SLD-Resolution Reduction of Second-Order Horn Fragments

Sophie Tourret[1(✉)[0000−0002−6070−796X]] and Andrew Cropper[2[0000−0002−4543−7199]]

[1] Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
sophie.tourret@mpi-inf.mpg.de
[2] University of Oxford, Oxford, UK
andrew.cropper@cs.ox.ac.uk

Abstract. We present the *derivation reduction* problem for SLD-resolution, the undecidable problem of finding a finite subset of a set of clauses from which the whole set can be derived using SLD-resolution. We study the reducibility of various fragments of second-order Horn logic with particular applications in Inductive Logic Programming. We also discuss how these results extend to standard resolution.

1 Introduction

Detecting and eliminating redundancy in a clausal theory (a set of clauses) is useful in many areas of computer science [3, 20]. Eliminating redundancy can make a theory easier to understand and may also have computational efficiency advantages [9]. The two standard criteria for redundancy are entailment [29, 30, 35] and subsumption [5, 17, 39]. In the case of entailment, a clause C is redundant in a clausal theory $T \cup \{C\}$ when $T \models C$. In the case of subsumption, a clause C is redundant in a clausal theory $T \cup \{C\}$ when there exists a clause $D \in T$ such that D subsumes C. For instance, consider the clausal theory T_1:

$$C_1 = p(x) \leftarrow q(x)$$
$$C_2 = p(x) \leftarrow q(x), r(x)$$

The clause C_2 is entailment and subsumption redundant because it is a logical consequence of C_1 (and is also subsumed by C_1). However, as we will soon show, entailment and subsumption redundancy can be too strong for some applications. To overcome this issue, we introduce a new form of redundancy based on whether a clause is *derivable* from a clausal theory using SLD-resolution [27]. Let $\vdash^*$ represent derivability in SLD-resolution. Then a Horn clause C is *derivationally redundant* in a Horn theory $T \cup \{C\}$ when $T \vdash^* C$. For instance, in T_1, although C_1 entails C_2, we cannot derive C_2 from C_1 using SLD-resolution because it is impossible to derive a clause with three literals from a clause with two literals.

We focus on whether theories formed of second-order function-free Horn clauses can be derivationally reduced to minimal (i.e. irreducible) finite theories from which the original theory can be derived using SLD-resolution. For instance, consider the following theory T_2, where the symbols P_i represent second-order variables (i.e. variables that can be substituted by predicate symbols):

$$C_1 = P_0(x) \leftarrow P_1(x)$$
$$C_2 = P_0(x) \leftarrow P_1(x), P_2(x)$$
$$C_3 = P_0(x) \leftarrow P_1(x), P_2(x), P_3(x)$$

Although C_1 subsumes C_2 and C_3, the two clauses cannot be derived from C_1 for the same reason as in the previous example. However, C_3 is derivationally redundant because it can be derived by self-resolving C_2. A minimal *derivation reduction* of T_2 is the theory $\{C_1, C_2\}$ because C_2 cannot be derived from C_1 and vice versa.

1.1 Motivation

Our interest in this form of redundancy comes from Inductive Logic Programming (ILP) [34], a form of machine learning which induces hypotheses from examples and background knowledge, where the hypotheses, examples, and background knowledge are represented as logic programs. Many forms of ILP [1, 10, 26, 36, 41, 45] and ILP variants [6, 16, 43, 48] use second-order Horn clauses as templates to denote the form of programs that may be induced. For instance, consider the father kinship relation:

$$father(A, B) \leftarrow parent(A, B), male(A).$$

A suitable clause template to induce this relation is:

$$P_0(A, B) \leftarrow P_1(A, B), P_2(A).$$

Determining which clauses to use for a given learning task is a major open problem in ILP [8, 9, 36], and most approaches uses clauses provided by the designers of the systems without any theoretical justifications [1, 6, 10, 26, 43, 45, 48]. The problem is challenging because on the one hand, you want to provide clauses sufficiently expressive to solve the given learning problem. For instance, it is impossible to learn the father relation using only monadic clauses. On the other hand, you want to remove redundant clauses to improve learning efficiency [9].

To illustrate this point, suppose you have the theory T_3:

$$C_1 = P_0(A, B) \leftarrow P_1(A, B)$$
$$C_2 = P_0(A, B) \leftarrow P_1(A, B), P_2(A)$$
$$C_3 = P_0(A, B) \leftarrow P_1(A, B), P_2(A, B)$$
$$C_4 = P_0(A, B) \leftarrow P_1(A, B), P_2(A, B), P_3(A, B)$$

Running entailment reduction on T_3 would remove C_2, C_3, and C_4 because they are logical consequence of C_1. But it is impossible to learn the intended father relation given only C_1. By contrast, running derivation reduction on T_3 would only remove C_4 because it can be *derived* by self-resolving C_3. As this example illustrates, any clause removed by derivation reduction can be recovered by derivation if necessary, while entailment reduction can be too strong and remove important clauses with no way to get them back using SLD-resolution. In this paper, we address this issue by studying the derivation reducibility of fragments of second-order Horn logic relevant to ILP. Although our notion of derivation reduction can be defined for any proof system, we initially focus on SLD-resolution because (1) most forms of ILP learn definite logic programs (typically Prolog programs), and (2) we want to reduce sets of metarules, which are themselves definite clauses (although second-order rather than first-order). The logic fragments we consider here also correspond to the search spaces typically targeted by ILP systems.

1.2 Contributions

Our main contributions are:

- We state the derivation reduction problem for SLD-resolution (Sect. 3) that we originally introduced in [12].
- We describe fragments of second-order Horn logic particularly relevant for ILP (Sect. 4).
- We show that, by constraining the arity of the predicates, an infinite fragment of connected Horn clauses can be derivationally reduced to a finite fragment made of clauses that contain at most two literals in the body (Sect. 5).
- We show that an infinite fragment of 2-connected (i.e. connected and without singleton occurrences of variables) Horn clauses *cannot* be derivationally reduced to any finite fragments (Sect. 6).
- We show similar but incomplete negative results for a more expressive 2-connected fragment (Sect. 7).
- We extend the reducibility results to standard resolution (Sect. 8).

A technical report including detailed proofs of all the results (including the ones only sketched in this paper) has been created as a separate document [47].

2 Related Work

In clausal logic there are two main forms of redundancy: (1) a literal may be redundant in a clause, and (2) a clause may be redundant in a clausal theory.

Literal Redundancy. Plotkin [39] used subsumption to decide whether a literal is redundant in a first-order clause. Joyner [25] independently studied the same problem, which he called *clause condensation*, where a condensation of a clause C is a minimum cardinality subset C' of C such that $C' \models C$. Gottlob and Fermüller [17] showed that determining whether a clause is condensed is coNP-complete. In contrast to eliminating literals from clauses, we focus on removing *clauses* from theories.

Clause Redundancy. Plotkin [39] also introduced methods to decide whether a clause is subsumption redundant in a first-order clausal theory. The same problem, and slight variants, has been extensively studied in the propositional logic [29,30] and has numerous applications, such as to improve the efficiency of SAT solving [20]. This problem has also been extensively studied in the context of first-order logic with equality due to its application in superposition-based theorem proving [21, 49]. Langlois et al. [28] studied combinatorial problems for propositional Horn clauses. Their results include bounds on entailment reduced sets of propositional Horn fragments. In contrast to these works, we focus on removing *second-order* Horn clauses (without equality) that are *derivationally* redundant.

Much closer to this paper is the work of Cropper and Muggleton [9]. They used entailment reduction [35] on sets of second-order Horn clauses to identify theories that are (1) entailment complete for certain fragments of second-order Horn logic, and (2) minimal or irreducible, in that no further reductions are possible. They demonstrate that in some cases as few as two clauses are sufficient to entail an infinite language.

In contrast to all these works, we go beyond entailment reduction and introduce *derivation reduction* because, as stated in the previous section, the former can be too strong to be of use in ILP. Thus our focus is on *derivationally* reducing sets of *second-order Horn* clauses.

Theory Minimisation and Program Transformation. In theory minimisation [19] the goal is to find a minimum equivalent formula to a given input formula. The fold/unfold transformations of first-order rules are used, e.g. to improve the efficiency of logic programs or to synthesise definite programs from arbitrary specifications [44]. Both allow for the introduction of new formulæ. By contrast, the derivation reduction problem only allows for the *removal* of redundant clauses.

Prime Implicates. Implicates of a theory T are the clauses entailed by T. They are called prime when they do not themselves entail other implicates of T. This notion differs from the redundancy elimination in this paper because (1) the notion of a prime implicate has been studied only in propositional, first-order, and some modal logics [4, 15, 32], and (2) implicates are defined using entailment, which as already stated is too strong for our purpose.

Descriptive Complexity. Second-order Horn logic is often the focus in descriptive complexity [24], which studies how expressive a logic must be to describe a given formal language. For instance, Grädel showed that existential second-order Horn logic can describe all polynomial-time algorithms [18]. In this paper, we do not study the expressiveness of the logic but whether the logic can be logically reduced.

