binary clauses in $SCL(B, F)$ push $ITERATION(B, F)$ toward entailing $abc \rightarrow def$, $def \rightarrow ghi$ and $ghi \rightarrow abc$, closing the loop short of including jm and qn.
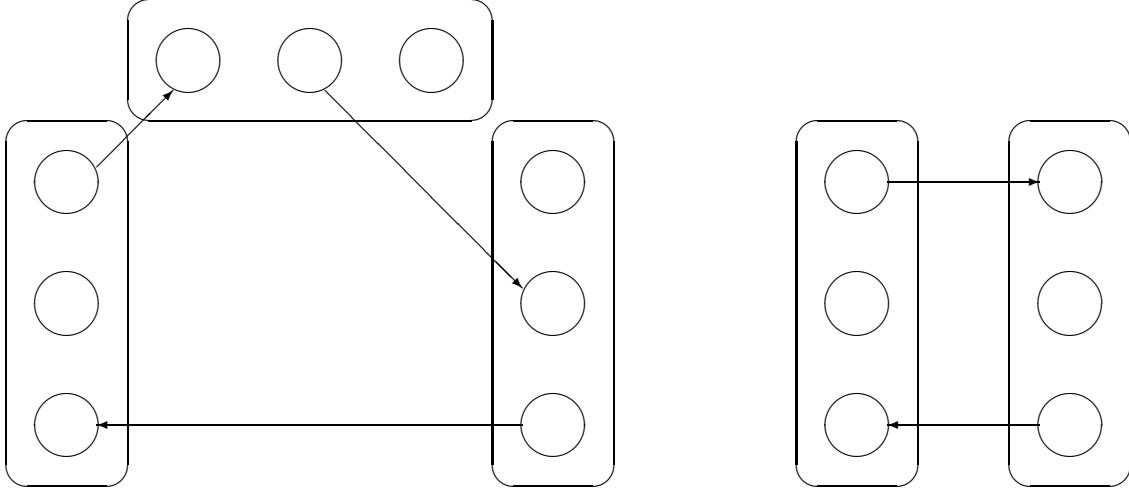
Looking at every single part of the formula in isolation it looks single-head equivalent, since each equivalent set can entail another. The problem is producing a single loop that reaches all of them. This shows that single-head equivalence is not a local property that can be checked by looking only at the individual equivalent sets of variables.

6 Python implementation

The `reconstruct.py` Python program implements the algorithm described in Sections 3 and 4. It is available at `https://github.com/paololiberatore/reconstruct`.

It takes either a sequence of clauses or a file name as commandline options. Formulae like `ab->cd` and `df=gh` are translated into clauses. Examples invocations are:

```
reconstruct.py -f 'ab->cd' 'df=gh'
reconstruct.py -t tests/chains.py
```

After turning `ab->cd` into $ab \to c$ and $ab \to d$ and turning `df=gh` into $df \to g$, $hg \to f$, $df \to h$ and $hg \to d$, it removes the tautologies from them, if any.

The reconstruction algorithm described in Section 3 with the iteration function in Section 4 is implemented in the `reconstruct(f)` function. The loops over the bodies of the formula and over the candidate set of clauses for the iteration function are in this function. This is because the optimized method for the latter uses a number of sets accumulated by the former: the set of clauses under construction, their heads, their bodies and the clauses of the original formula used so far. Only the calculation of $RCN()$, $UCL()$, $HCLOSE()$ and the search for the minimal bodies are split into separate functions: `rcnucl(b, f)`, `hclose(heads, usable)` and `minbodies(f)`. The first is copied verbatim from the implementation of the incomplete algorithm described in a previous article [Lib20c].

The function `rcnucl(b,f)` is first called on each body of the formula, and the results stored. These are used to check for example $A \leq_F B$. This condition is indeed the same as $F \models B \to A$, which is the same as $BCN(A, F) \subseteq BCN(B, F)$, in turn equivalent to as $A \cup RCN(A, F) \subseteq B \cup RCN(B, F)$.