Higher-Order Calculi. SLD-resolution on second-order clauses, as used in this paper, supports the unification of predicate variables. By contrast, there are extensions of SLD-resolution and standard resolution that handle the full expressivity of higher-order logic [7,23]. These richer extensions handle more complex clauses, e.g. clauses including function symbols and λ-terms. We do not consider such complex clauses because most ILP approaches use second-order Horn clauses to learn function-free first-order Horn programs [1, 10, 16, 26, 36]. Extending our results to full higher-order logic is left for future work.

Second-Order Logic Templates. McCarthy [33] and Lloyd [31] advocated using second-order logic to represent knowledge. Similarly, in [37], the authors argued for using second-order representations in ILP to represent knowledge. As mentioned in the introduction, many forms of ILP use second-order Horn clauses as a form of declarative bias [40] to denote the structure of rules that may be induced. However, most approaches either (1) assume correct templates as input, or (2) use clauses without any theoretical justifications. Recent work [9] has attempted to address this issue by reasoning about the completeness of these templates, where the goal is to identify finite sets of templates sufficiently expressive to induce all logic programs in a given fragment. Our work contributes to this goal by exploring the derivation redundancy of sets of templates.

Derivation Reduction. In earlier work [12] we introduced the derivation reduction problem and a simple algorithm to compute reduction cores. We also experimentally studied the effect of using derivationally reduced templates on ILP benchmarks. Whereas our earlier paper mainly focuses on the application of derivation reduction to ILP, the

current paper investigates derivation reduction itself in a broader perspective, with more emphasis on whether infinite fragments can be reduced to finite subsets. Another main distinction between the two papers is that here we focus on derivation reduction modulo first-order variable unification. The overlap includes the definition of derivation reduction and Sect. 4.2 in [12] which covers in less detail the same topic as our Sect. 6.

3 Problem Statement and Decidability

We now define the derivation reduction problem, i.e. the problem of removing derivationally redundant clauses from a clausal theory.

3.1 Preliminaries

We focus on function-free second-order Horn logic. We assume infinite enumerable sets of *term variables* $\{x_1, x_2, ..\}$ and *predicate variables* $\{P, P_0, P_1, ..\}$. An *atom* $P(x_{k_1}, .., x_{k_a})$ consists of a predicate variable P of arity a followed by a term variables. A *literal* is an atom (*positive* literal) or the negation of an atom (*negative* literal). A *clause* is a finite disjunction of literals. A *Horn clause* is a clause with at most one positive literal. From this point on, we omit the term *Horn* because all clauses in the rest of the paper are Horn clauses (λ-free function-free second-order Horn clauses to be precise). The positive literal of a clause C, when it exists, is its *head* and is denoted as $h(C)$. The set of negative literals of C is called its *body* and is denoted as $b(C)$. The clause C is written as $h(C) \leftarrow b(C)$. We denote the empty clause as $\square$. We denote the number of literals occurring in $b(C)$ as $|\mathsf{b}(C)|$, i.e. the body size of C. A *theory* T is a set of clauses.

A *substitution* σ is a function mapping term variables to term variables, and predicate variables to predicate variables with the same arity. The application of a substitution σ to a clause C is written $C\sigma$. A substitution σ is a *unifier* of two literals when they are equal after substitution. A substitution σ is a *most general unifier* of two literals, denoted as m.g.u., when no smaller substitution is also a unifier of the two literals, i.e. there exist no σ' and γ such that σ' unifies the two literals and $\sigma = \sigma' \circ \gamma$. The variables in a clause are implicitly universally quantified. In practice, ILP approaches typically use existentially quantified predicate variables [1, 9, 16, 36]. However, we ignore the quantification of the predicate variables because we are not concerned with the semantics of the clauses, only their syntactic form.

3.2 Derivation Reduction

The derivation reduction problem can be defined for any proof system but we focus on SLD-resolution [27] because of the direct application to ILP. SLD-resolution is a restricted form of resolution [42] based on linear resolution with two main additional constraints (1) it is restricted to Horn clauses, and (2) it does not use factors, where factoring unifies two literals in the same clause during the application of the resolution inference rule (this implies that all resolvents are binary resolvents). SLD-resolution is usually defined for first-order logic. To apply it to the second-order clauses in this

paper, we replace the standard notion of a m.g.u. with the one defined in the previous paragraph that also handles predicate variables. An SLD-resolution inference is denoted as $C_1, C_2 \vdash C$ where the necessary m.g.u. is implicitly applied on C. The clauses C_1 and C_2 are the *premises* and C is the *resolvent* of the inference. The literal being resolved upon in C_1 and C_2 is called the *pivot* of the resolution. We define a function $S^n(T)$ of a theory T as:

$$S^0(T) = T$$
$$S^n(T) = \{C | C_1 \in S^{n-1}(T), C_2 \in T, \text{s.t. } C_1, C_2 \vdash C\}$$

The *SLD-closure* of a theory T is defined as:

$$S^*(T) = \bigcup_{n \in \mathbb{N}} S^n(T)$$

A clause C is *derivable* from the theory T, written $T \vdash^* C$, if and only if $C \in S^*(T)$. Given a theory T, a clause $C \in T$ is *reducible* if it is the resolvent of an inference whose premises all belong to T and have a body size smaller than $|\mathsf{b}(C)|$. A clause C is *redundant* in the theory $T \cup \{C\}$ if and only if $T \vdash^* C$. By extension, a theory T is *redundant* to another theory $T' \subseteq T$ if for all $C \in T$, $T' \vdash^* C$. A theory is *reduced* if and only if it does not contain any redundant clauses. We state the *reduction problem*:

Definition 1 (Reduction Problem). *Given a possibly infinite theory T, the reduction problem is to find a finite theory $T' \subseteq T$ such that (1) T is redundant to T', and (2) T' is reduced. In this case, we say that T' is a* reduction core *of T.*

Note that in the case of a finite theory T, the existence of a reduction core is obvious since at worst it is T itself. However, for arbitrary theories it is impossible to compute or a reduction core because the derivation reduction problem is undecidable [12].

4 Fragments of Interest in $\mathcal{H}$

From Sect. 5 onwards we study whether derivationally reduced theories exist for various fragments of Horn logic. Horn logic with function symbols has the expressive power of Turing machines and is consequently undecidable [46], hence ILP approaches typically learn programs without function symbols [38], which are decidable [13]. We therefore focus on function-free Horn clauses. We denote the set of all second-order function-free Horn clauses as $\mathcal{H}$.

We further impose syntactic restrictions on clauses in $\mathcal{H}$ principally on the arity of the literals and on the number of literals in the clauses. Let us consider a fragment $\mathcal{F}$ of $\mathcal{H}$. We write $\mathcal{F}_{a,b}$ to denote clauses in $\mathcal{F}$ that contain literals of arity at most a and clauses of body size at most b. For example, the clause $P_0(x_1) \leftarrow P_1(x_2, x_3, x_4)$ is in $\mathcal{H}_{3,1}$. When one of these restrictions is not imposed, the symbol ∞ replaces the corresponding number. When restrictions are imposed on a fragment that is already restricted, the stricter restrictions are kept. For example, $(\mathcal{H}_{4,1})_{3,\infty} = \mathcal{H}_{3,1} = \mathcal{H}_{4,1} \cap \mathcal{H}_{3,\infty}$. We rely on the body size restriction to bound the reduction cores of the studied fragments.

We also constrain the fragments so that they are defined modulo variable renaming and so that only the most general clauses up to variable unification are considered. Let

C be a clause verifying the syntactic restrictions of a given fragment $\mathcal{F}$. Then there exists a clause $C_{\mathcal{F}} \in \mathcal{F}$ such that $C_{\mathcal{F}}\sigma = C$ for some substitution σ. The motivation behind this restriction is that SLD-resolution only applies m.g.u.s and not any unifiers but some clauses like C' may need more specific unifiers to be generated and can thus be unreachable by SLD-resolution. This is not restrictive because up to variable renaming any such C' can be obtained from C by renaming and unifying variables.

Definition 2 (Reducible fragment). *A fragment $\mathcal{F}$ of $\mathcal{H}$ is reducible to $\mathcal{F}_{\infty,b}$ when, for all $C \in \mathcal{F}$ such that $b < |\mathsf{b}(C)|$, there exists $b' < |\mathsf{b}(C)|$ such that $\mathcal{F}_{\infty,b'} \vdash C$, i.e. C is the resolvent of an inference with premises in $\mathcal{F}_{\infty,b'}$.*

The following results are consequences of this definition and of the reduction problem statement.

Proposition 3 (Reduciblility). *If a fragment $\mathcal{F}$ is reducible to $\mathcal{F}_{\infty,b}$ then $\mathcal{F}$ is redundant to $\mathcal{F}_{\infty,b}$.*

Theorem 4 (Cores of Reducible Fragments). *If a fragment $\mathcal{F}$ is reducible to $\mathcal{F}_{\infty,b}$ then the solutions of the reduction problem for $\mathcal{F}$ and $\mathcal{F}_{\infty,b}$ are the same, i.e. the reduction cores of $\mathcal{F}$ and $\mathcal{F}_{\infty,b}$ are the same.*

Because we are motivated by applications in ILP, we focus on connected clauses [1, 9, 16, 26, 38]:

Definition 5 (Connected Fragment). *A clause is connected if the literals in the clause cannot be partitioned into two non-empty sets such that the variables appearing in the literals of one set are disjoint from the variables appearing in the literals of the other set. The* connected *fragment, denoted as $\mathcal{H}^c$, is the subset of $\mathcal{H}$ where all clauses are connected.*

Example 6. The clause $C_1 = P_0(x_1, x_2) \leftarrow P_1(x_3, x_1), P_2(x_2), P_3(x_3)$ is in $\mathcal{H}^c$, but the clause $C_2 = P_0(x_1, x_2) \leftarrow P_1(x_3, x_4), P_2(x_2), P_3(x_3)$ is not because none of the variables in P_0 and P_2 (x_1 and x_2) appear in P_1 and P_3 and vice versa.

A stricter version of connectedness, denoted here as 2-connectedness, describes the fragment that is used the most in ILP [9]. It essentially eliminates singleton variables.

Definition 7 (2-Connected Fragment). *The* 2-connected *fragment, denoted as $\mathcal{H}^{2c}$, is the subset of $\mathcal{H}^c$ such that all the term variables occur at least twice in distinct literals. In this context, a term variable that does not follow this restriction is denoted as* pending.

Example 8. The clause C_1 from Example 6 is in $\mathcal{H}^{2c}$ because x_1 is in P_0 and P_1, x_2 is in P_0 and P_2, and x_3 is in P_1 and P_3. By contrast, the clause $C_3 = P_0(x_1, x_2) \leftarrow P_1(x_3, x_1), P_2(x_1), P_3(x_3)$ is in $\mathcal{H}^c$ but not in $\mathcal{H}^{2c}$ because x_2 only occurs once and is thus pending.

Note that the simple syntactic restrictions can be combined with both connectedness and 2-connectedness. In the following sections we consider the reduction problem for $\mathcal{H}^c$ (Sect. 5), $\mathcal{H}^{2c}_{2,\infty}$ (Sect. 6), and $\mathcal{H}^{2c}_{3,\infty}$ (Sect. 7).

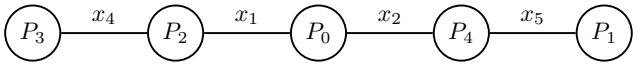


Fig. 1: Encoding of $C = P_0(x_1, x_2) \leftarrow P_2(x_1, x_3, x_4), P_3(x_4), P_4(x_2, x_5), P_1(x_5, x_6)$ where vertices correspond to literals and edges represent variables shared by two literals

5 The Fragment $\mathcal{H}^c$ is Reducible to $\mathcal{H}^c_{\infty,2}$

We now study whether certain fragments can be reduced. Our first focus is on the fragment $\mathcal{H}^c$which contains all connected clauses. We are primarily interested in whether this fragment can be reduced using SLD-resolution to a minimal fragment, preferably with only two literals in the body ($\mathcal{H}^c_{\infty,2}$).

5.1 Graph Encoding

To prove the reducibility of $\mathcal{H}^c$ we consider $\mathcal{H}^c_{a,\infty}$ for any $a \in \mathbb{N}^*$ and show that it can be reduced to $\mathcal{H}^c_{a,2}$. To reduce all clauses in $\mathcal{H}^c_{a,\infty}$ of body size greater than two, we rely on the following graph encoding to create connected premises to infer C. We assume reader familiarity with basic notions of graph theory, in particular, notions of *spanning trees*, *connected graphs*, *degree* of vertices and *outgoing* edges (from a set of vertices).

Definition 9 (Graph Encoding). *Let C be a clause in $\mathcal{H}^c_{m,\infty}$. The undirected graph $\mathcal{G}_C$ is such that:*

- *There is a bijection between the vertices of $\mathcal{G}_C$ and the predicate variable occurrences in C (head and body).*
- *There is an edge in $\mathcal{G}_C$ between each pair of vertices for each corresponding pair of literals that share a common term variable. The edge is labeled with the corresponding variable.*

Example 10. $C = P_0(x_1, x_2) \leftarrow P_2(x_1, x_3, x_4),\ P_3(x_4),\ P_4(x_2, x_5),\ P_1(x_5, x_6)$ is mapped to $\mathcal{G}_C$ as illustrated in Fig. 1. Note that since the variables x_3 and x_6 occur only in P_2 and P_1 respectively, they are not present in $\mathcal{G}_C$. In fact $\mathcal{G}_C$ also represents many other clauses, e.g. $P_1(x_5, x_5) \leftarrow P_0(x_2, x_1), P_2(x_4, x_3, x_1), P_3(x_4), P_4(x_2, x_5)$.

This graph encoding allows us to focus on connectivity, as stated in the following proposition.

Proposition 11. *Let $C \in \mathcal{H}$. The graph $\mathcal{G}_C$ is connected if and only if $C \in \mathcal{H}^c$.*

In other words, the notion of connectedness that we introduced for clauses in Def. 5 is equivalent to graph connectedness when encoding the clauses in graph form using Def. 9. Because we are only interested in connected clauses, we only handle connected graphs.

5.2 Reducibility of $\mathcal{H}^c$

Proposition 12 is the main intermediary step in the proof of reducibility of the connected fragment (Th. 13). A detailed proof of this result is available in the technical report version of this paper [47].

Proposition 12 (Spanning Tree). *For any clause $C \in \mathcal{H}^c_{a,\infty}$, $a \in \mathbb{N}^*$, there exists a spanning tree of $\mathcal{G}_C$ in which there exist two adjacent vertices such that the number of edges outgoing from this pair of vertices is at most a.*

Proof sketch. Assuming no such pair of vertices exists in any spanning tree of $\mathcal{G}_C$, we show in a case analysis that it is always possible to transform a spanning tree into another one where such a pair exists, a contradiction.

The main result of this section is the next theorem stating that any connected fragment of constrained arity has a reduction core containing clauses of body size at most two.

Theorem 13 (Reducibility of $\mathcal{H}^c_{a,\infty}$). *For any $a \in \mathbb{N}^*$, $\mathcal{H}^c_{a,\infty}$ is reducible to $\mathcal{H}^c_{a,2}$.*

Proof. Let $a \in \mathbb{N}^*$ be fixed and $C = P_0(..) \leftarrow P_1(..), .., P_k(..) \in \mathcal{H}^c_{a,\infty}$ ($k \geq 3$). By applying Prop. 12, it is possible to identify two adjacent vertices v and v' in $\mathcal{G}_C$ such that there exists a spanning tree $\mathcal{S}$ of $\mathcal{G}_C$ where the number of edges outgoing from the pair v, v' is less than or equal to a. Let P_v and $P_{v'}$ be the predicate variables respectively corresponding to v and v' in C. Let $x_1, .., x_{a'}$ ($a' \leq a$) be the variables corresponding to the edges outgoing from the pair of vertices v, v'. Let P'_0 be an unused predicate variable of arity a'. We define: $C_1 = P_0(..) \leftarrow P'_0(x_1, .., x_{a'}), P_1(..), .., P_k(..) \backslash \{P_v(..), P_{v'}(..)\}$ and $C_2 = P'_0(x_1, .., x_{a'}) \leftarrow P_v(..), P_{v'}(..)$. These clauses are such that $C_1, C_2 \in \mathcal{H}^c_{a',\infty}$ and $C_1, C_2 \vdash C$ modulo variable unification.[1] Thus, C is reducible.

We extend this result to the whole connected fragment.

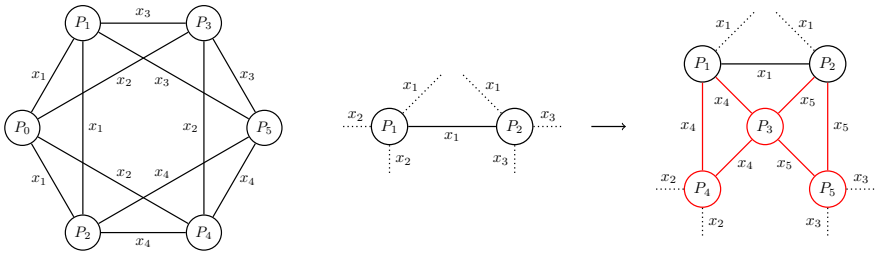
Theorem 14 (Reducibility of $\mathcal{H}^c$). *The fragment $\mathcal{H}^c$ is reducible to $\mathcal{H}^c_{\infty,2}$.*

Note that Theorem 14 does not imply that $\mathcal{H}^c$ has a reduction core because $\mathcal{H}^c_{\infty,2}$ is also infinite. In fact, since it is not possible to increase the arity of literals through SLD-resolution, any fragment where this arity is not constrained is guaranteed to have no reduction core since at least one literal of each arity must occur in it and the number of literals that occur in a clause is finite.