For each precondition B, the candidate values for $ITERATION(B, F)$ are produced by first calculating the required heads $HEADS(B, F)$ and attaching the allowed bodies to them in all possible ways. All property of valid iteration functions hold

72

by construction except $G \cup ITERATION(B, F) \models BCL(B, F)$, which is checked as $HCLOSE(RCN(B, F), UCL(B, F)) = HCLOSE(RCN(B, G \cup IT), UCL(B, G \cup IT))$ using `hclose(heads, usable)` and `rcnucl(b, f)`.

The algorithm is complete, but slower than the incomplete algorithm presented in the previous article [Lib20c]. The number of candidate sets for $ITERATION(B, F)$ may exponentially increase with the number of variables. This happens when a set of variables B is equivalent to many minimal equivalent sets. Such equivalences are due to loops of clauses. They are the reason why the previous algorithm is incomplete, and they make the present one exponential in time.

7 Conclusions

The algorithm for turning a formula in single-head form if possible is correct and complete, but may take exponential time. It complements the previous algorithm, which is polynomial but incomplete [Lib20c], rather than overcoming it. Which of them is the best depends on the goal.

If the goal is to reduce the size of forgetting rather than computing it quickly, completeness wins over efficiency. The formulae that are not turned into single-head form by the previous algorithm are by the present one. Forgetting from them produces polynomially-sized formulae. Running time is sacrificed on the altar of size reduction.

If the goal is to forget quickly, spending exponential time on preprocessing is self-harming. Forgetting may take polynomial time on the single-head formula, but producing it took exponential time. Better spend less time preparing the formula, and rather jump straight to forgetting.

The present algorithm may still be of use in the second case. What it does is to reconstruct the formula by groups of clauses of equivalent bodies. If a group takes too long, the reconstruction can be cut short by just producing the clauses with these bodies in the original formula. This part of the output formula may not be single-head, but the rest will. Since the efficiency of forgetting depends on the number of duplicated heads, this is still an improvement.

The two algorithms can be applied in turn. The old computes a formula $SHMIN(F)$ that is always single-head but may not be equivalent to the original. Only if it is not the second, slower algorithm is run. It may take advantage of $SHMIN(F)$. When searching for the clauses with a certain set of heads, the ones from $SHMIN(F)$ may be tested first.

Given that the only source of exponentiality in time are cycles of clauses, it makes sense to simplify them as much as possible. A way to do it is to merge equivalent sets of variables. As an example, if the formula under construction entails the equivalence of A and B, only one of them is really necessary. For example, every occurrence of B can be replaced by A. This is a simplification because it negates the need to test both $(A \cup \{y\}) \to x$ and $(B \cup \{y\}) \to x$ as possible clauses for the head x. A followup article [Lib20d] further analyzes this trick.

A specific formula that turned out important during the analysis of the algorithm is $F = \{x \to a, a \to d, x \to b, b \to c, ac \to x, bd \to x\}$, in the `disjointnotsingle.py` test file of the `singlehead.py` program.
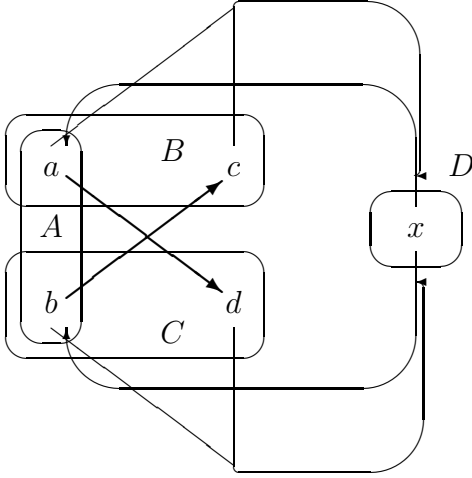
It is not single-head because the four sets A, B, C and D are equivalent, but only three heads are available: a, b and x. The other two variables are already the heads of $a \rightarrow d$ and $b \rightarrow c$.