6 Reducibility of $\mathcal{H}^{2c}_{2,\infty}$

We now consider the reducibility of $\mathcal{H}^{2c}_{2,\infty}$. The restriction to monadic and dyadic literals is common not only in ILP [1, 6, 16, 36] but also in description logics [2] and in ontology reasoning [22]. Although this fragment is only slightly more constrained than $\mathcal{H}^c_{2,\infty}$,

[1] Some connections may be lost between variables in C_1 and C_2 since only the ones occurring in the spanning tree $\mathcal{S}$ are preserved. However, they can be recovered by unifying the disconnected variables together in the resolvent.



(a) Graph encoding of C_{base}, $\mathcal{G}_{C_{base}}$

(b) Partial graph encoding of a clause before and after a non-red preserving transformation

Fig. 2: Graph encoding of $\mathcal{H}^{nr}$ base and construction rule

itself reducible to $\mathcal{H}^{c}_{2,2}$, we show that it is impossible to reduce $\mathcal{H}^{2c}_{2,\infty}$ to any size-constrained sub-fragment. To do so we exhibit a subset $\mathcal{H}^{nr}$ in $\mathcal{H}^{2c}_{2,\infty}$ that cannot be reduced. This set contains clauses of arbitrary size. In practice, this means that in $\mathcal{H}^{2c}_{2,\infty}$ given any integer k it is possible to exhibit a clause of body size superior or equal to k that cannot be reduced, thus preventing $\mathcal{H}^{2c}_{2,\infty}$ itself to be reducible to $\mathcal{H}^{2c}_{2,k}$ no matter how big k is. We start by defining the clause $C_{base} \in \mathcal{H}^{nr}$.

Definition 15 **(C_{base}).**

$$C_{base} = P_0(x_1, x_2) \leftarrow P_1(x_1, x_3), P_2(x_1, x_4), P_3(x_2, x_3), P_4(x_2, x_4), P_5(x_3, x_4).$$

In C_{base} all the literals are symmetrical to each other. Each literal (vertex) has (1) two neighbours connected by their first variable, (2) two other neighbours connected by their second variable, and (3) another literal that it is not connected to but which all the other literals are. This symmetry is better seen on the graphical representation of C_{base} in Fig. 2a. For example P_0 does not share literals with P_5 but does with all other predicates.

Proposition 16 **(Non-reducibility of C_{base}).** *C_{base} is irreducible.*

Proof. To derive C_{base} from two smaller clauses, these two smaller clauses C_1 and C_2 must form a partition of the literals in C_{base} if one excludes the pivot. To solve this problem, we partition the vertices of $\mathcal{G}_{C_{base}}$ in two sets and count the number of edges with distinct labels that link vertices from the two sets. These edges correspond to pending variables in one of the sets, i.e. to the variables that must occur in the pivot that will be added in both sets to form C_1 and C_2. If there are more than two of these variables, the pivot cannot contain all of them, thus at least one of C_1 and C_2 is not in $\mathcal{H}^{2c}_{2,\infty}$ for lack of 2-connectivity. Each of the two sets in the partition must contain at least two elements, otherwise one of C_1, C_2 is as big as C_{base} which does not make C_{base} reducible even though it is derivable from C_1, C_2. The symmetries in $\mathcal{G}_{C_{base}}$ are exploited to reduce the number of cases to consider to only four that vary along two dimensions: the cardinalities of the two subsets, either 2-4 or 3-3 respectively; and the connectedness of the subsets. In the 2-4 partition, only the following cases or symmetric ones are possible:

- if $\{P_0, P_5\}$ is the subset of cardinality 2 in a 2-4 partition, then the edges outgoing from this subset, connecting the two subsets and that correspond to pending variables, are labeled with x_1, x_2, x_3 and x_4;
- if $\{P_0, P_1\}$ is the subset of cardinality 2 in a 2-4 partition, then the outgoing edges are labeled with x_1, x_2 and x_3.

All the remaining 2-4 cases where P_0 is in the subset of cardinality 2 are symmetric to this case. The other 2-4 cases are symmetric to either one of these two cases. Similarly, all the 3-3 partition are symmetric to one of the following cases:

- if $\{P_0, P_1, P_2\}$ is one of the subsets in a 3-3 partition then the outgoing edges are labeled with x_2, x_3 and x_4;
- if $\{P_0, P_1, P_4\}$ is one of the subsets in a 3-3 partition then the outgoing edges are labeled with x_1, x_2, x_3 and x_4.

In all cases, there are 3 or more distinct labels on the edges between the two subsets, corresponding to pending variables, thus C_{base} is irreducible. Note that this proof works because there are exactly three occurrences of each variable in C_{base}. Otherwise it would not be possible to match the labels with the pending variables.

We define a transformation that turns a clause into a bigger clause (Def. 17) such that when applied to an irreducible clause verifying some syntactic property, the resulting clause is also irreducible (Prop.18).

Definition 17 (Non-red Preserving Extension). *Let the body of a clause $C \in \mathcal{H}^{2c}_{2,\infty}$ contain two dyadic literals sharing a common variable, e.g. $P_1(x_1, x_2)$ and $P_2(x_1, x_3)$, without loss of generality. A* non-red preserving *extension of C is any transformation which replaces two such literals in C by the following set of literals: $P_1(x_1, x_4)$, $P_2(x_1, x_5)$, $P_3(x_4, x_5)$, $P_4(x_4, x_2)$, $P_5(x_5, x_3)$ where P_3, P_4, P_5, x_4 and x_5 are new predicate and term variables.*

Proposition 18 (Non-red Preserving Extension). *If a clause C is irreducible and all the term variables it contains occur three times then any non-red preserving extension of C is also irreducible.*

Proof sketch. We assume that a non-red preserving extension of C is reducible and we use a case analysis to show that this implies that C is also reducible, a contradiction. This proof heavily exploits the symmetry that can be seen on Fig. 2b to reduce the number of cases to consider.

Starting from C_{base} and using this extension, we define $\mathcal{H}^{nr}$ formally (Def. 19) and, as a consequence of Prop. 18, $\mathcal{H}^{nr}$ contains only irreducible clauses (Prop. 20).

Definition 19 (Non-reducible Fragment). *The subset $\mathcal{H}^{nr}$ of $\mathcal{H}^{2c}_{2,\infty}$ contains C_{base} and all the clauses that can be obtained by applying a non-red extension to another clause in $\mathcal{H}^{nr}$.*

Proposition 20 (Non-reducibility of $\mathcal{H}^{nr}$). *For all $C \in \mathcal{H}^{nr}$, C is irreducible.*

The non-reducibility of $\mathcal{H}^{nr}$ ensures that the body size of the clauses in a hypothetical reduction core of $\mathcal{H}^{2c}_{2,\infty}$ cannot be bounded, which in turn prevents the existence of this reduction core. This result has negative consequences on ILP approaches that use second-order templates. We discuss these consequences in the conclusion.

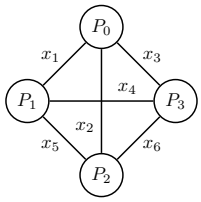


Fig. 3: $\mathcal{G}_C$ for $C = P_0(x_1, x_2, x_3) \leftarrow P_1(x_1, x_4, x_5), P_2(x_2, x_5, x_6), P_3(x_3, x_4, x_6)$

7 Reducibility of $\mathcal{H}^{2c}_{3,\infty}$

The reducibility of $\mathcal{H}^{2c}_{3,\infty}$ is still an open problem. However, we know that it cannot be reduced to $\mathcal{H}^{2c}_{3,2}$.

Theorem 21 (Non-reducibility of $\mathcal{H}^{2c}_{3,2}$). *$\mathcal{H}^{2c}_{3,\infty}$ cannot be reduced to $\mathcal{H}^{2c}_{3,2}$*

Proof. The clause $C = P_0(x_1, x_2, x_3) \leftarrow P_1(x_1, x_4, x_5),\ P_2(x_2, x_5, x_6),\ P_3(x_3, x_4, x_6)$, shown in graph form in Fig. 3, is a counter-example because any pair of literals in it contain exactly four pending variables. For example, consider the following pair of literals: $(P_1(x_1, x_4, x_5), P_0(x_1, x_2, x_3))$ leaves x_2, x_3, x_4, x_5 pending. By symmetry the same holds for all the other pairs of literals. Thus none of these pairs can be completed by a triadic (or less) pivot. In addition, the removal of any single literal from C does not lead to a reduction of the clause since all the variables occurring in the literal then occur only once in each subset of the clause. For example, to replace $P_1(x_1, x_4, x_5)$, a triadic literal containing x_1, x_4 and x_5 needs to be added, creating a clause identical to C up to the name of one predicate variable and the order of the term variables in it. Therefore C is irreducible in $\mathcal{H}^{2c}_{3,\infty}$, thus $\mathcal{H}^{2c}_{3,\infty}$ cannot be reduced to $\mathcal{H}^{2c}_{3,2}$.

In addition to this result, for lack of finding a reduction to $P_0(x_1, x_2, x_3) \leftarrow P_1(x_1, x_5, x_6), P_2(x_2, x_4, x_8), P_3(x_6, x_7, x_8), P_4(x_4, x_5, x_7), P_5(x_3, x_4, x_7)$ (not formally proved) we conjecture that $\mathcal{H}^{2c}_{3,\infty}$ cannot be reduced to $\mathcal{H}^{2c}_{3,4}$. Clarifying this situation and that of any $\mathcal{H}^{2c}_{a,\infty}$ with $a \geq 3$ is left as future work.

8 Extension to Standard Resolution

Although we introduced the derivation reduction problem for SLD-resolution, the principle applies to any standard deductive proof system, and in particular, it can be applied to standard resolution, extended from first to second-order logic in the same way that was used for SLD-resolution. Given that SLD-resolution is but a restriction of resolution, the positive reducibility result for $\mathcal{H}^{c}_{2,\infty}$ is directly transferable to standard resolution. On the contrary, the fragment $\mathcal{H}^{2c}_{2,\infty}$, that we proved irreducible with SLD-resolution, can be reduced to $\mathcal{H}^{2c}_{2,2}$ with standard resolution.

Theorem 22 (Reducibility$_R$ of $\mathcal{H}^{2c}_{2,\infty}$). *$\mathcal{H}^{2c}_{2,\infty}$ is reducible$_R$ to $\mathcal{H}^{2c}_{2,2}$*

Table 1: Summary of the results. When a fragment is preceded with $>$ the entry must be read as "no reduction up to this fragment". The word *possibly* precedes results that have not been proved and are only conjectured.

	Reducibility	
Fragment	SLD-resolution	Standard resolution
$\mathcal{H}^c$	$\mathcal{H}^c_{\infty,2}$	$\mathcal{H}^c_{\infty,2}$
$\mathcal{H}^{2c}_{2,\infty}$	no	$\mathcal{H}^{2c}_{2,2}$
$\mathcal{H}^{2c}_{3,\infty}$	$> \mathcal{H}^{2c}_{3,2}$	$> \mathcal{H}^{2c}_{3,2}$
	possibly $> \mathcal{H}^{2c}_{3,4}$	possibly $> \mathcal{H}^{2c}_{3,4}$

Proof sketch. We first analyse the structure of C and show how to reduce C in the simple cases where it is also possible to reduce C using SLD-resolution. We are then left to consider clauses where C contains only dyadic predicates, no two predicates in C have the same pair of variables and all variables occur exactly three times in C. An example of such clauses is the $\mathcal{H}^{nr}$ family from Sect. 6. Then we present a method to reduce$_R$ such a clause C. The key point that justifies Th. 22 is that in standard resolution, factorisation is allowed and thus allows inferences that remove duplicate literals. The removal of duplicate literals would be also possible with SLD-resolution but only when the fragment contains bodyless clauses which is prevented by 2-connectedness.

Let us consider an example of additional inferences allowed with resolution but not with SLD-resolution in the $\mathcal{H}^{2c}_{2,\infty}$ fragment, that make the C_{base} clause redundant:

$$\begin{array}{l} P_0(x_1,x_2) \leftarrow P_1(x_1,x_3), P_2(x_1,x_4), P_3(x_2,x_3), H(x_2,x_4) \\ H'(x'_2,x'_4) \leftarrow P'_3(x'_2,x'_3), P'_4(x'_2,x'_4), P'_5(x'_3,x'_4) \\ \hline P_0(x_1,x_2) \leftarrow P_1(x_1,x_3), P_2(x_1,x_4), P_3(x_2,x_3), P'_3(x_2,x'_3), P'_4(x_2,x_4), P'_5(x'_3,x_4) \\ \hline P_0(x_1,x_2) \leftarrow P_1(x_1,x_3), P_2(x_1,x_4), P_3(x_2,x_3), P'_4(x_2,x_4), P'_5(x_3,x_4) \end{array}$$

The first step is a resolution that unifies H' with H, x'_2 with x_2 and x'_4 with x_4 and uses $H(x_2,x_4)$ as pivot. The second step is a factorisation that unifies P'_3 with P_3, and x'_3 with x_3. The result is C_{base} up to variable renaming.

Finally, the result that we presented for $\mathcal{H}^{2c}_{3,\infty}$ is also transferable from SLD- to standard resolution since the proof of Th. 21 remains the same. This is because the size of the considered clauses does not allow for the kind of resolution inferences that make Th. 22 possible. Table 1 summarises our findings and their extension to standard resolution.

9 Conclusion

We have introduced the derivation reduction problem for second-order Horn clauses ($\mathcal{H}$), i.e. the undecidable problem of finding a finite subset of a set of clauses from which the whole set can be derived using SLD-resolution. We have considered the derivation reducibility of several fragments of $\mathcal{H}$, for which the results are summarised in Tab. 1. We have also extended the results from SLD-resolution to standard resolution. Further

work is necessary to clarify the situation for $\mathcal{H}^{2c}_{3,\infty}$ and for fragments with higher arity constraints.

Although we have positive results regarding the reducibility of certain fragments, we have not identified the reductions of those fragments, nor have we provided any results regarding the cardinality of the reductions. Future work should address this limitation by introducing algorithms to compute the reductions.

Our results have direct implications in ILP. As described in the introduction, many ILP systems use second-order Horn clauses as templates to define the hypothesis space. An open question [8, 9, 36] is whether there exists finite sets of such clauses from which these systems could induce any logic program in a specific fragment of logic. Prop. 20 shows that for the $\mathcal{H}^{2c}_{2,\infty}$ fragment, which is often the focus of ILP, the answer is no. This result implies that ILP systems, such as Metagol [11] and HEXMIL [26], are incomplete in that they cannot learn all programs in this fragment without being given an infinite set of clauses (these approaches require a finite set of such clauses hence the incompleteness).

Our work now opens up a new challenge of overcoming this negative result for $\mathcal{H}^{2c}_{2,\infty}$ (and negative conjectures for $\mathcal{H}^{2c}_{3,\infty}$). One possible solution would be to allow the use of triadic literals as pivot in inferences in specific cases where SLD-resolution fails to derive the desired clause, but this idea requires further investigation.

Acknowledgements. The authors thank Katsumi Inoue and Stephen Muggleton for discussions on this work.

References

1. Albarghouthi, A., Koutris, P., Naik, M., Smith, C.: Constraint-based synthesis of datalog programs. In: Beck, J.C. (ed.) Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Proceedings. Lecture Notes in Computer Science, vol. 10416, pp. 689–706. Springer (2017). https://doi.org/10.1007/978-3-319-66158-2_44
2. Baader, F., Horrocks, I., Lutz, C., Sattler, U.: An Introduction to Description Logic. Cambridge University Press (2017)
3. Balcázar, J.L.: Redundancy, deduction schemes, and minimum-size bases for association rules. Logical Methods in Computer Science **6**(2) (2010)
4. Bienvenu, M.: Prime implicates and prime implicants in modal logic. In: Proceedings of the National Conference on Artificial Intelligence. vol. 22, p. 379. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press (2007)
5. Buntine, W.: Generalized subsumption and its applications to induction and redundancy. Artificial intelligence **36**(2), 149–176 (1988). https://doi.org/10.1016/0004-3702(88)90001-X
6. Campero, A., Pareja, A., Klinger, T., Tenenbaum, J., Riedel, S.: Logical Rule Induction and Theory Learning Using Neural Theorem Proving. ArXiv e-prints (Sep 2018)
7. Charalambidis, A., Handjopoulos, K., Rondogiannis, P., Wadge, W.W.: Extensional higher-order logic programming. ACM Trans. Comput. Log. **14**(3), 21:1–21:40 (2013). https://doi.org/10.1145/2499937.2499942
8. Cropper, A.: Efficiently learning efficient programs. Ph.D. thesis, Imperial College London, UK (2017)