A formula may also not be single-head equivalent for other reasons, but examples for these cases are much simpler than this. They range from two to three binary clauses. They are quite immediate. The relative size of this one, with six clauses including two ternary ones, witnesses the complexity of this case.

Given that the first algorithm is polynomial but incomplete and the second is complete but not polynomial, a natural question is whether a complete and polynomial algorithm exists. If so, the problem of recognizing whether a formula is single-head equivalent would be tractable. This problem is in the complexity class NP, but whether it is NP-complete or not is an open question.

Another open question is the applicability of the algorithms outside the Horn fragment of propositional logic.

In the general propositional case, resolving all occurrences of a variable forgets the variable. This is the same as viewing a clause like $\neg a \vee b \vee c$ as an implication $a \neg b \rightarrow c$. If no other occurrence of c is positive, it is forgotten by replacing every negative occurrence of it with $a \neg b$. The size increase is only linear in the single-head case. The question is whether the algorithms for turning a formula in single-head form extends to the general propositional case.

The problem complicates when forgetting multiple variables, since the replacement may multiply the number of positive occurrences of other variables, like a in this case. An alternative is to apply the replacing algorithm only to the definite Horn clauses of the formula. In the example, a clause like $\neg c \vee d \vee e$ would be left alone, and c only replaced by definite Horn clauses like $\neg c \vee \neg d \vee e$, equivalent to $dc \rightarrow e$. The remaining occurrences of c are then forgotten by resolving them out. If the formula contains few non-Horn clauses, this procedure may be relatively efficient.

This second mechanism would probably apply to every logic that includes the definite Horn fragment. Information like "this, this and this imply that" is common, so definite Horn clauses are likely to be present in large number. Turning them into single-head formula would allow to efficiently forget them before turning to the others.

References

[AEW12]   G. Antoniou, T. Eiter, and K. Wang. Forgetting for defeasible logic. In *Proceedings of the eighteenth conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-18)*, volume 7180 of *Lecture Notes in Computer Science*, pages 77–91. Springer, 2012.

[BK94]   A.J. Bonner and M. Kifer. An overview of transaction logic. *Theoretical Computer Science*, 133(2):205–265, 1994.

[Boo54]   G. Boole. *Investigation of The Laws of Thought, On Which Are Founded the Mathematical Theories of Logic and Probabilities*. Walton and Maberly, 1854.

[BV06]   R. Bělohlávek and V. Vychodil. Fuzzy horn logic i. *Archive for Mathematical Logic*, 45(1):3–51, 2006.

[Coo03]   Stephen Cook. The importance of the P versus NP question. *Journal of the ACM*, 50(1):27–29, 2003.

[Del17]   J.P. Delgrande. A knowledge level account of forgetting. *Journal of Artificial Intelligence Research*, 60:1165–1213, 2017.

[DW15]   J.P. Delgrande and K. Wang. A syntax-independent approach to forgetting in disjunctive logic programs. In *Proceedings of the Twenty-Ninth Conference on Artificial Intelligence (AAAI 2015)*, pages 1482–1488. AAAI Press, 2015.

[EF07]   E. Erdem and P. Ferraris. Forgetting actions in domain descriptions. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, pages 409–414. AAAI Press, 2007.

[EIS+06]   T. Eiter, G. Ianni, R. Schindlauer, H. Tompits, and K. Wang. Forgetting in managing rules and ontologies. In *2006 IEEE / WIC / ACM International Conference on Web Intelligence (WI 2006), 18-22 December 2006, Hong Kong, China*, pages 411–419. IEEE Computer Society Press, 2006.

[EKI19]   T. Eiter and G. Kern-Isberner. A brief survey on forgetting from a knowledge representation and perspective. *KI — Kuenstliche Intelligenz*, 33(1):9–33, 2019.

[EL00]   T. Eiter and T. Lukasiewicz. Default reasoning from conditional knowledge bases: Complexity and tractable cases. *Artificial Intelligence*, 124(2):169–241, 2000.