9. Cropper, A., Muggleton, S.H.: Logical minimisation of meta-rules within meta-interpretive learning. In: Davis, J., Ramon, J. (eds.) Inductive Logic Programming - 24th International Conference, ILP 2014, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9046, pp. 62–75. Springer (2014). https://doi.org/10.1007/978-3-319-23708-4_5
10. Cropper, A., Muggleton, S.H.: Learning higher-order logic programs through abstraction and invention. In: Kambhampati, S. (ed.) Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016. pp. 1418–1424. IJCAI/AAAI Press (2016)
11. Cropper, A., Muggleton, S.H.: Metagol system. https://github.com/metagol/metagol (2016)
12. Cropper, A., Tourret, S.: Derivation reduction of metarules in meta-interpretive learning. In: Inductive Logic Programming - 28th International Conference, ILP 2018, Ferrara, Italy, September 2-4, 2018, Proceedings. pp. 1–21 (2018). https://doi.org/10.1007/978-3-319-99960-9_1
13. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Comput. Surv. **33**(3), 374–425 (2001). https://doi.org/10.1145/502807.502810
14. Dirac, G.A.: Some theorems on abstract graphs (1952). https://doi.org/10.1112/plms/s3-2.1.69
15. Echenim, M., Peltier, N., Tourret, S.: Quantifier-free equational logic and prime implicate generation. In: CADE-25. pp. 311–325. Springer (2015)
16. Evans, R., Grefenstette, E.: Learning explanatory rules from noisy data. J. Artif. Intell. Res. **61**, 1–64 (2018). https://doi.org/10.1613/jair.5714
17. Gottlob, G., Fermüller, C.G.: Removing redundancy from a clause. Artificial Intelligence **61**(2), 263–289 (1993)
18. Grädel, E.: The expressive power of second order horn logic. In: STACS 91, 8th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings. pp. 466–477 (1991). https://doi.org/10.1007/BFb0020821
19. Hemaspaandra, E., Schnoor, H.: Minimization for generalized boolean formulas. In: IJCAI Proceedings-International Joint Conference on Artificial Intelligence. vol. 22, p. 566 (2011)
20. Heule, M., Järvisalo, M., Lonsing, F., Seidl, M., Biere, A.: Clause elimination for SAT and QSAT. Journal of Artificial Intelligence Research **53**, 127–168 (2015)
21. Hillenbrand, T., Piskac, R., Waldmann, U., Weidenbach, C.: From search to computation: Redundancy criteria and simplification at work. In: Voronkov, A., Weidenbach, C. (eds.) Programming Logics - Essays in Memory of Harald Ganzinger. Lecture Notes in Computer Science, vol. 7797, pp. 169–193. Springer (2013). https://doi.org/10.1007/978-3-642-37651-1_7
22. Hohenecker, P., Lukasiewicz, T.: Deep learning for ontology reasoning. CoRR **abs/1705.10342** (2017)
23. Huet, G.P.: A mechanization of type theory. In: Proceedings of the 3rd International Joint Conference on Artificial Intelligence. pp. 139–146 (1973)
24. Immerman, N.: Descriptive complexity. Springer Science & Business Media (2012)
25. Joyner Jr., W.H.: Resolution strategies as decision procedures. J. ACM **23**(3), 398–417 (1976). https://doi.org/10.1145/321958.321960
26. Kaminski, T., Eiter, T., Inoue, K.: Exploiting answer set programming with external sources for meta-interpretive learning. In: 34th International Conference on Logic Programming (2018)
27. Kowalski, R.A.: Predicate logic as programming language. In: IFIP Congress. pp. 569–574 (1974)
28. Langlois, M., Mubayi, D., Sloan, R., Turán, G.: Combinatorial problems for Horn clauses. Graph Theory, Computational Intelligence and Thought pp. 54–65 (2009)
29. Liberatore, P.: Redundancy in logic I: CNF propositional formulae. Artif. Intell. **163**(2), 203–232 (2005). https://doi.org/10.1016/j.artint.2004.11.002

30. Liberatore, P.: Redundancy in logic II: 2CNF and Horn propositional formulae. Artif. Intell. **172**(2-3), 265–299 (2008). https://doi.org/10.1016/j.artint.2007.06.003
31. Lloyd, J.: Logic for Learning. Springer, Berlin (2003)
32. Marquis, P.: Consequence finding algorithms. In: Handbook of Defeasible Reasoning and Uncertainty Management Systems, pp. 41–145. Springer (2000)
33. McCarthy, J.: Making robots conscious of their mental states. In: Machine Intelligence 15, Intelligent Agents [St. Catherine's College, Oxford, July 1995]. pp. 3–17 (1995)
34. Muggleton, S.: Inductive Logic Programming. New Generation Computing **8**(4), 295–318 (1991)
35. Muggleton, S.: Inverse entailment and Progol. New Generation Computing **13**, 245–286 (1995)
36. Muggleton, S., Lin, D., Tamaddoni-Nezhad, A.: Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. Machine Learning **100**(1), 49–73 (2015)
37. Muggleton, S., Raedt, L.D., Poole, D., Bratko, I., Flach, P.A., Inoue, K., Srinivasan, A.: ILP turns 20 - biography and future challenges. Machine Learning **86**(1), 3–23 (2012). https://doi.org/10.1007/s10994-011-5259-2
38. Nienhuys-Cheng, S.H., Wolf, R.d.: Foundations of Inductive Logic Programming. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1997)
39. Plotkin, G.: Automatic Methods of Inductive Inference. Ph.D. thesis, Edinburgh University (August 1971)
40. Raedt, L.D.: Declarative modeling for machine learning and data mining. In: Algorithmic Learning Theory - 23rd International Conference, ALT, Proceedings. p. 12 (2012). https://doi.org/10.1007/978-3-642-34106-9_2
41. Raedt, L.D., Bruynooghe, M.: Interactive concept-learning and constructive induction by analogy. Machine Learning **8**, 107–150 (1992). https://doi.org/10.1007/BF00992861
42. Robinson, J.A.: A machine-oriented logic based on the resolution principle. J. ACM **12**(1), 23–41 (1965). https://doi.org/10.1145/321250.321253
43. Rocktäschel, T., Riedel, S.: End-to-end differentiable proving. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017. pp. 3791–3803 (2017)
44. Sato, T.: Equivalence-preserving first-order unfold/fold transformation systems. Theor. Comput. Sci. **105**(1), 57–84 (1992). https://doi.org/10.1016/0304-3975(92)90287-P
45. Si, X., Lee, W., Zhang, R., Albarghouthi, A., Koutris, P., Naik, M.: Syntax-guided synthesis of datalog programs. In: Leavens, G.T., Garcia, A., Pasareanu, C.S. (eds.) Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018. pp. 515–527. ACM (2018). https://doi.org/10.1145/3236024.3236034
46. Tärnlund, S.: Horn clause computability. BIT **17**(2), 215–226 (1977)
47. Tourret, S., Cropper, A.: SLD-resolution reduction of second-order Horn fragments. Tech. rep., `http://people.mpi-inf.mpg.de/%7Estourret/reports/techrep_jelia19tc.pdf` (2018), a more stable url will be provided in case of acceptance.
48. Wang, W.Y., Mazaitis, K., Cohen, W.W.: Structure learning via parameter learning. In: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management. pp. 1199–1208. ACM (2014)
49. Weidenbach, C., Wischnewski, P.: Subterm contextual rewriting. AI Commun. **23**(2-3), 97–109 (2010). https://doi.org/10.3233/AIC-2010-0459

A Omitted Proofs from Section 5 (Prop. 12)

The proofs in this section assume that in a clause (1) no two literals share more than one variable, and (2) no predicate variable occurs more than once. Condition (1) allows us to identify edges with pairs of vertices without care for their label since there is then at most one edge between two vertices, as is the case in standard unlabeled graphs. The clauses not verifying this condition are less general than the clauses that do. For example, a connected clause C, containing the literals $P_1(x_1, x_2, ..)$ and $P_2(x_1, x_2, ..)$, can be obtained from the connected clause C' equal to C, except that the literals $P_2(x_1, x_2, ..)$ is replaced with $P_2(x_1, x_3, ..)$ where x_3 is a variable not occurring in C, by using the substitution that maps x_3 to x_2. Condition (2) allows us to name vertices with the predicates they represent. Clauses that do not respect this criterion can also be produced by unifying variables. These two constraints stem directly from our working modulo variable unification. The following theorem is a reminder of a classical result in graph theory that is used in the subsequent proof.

Theorem 23 (Th. 2 from [14]). *A finite graph in which the degree of every vertex is at least $d(> 1)$ contains a circuit of length at least $d + 1$.*

Proposition 24. *Let $\mathcal{G}$ be a connected graph containing a circuit of length 3. If $\mathcal{S}$ is a spanning tree of $\mathcal{G}$ containing two edges of the circuit then replacing in $\mathcal{S}$ one of these edges by the non-used one from the circuit yields another spanning tree of $\mathcal{G}$.*

Proof. Let $\mathcal{S}$' be $\mathcal{S}$ after the replacement described in the proposition and v_1, v_2 and v_3 be the vertices in the circuit of length 3. We assume w.l.o.g. that the edges (v_1, v_2) and (v_2, v_3) belong to $\mathcal{S}$ and that the edge (v_2, v_3) is replaced by (v_1, v_3) in $\mathcal{S}$'. Wherever there exists a path between two vertices in $\mathcal{S}$, there also exists a path between them in $\mathcal{S}$':

- if the path doesn't go through (v_2, v_3) in $\mathcal{S}$ then the same path also exists in $\mathcal{S}$',
- if the path goes through (v_2, v_3) in $\mathcal{S}$ then the same path where v_1 is inserted between all contiguous occurrences of v_2 and v_3 exists in $\mathcal{S}$'.

Let us assume the existence of a circuit in $\mathcal{S}$'. Since $\mathcal{S}$ is a spanning tree, the circuit in $\mathcal{S}$' necessarily go through (v_1, v_3). Let us consider a path from v_1 to itself in $\mathcal{S}$'. Then by inserting v_2 in any contiguous occurrences of v_1 and v_3, a path from v_1 to itself in $\mathcal{S}$ is obtained, a contradiction.

Proposition 25 (Spanning Tree). *For any clause $C \in \mathcal{H}^c_{a,\infty}$, $a \in \mathbb{N}^*$, there exists a spanning tree of $\mathcal{G}_C$ in which there exist two adjacent vertices such that the number of edges outgoing from this pair of vertices is at most a.*

Proof. By contradiction, let us assume that no such pair of vertices exists in any spanning tree. Due to Th. 23 there exists at least one vertex v of degree 1 in the spanning tree, because by definition it has no circuit. The vertex v' adjacent to v is thus of degree at least $a + 2$, so that there are at least $a + 1$ outgoing edges from the pair (v, v'). Among the $a + 2$ edges outgoing from v', at least two are labeled with the same term variable(s) as some other edge(s). We want to remove one of these edges and replace it with an edge not connected to v'. These two edges may or may not be labeled with the same variable, thus there are two cases to examine:

1. there is one variable that is the label of at least three distinct edges outgoing from v', thus connecting v' with at least three other distinct vertices, or
2. there are two distinct variables such that each one is the label of two distinct edges outgoing from v', thus connecting two vertices to v', the four such vertices being distinct.

In the first case, since there are at least three distinct vertices connected to v', at least two are distinct from v. We call w and w' these two vertices. Since both w and w' are connected to v', there is no edge between them in the spanning tree, or it would have a circuit. However, note that the edge (w, w') belongs to $\mathcal{G}_C$ because the corresponding literals share a common variable. Let us consider the same spanning tree where the edge (w, v') has been replaced by the edge (w, w'). This new graph is also a spanning tree of $\mathcal{G}_C$ by Prop. 24.

In the second case, among the two pairs of vertices previously identified, we consider the pair of vertices $\{w, w'\}$ such that both are distinct from v. Since the four vertices are distinct, one such pair necessarily exists. As with the first case, since w, w' and v' share a common variable, it is possible to swap the edge (w, v') with (w, w') in the spanning tree and as in the first case, Prop. 24 guarantees that the obtained graph is also a spanning tree of $\mathcal{G}_C$.

In both cases, this operation reduces the number of edges outgoing from the pair v, v' by one. This process can be repeated until this number reaches a, since the conditions to apply the transformation hold as long as the degree of v' is greater than a. This contradicts our initial assumption.

B Omitted Proof from Section 6 (Prop. 18)

On Fig. 2b the neighboring structure of the literals after the non-red preserving transformation shows a symmetry between the ordered pairs of vertices (P_1, P_4) and (P_2, P_5) that mirrors the symmetry between the original vertices P_1 and P_2. Even if this symmetry is partial due to the fact that the rest of the clause is unknown, it can still be exploited to reduce the number of cases to consider in the proof of Prop. 26

Proposition 26 (Non-red Preserving Extension). *If a clause C is irreducible and all the term variables it contains occur three times then any non-red preserving extension of C is also irreducible.*

Proof. Let C be a irreducible clause containing the two literals $P_1(x_1, x_2)$ and $P_2(x_1, x_3)$ (without loss of generality) in which all variables occur exactly three times. Let C_{ext} be the non-red preserving extension of C where the two previously mentioned literals have been replaced by the set of literals $P_1(x_1, x_4)$, $P_2(x_1, x_5)$, $P_3(x_4, x_5)$, $P_4(x_4, x_2)$, $P_5(x_5, x_3)$ where P_3, P_4, P_5, x_4 and x_5 are new predicate and term variables. Assume that C_{ext} is reducible. Then there exist two clauses C_{ext1} and C_{ext2} in $\mathcal{H}^{2c}_{2,\infty}$ both smaller than $C_{ext,}$ such that $C_{ext1}, C_{ext2} \vdash C_{ext}$. If C_{ext1} is made of a subset of the literals in $C_{ext} \backslash C$ (plus a pivot), then C_{ext1} is not 2-connected because all these subsets leave three or more variables pending. The pending variables for each case are described in Tab. 4 (the symmetrical cases are excluded). To illustrate how the table was built, we

Fig. 4: Pending variables when the given literal set is the body of C_{ext1} - the $\star$ symbol indicates a variable pending in C_{ext2}

new literals in C_{ext1}	pending variables
$P_1(x_1, x_4), P_2(x_1, x_5), P_3(x_4, x_5),$ $P_4(x_4, x_2), P_5(x_5, x_3)$	$x_1\star, x_2\star, x_3\star$
$P_1(x_1, x_4), P_2(x_1, x_5),$ $P_3(x_4, x_5), P_4(x_4, x_2)$	$x_1\star, x_2, x_5\star$
$P_1(x_1, x_4), P_2(x_1, x_5),$ $P_4(x_4, x_2), P_5(x_5, x_3)$	$x_1\star, x_2, x_3, x_4\star, x_5\star$
$P_1(x_1, x_4), P_3(x_4, x_5),$ $P_4(x_4, x_2), P_5(x_5, x_3)$	$x_1, x_2, x_3, x_5\star$
$P_1(x_1, x_4), P_2(x_1, x_5), P_3(x_4, x_5)$	$x_1\star, x_4\star, x_5\star$
$P_1(x_1, x_4), P_2(x_1, x_5), P_4(x_4, x_2)$	$x_1\star, x_2, x_4\star, x_5$
$P_1(x_1, x_4), P_3(x_4, x_5), P_4(x_4, x_2)$	x_1, x_2, x_5
$P_1(x_1, x_4), P_3(x_4, x_5), P_5(x_5, x_3)$	$x_1, x_3, x_4\star, x_5\star$
$P_1(x_1, x_4), P_4(x_4, x_2), P_5(x_5, x_3)$	$x_1, x_2, x_3, x_4\star, x_5$
$P_3(x_4, x_5), P_4(x_4, x_2), P_5(x_5, x_3)$	$x_2, x_3, x_4\star, x_5\star$
$P_1(x_1, x_4), P_2(x_1, x_5)$	$x_1\star, x_4, x_5$
$P_1(x_1, x_4), P_3(x_4, x_5)$	$x_1, x_4\star, x_5$
$P_1(x_1, x_4), P_4(x_4, x_2)$	$x_1, x_2, x_4\star$
$P_1(x_1, x_4), P_5(x_5, x_3)$	x_1, x_3, x_4, x_5
$P_3(x_4, x_5), P_4(x_4, x_2)$	$x_2, x_4\star, x_5$
$P_4(x_4, x_2), P_5(x_5, x_3)$	x_2, x_3, x_4, x_5

consider the case where C_{ext1} contains $P_1(x_1, x_4), P_2(x_1, x_5), P_3(x_4, x_5), P_4(x_4, x_2)$, i.e. the second line of Tab. 4. Consider these four literals, the variable x_2 is pending since it occurs exactly once. In addition, since the variables x_1 and x_5 occur only three times in C_{ext}, they are also pending, albeit in C_{ext2}. In the table, variables that are pending for this reason are followed by a star ($\star$). In total, there are three variables pending, which is one too many for the pivot to include all of them as arguments. The cases where C_{ext2} is made only of the literals in $C_{ext}\backslash C$ plus the pivot are symmetrical to the ones in Tab. 4.

The remaining possibilities are when both C_{ext1} and C_{ext2} are made of a mix of the literals in $C_{ext}\backslash C$ and $C_{ext} \cap C$. In these cases, the contradiction appears by going from $C_{ext1}, C_{ext2} \vdash C_{ext}$ to $C_1, C_2 \vdash C$. For example, if $P_1(x_1, x_4)$ and $P_2(x_1, x_5)$ belong to C_{ext1} while the other literals from $C_{ext}\backslash C$ belong to C_{ext2}, then x_4 and x_5 are pending in C_{ext1} (without pivot). There cannot be more than two variables pending in the pivot-less C_{ext1}, C_{ext2} pair or the pivot cannot take all of them as arguments so that they are not pending in C_{ext1} and C_{ext2} (with pivot), thus x_4 and x_5 are the only ones. Now consider C_1 and C_2, obtained respectively from C_{ext1} and C_{ext2} by deleting the five literals of $C_{ext}\backslash C$ from them and adding $P_1(x_1, x_2)$ and $P_2(x_1, x_3)$, i.e. the literals in $C\backslash C_{ext}$ into C_1. Before this transformation, the three occurrences of the variables x_2 and x_3 were located in C_{ext2}. Due to the deletion of literals, only two occurrences of each remain in C_2 and one occurrence of each is now in C_1. Hence both x_2 and x_3 are pending in that case. Except for the variables x_4 and x_5 that are absent from C_1, C_2, the distribution of the remaining variables is unchanged when transforming C_{ext1}, C_{ext2} in