[FAS+20]   R. Feng, E. Acar, S. Schlobach, Y. Wang, and W. Liu. On sufficient and necessary conditions in bounded ctl: A forgetting approach. In *Proceedings of the Seventeenth International Conference on Principles of Knowledge Representation and Reasoning, (KR 2020)*, pages 361–370, 2020.

[FHMV95]   R. Fagin, J.Y. Halpern, Y. Moses, and M. Vardi. *Reasoning about knowledge*. The MIT Press, 1995.

[GKL16]   R. Gonçalves, M. Knorr, and 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)*, pages 135–144. AAAI Press, 2016.

[GKLW17]   R. Gonçalves, M. Knorr, J. Leite, and S. Woltran. When you must forget: Beyond strong persistence when forgetting in answer set programming. *Theory and Practice of Logic Programming*, 17(5-6):837–854, 2017.

[Got92]   G. Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2(3):397–425, 1992.

[KRH13]   M. Krötzsch, S. Rudolph, and P. Hitzler. Complexities of horn description logics. *ACM Transactions on Computational Logic*, 14(1):1–36, 2013.

[KWW09]   B. Konev, D. Walther, and F. Wolter. Forgetting and uniform interpolation in extensions of the description logic EL. In *Proceedings of the 22nd International Workshop on Description Logics (DL 2009)*, volume 9, 2009.

[Lib20a]   P. Liberatore. Common equivalence and size after forgetting. *Computing Research Repository (CoRR)*, abs/2006.11152, 2020.

[Lib20b]   P. Liberatore. The ghosts of forgotten things: A study on size after forgetting. *Computing Research Repository (CoRR)*, abs/2005.04123, 2020.

[Lib20c]   P. Liberatore. One head is better than two: a polynomial restriction for propositional definite horn forgetting. *Computing Research Repository (CoRR)*, abs/2009.07497, 2020.

[Lib20d]   P. Liberatore. Reconstructing a single-head formula to facilitate logical forgetting. Unpublished, 2020.

[LLM03]   J. Lang, P. Liberatore, and P. Marquis. Propositional independence — formula-variable independence and forgetting. *Journal of Artificial Intelligence Research*, 18:391–443, 2003.

[LM10]   J. Lang and P. Marquis. Reasoning under inconsistency: A forgetting-based approach. *Artificial Intelligence*, 174(12-13):799–823, 2010.

[LR94]   F. Lin and R. Reiter. Forget it! In *Proceedings of the AAAI Fall Symposium on Relevance*, pages 154–159, 1994.

[Mak87]   J.A. Makowsky. Why horn formulas matter in computer science: Initial structures and generic examples. *Journal of Computer and System Sciences*, 34(2-3):266–292, 1987.

[Moi07]   Y. Moinard. Forgetting literals with varying propositional symbols. *Journal of Logic and Computation*, 17(5):955–982, 2007.

[NCL07]   A. Nayak, Y. Chen, and F. Lin. Forgetting and update - an exploration. In *Dagstuhl Seminar Proceedings*, 2007.

[RHPT14]   D. Rajaratnam, Levesque. H.J., M. Pagnucco, and M. Thielscher. Forgetting in action. In *Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2014)*. AAAI Press, 2014.

[vDHLM09] H. van Ditmarsh, A. Herzig, J. Lang, and P. Marquis. Introspective forgetting. *Synthese*, 169:809–827, 2009.

[WSS05]   K. Wang, A. Sattar, and K. Su. A theory of forgetting in logic programming. In *Proceedings of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence*, pages 682–688. AAAI Press/The MIT Press, 2005.

[WZZZ14]   Y. Wang, Y. Zhang, Y. Zhou, and M. Zhang. Knowledge forgetting in answer set programming. *Journal of Artificial Intelligence Research*, 50, 05 2014.

[ZP12]   Z. Zhuang and M. Pagnucco. Model based horn contraction. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*, pages 169–178. AAAI Press/The MIT Press, 2012.

[ZZ11]   Y. Zhou and Y. Zhang. Bounded forgetting. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*. AAAI Press, 2011.