Fig. 5: Transformation from (C_{ext1}, C_{ext2}) to (C_1, C_2) and corresponding evolution of the pending variables

C_{ext1} ; C_{ext2}	pending variables	C_1 ; C_2
$1,2,3,4,5$; $\emptyset$	$\emptyset$; $\emptyset$	$1,2$; $\emptyset$
$1,2,3,4$; 5	x_5 ; x_1	1 ; 2
$1,2,4,5$; 3	x_4, x_5 ; $\emptyset$	$1,2$; $\emptyset$
$1,3,4,5$; 2	x_1, x_5 ; $\emptyset$	$1,2$; $\emptyset$
$1,2$; $3,4,5$	x_4, x_5 ; x_2, x_3	$1,2$; $\emptyset$
$1,3$; $2,4,5$	x_1, x_4, x_5 ; $***$	$***$; $***$
$1,4$; $2,3,5$	x_4 ; $\emptyset$	1 ; 2
$1,5$; $2,3,4$	x_1, x_4, x_5 ; $***$	$***$; $***$
$3,4$; $1,2,5$	x_4, x_5 ; x_2	$\emptyset$; $1,2$
$4,5$; $1,2,3$	x_4, x_5 ; x_2, x_3	$\emptyset$; $1,2$

C_1, C_2, hence these variables are not pending. Thus the pair C_1 C_2 make C reducible, a contradiction.

By taking into account all the symmetries of the problem, there are only ten such cases to consider. They are summarized in Tab. 5. On the left-hand side of the table is the partition between C_{ext1} and C_{ext2} of the five literals in $C_{ext}\backslash C$. On the right-hand side of the table is the partition between C_1 and C_2 of the two literals in $C\backslash C_{ext}$. As was done in the previous example, C_1 and C_2 are obtained by removing the five literals in $C_{ext}\backslash C$ from C_{ext1} and C_{ext2} respectively and replacing them with the two literals in $C\backslash C_{ext}$ as indicated in the table. For readability, the literals are only referred to by their number. In the middle of the table are the variables that are known to be pending in each case (in C_{ext1} and C_{ext2} on the left-hand side and in C_1 and C_2 on the right-hand side). In the cases where there are strictly less than two identified pending variables, there may also be unknown pending variables, but these are preserved by the transformation and thus do not impact the reasoning. In most of the cases, it is possible to have at most two variables pending on the right-hand side of the table, implying that C is reducible, a contradiction. There are also two cases where the assumption that $C_{ext1}, C_{ext2} \in \mathcal{H}^{2c}_{2,\infty}$ is not verified because there are already more than two variables pending in the explicit parts of C_{ext1} and C_{ext2}. In such cases, there is nothing to verify so the right-hand side of the table is filled with asterisks (*).

C Omitted Proof from Section 8 (Th. 22)

We identify with $_R$ the notions where standard resolution replaces SLD-resolution.

Theorem 27 (Reducibility$_R$ of $\mathcal{H}^{2c}_{2,\infty}$). *$\mathcal{H}^{2c}_{2,\infty}$ is reducible$_R$ to $\mathcal{H}^{2c}_{2,2}$*

Proof. Let $C \in \mathcal{H}^{2c}_{2,\infty}$ such that $|\mathsf{b}(C)| \geq 3$.

– If C contains a monadic literal, denoted $P(x)$, then due to the 2-connected constraint, there is at least another occurrence of x in C. Let us denote the other literal in which x occurs as P_x. Its parameters are left implicit since P_x can be

either monadic or dyadic, but x is necessarily one of them. The partition of C into $\{P(x), P_x\}$, $C\backslash\{P(x), P_x\}$ leaves between zero and two variables pending (these pending variables occur in P_x). Thus C is reducible$_R$ and the premises of the corresponding inference are C_1 and C_2, both in $\mathcal{H}^{2c}_{2,\infty}$, such that $\{P(x), P_x\} \subset C_1$, $C\backslash\{P(x), P_x\} \subset C_2$ and the pivot contains the pending variables. If no variable is pending, then x occurs at least twice in both sets due to 2-connectedness, thus the resolving literal can also contain this variable while preserving the 2-connectedness of the two newly formed clauses. This transformation creates one rule of body size two (C_1), and another of body size $|\mathsf{b}(C)| - 1$ (C_2).

- If C contains two dyadic predicates with the same variables (the occurrence of two monadic predicates is covered by the previous case), e.g., $P_1(x_1, x_2)$ and $P_2(x_1, x_2)$, then the partition of C into $\{P_1(x_1, x_2), P_2(x_1, x_2)\}$ and $C\backslash\{P_1(x_1, x_2), P_2(x_1, x_2)\}$ can be used in the same way as in the preceding case to obtain two smaller clauses resolving into C since at most x_1 and x_2 are pending.
- If C contains exactly two occurrences of the same variable in distinct literals or more than three occurrences of the same variable in distinct literals, it is possible to reduce$_R$ C by taking away two of these literals. This does not create more than two pending variables in the resulting partition, as in the two previous cases.

Let us now consider a clause C that cannot be reduced$_R$ following any of the three previous schemes. Such a clause C has the following characteristics:

- C contains only dyadic predicates,
- C does not contain two predicates that have the same pair of variables,
- all variables in C occur exactly three times.

If any of these conditions is not verified, one of the previous schemes can be applied on C to reduce$_R$ it as described. A first notable consequence of these characteristics is that C is such that $|\mathsf{b}(C)| \geq 5$ because C needs at least four variables to not have the same pair of variables occurring twice in different literals of C (all variables occur three times, thus C needs at least twelve variable occurrences grouped in 6 pairs). Of course, these conditions are not sufficient to prevent C from being reducible$_R$ in $\mathcal{H}^{2c}_{2,\infty}$.

To prove this point, let us assume that the clause C is not reducible$_R$ in $\mathcal{H}^{2c}_{2,\infty}$. Let x_1 be a variable occurring in C in the literals $P_1(x_1, x_2)$, $P_2(x_1, x_3)$ and $P_3(x_1, x_4)$ (without loss of generality). The partition of C into $C' = \{P_1(x_1, x_2), P_2(x_1, x_3), P_3(x_1, x_4)\}$ and $C\backslash C'$ leaves three variables pending in C', namely x_2, x_3 and x_4, and none in $C\backslash C'$. As such, it is not suitable because one too many variable is pending. However, it is still possible to use this partition to find resolvents that prove C reducible$_R$ in $\mathcal{H}^{2c}_{2,\infty}$. To do so, let us use again briefly the graph encoding from Def. 9. It is possible to find a path in $\mathcal{G}_{C\backslash C'}$ between two vertices mapped to predicates in each of which a distinct variable among x_2, x_3 and x_4 occurs. This is proven by contradiction. Assume no such path exists, then $\mathcal{G}_{C\backslash C'}$ is made of three disconnected components, corresponding to three clauses C_2, C_3 and C_4, subclauses of C where respectively only x_2, x_3 and x_4 occurs, i.e. in C_2, x_3 and x_4 do not occur but other unrelated variables possibly do, and x_2 certainly occurs, and C_3 and C_4 follow the same pattern. It follows that the partition of C in, e.g., $C_2 \cup \{P_1(x_1, x_2)\}$, $C_3 \cup C_4 \cup \{P_2(x_1, x_3), P_3(x_1, x_4)\}$, with the addition of a pivot on x_1 creates the necessary premises for C to be reducible$_R$, since x_1 is the

only variable pending in the partition, a contradiction. Thus a path containing edges labeled with two of the variables x_2, x_3 and x_4 must exist in $\mathcal{G}_{C \setminus C'}$.

We assume w.l.o.g. that this path links vertices corresponding to predicates that occur applied respectively to x_2 and x_3 without going through an edge labeled with x_4 (otherwise, the shorter path between, e.g., the predicates that occur applied to x_2 and x_4 should be preferred). We denote C_{23} the set of literals that are mapped in $\mathcal{G}_{C \setminus C'}$ to edges in this path. Then the clause $C' \cup C_{23}$ has only one pending variable: x_4. By adding the pivot to the pair of clauses $C' \cup C_{23}$, $C \setminus C'$, premises to derive$_R$ C are created. Note that the situation where the head of C occurs in both sets, preventing the addition of the resolving literal can be solved by selecting x_1 in such a way that the head of C belongs to C'. The size of both sets is smaller than that of C but still greater or equal to 3 since $|C| \geq 6$:

- $|C \setminus C'| = |C| - 3$,
- $|C \cup C_{23}| \leq |C| - 2$ because, without loss of generality, the path including edges labeled with x_2 and x_3 does not go through x_4 (in case it does, the shorter path where x_2 and x_4 occur as edge labels should be used instead), thus two of the three occurrences of x_4 in C do not occur in $C \cup C_{23}$.

Thus the two obtained clauses are smaller than C and we can conclude that C is reducible$_R$.