SLD-Resolution Reduction of Second-Order Horn Fragments - Extended Abstract

Sophie Tourret
Max Planck Institute for Informatics, Saarland Informatics Campus, Germany
sophie.tourret@mpi-inf.mpg.de

Andrew Cropper
University of Oxford, UK
andrew.cropper@cs.ox.ac.uk

We introduce the *derivation reduction* problem, the undecidable problem of finding a finite subset of a set of clauses such that the whole set can be derived using SLD-resolution. We also study the derivation reducibility of various fragments of second-order Horn logic using a graph encoding technique, with particular applications in the field of Inductive Logic Programming.

1 Introduction

Our main focus is studying whether theories formed of second-order function-free Horn clauses can be derivationally reduced to minimal (i.e. irreducible) finite theories, such that all the clauses in the original theory can be derived from the reduced theory. For instance, consider the following theory T, where the symbols P_i represent existentially quantified second-order variables (i.e. variables that can be substituted by any predicate symbols):

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

The clause C_3 is derivationally redundant in T because it can be derived through a self-resolutions of C_2. Since C_2 cannot be derived from C_1 and vice versa, in this case a minimal derivation reduction of T is the theory containing only C_1 and C_2.

Our motivation for studying this problem comes from Inductive Logic Programming [8] (ILP), a subfield of machine learning which induces hypotheses from examples and background knowledge, where hypotheses, examples, and background knowledge are all represented as logic programs. Many forms of ILP [3, 5, 9, 11], or ILP variants [1, 6, 14], rely on SLD-resolution and use second-order Horn clauses as program templates, also denoted as metarules in ILP, to denote the form of hypotheses that may be induced. For instance, the following second-order Horn clause could be used to learn the concept of a transitive closure:

$$P_0(x_1, x_2) \leftarrow P_1(x_1, x_3), P_2(x_3, x_2)$$

However, determining which program templates to use is an open problem in ILP [3,9]. On the one hand, you want to provide enough clauses to ensure completeness (or near-completeness) of the learner [3]. For instance, it is impossible to express transitive closure using the following second-order Horn clause as program template because it only contains monadic literals:

$$P_0(x) \leftarrow P_1(x)$$

On the other hand, you want to remove redundant clauses to improve the efficiency of the learner [3].

Submitted to:
TERMGRAPH 2018

2 Problem Statement and Decidability Results

We now introduce the derivation reduction problem, which is the problem of removing derivationally redundant clauses from a clausal theory. The problem can be defined for any first-order proof system, but we focus on SLD-resolution [7] for the reason previously stated.

We assume two infinite enumerable sets of variables: *term variables* $x_1, x_2, ..$; and *predicate variables* $P, P_0, P_1, ...$ An *atom* is of the form $P_i(x_{k_1}, .., x_{k_a})$, where $i \in \mathbb{N}$ and $a \in \mathbb{N}^*$. The number of term variables in an atom, denoted a, corresponds to the *arity* of the predicate P_i. A *literal* is either an atom (*positive* literal) or the negation of an atom (*negative* literal). A *Horn clause*, denoted as C, D, C', C_1, .. is a finite set of literals interpreted as a disjunction with at most one positive literal. The term "Horn" will be omitted in the rest of the paper, where all clauses are in fact Horn clauses (or λ-free function-free second-order Horn clauses to be precise). We use standard set operations such as union on clauses. The term and predicate variables are respectively assumed to be universally and existentially quantified[1]. A *theory* T is a set of clauses. We denote the set of all clauses, as defined previously, as $\mathscr{H}$. 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 $|b(C)|$, i.e. the body size.

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. SLD-resolution [7] is a restricted form of resolution [12] based on linear resolution, with two 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). Given two clauses C and D such that $h(C)$ can be unified with a literal in $b(D)$ by an m.g.u. σ of these two literals, and given that σ does not unify any other two literals from C and D, the resolvent of C and D by SLD-resolution is $C' = (h(D) \leftarrow b(C) \vee D')\sigma$ where D' is $b(D)$ without the *pivot*, i.e. the literal being resolved upon (i.e. $b(C)$ in C and the literal that unifies with it in D). We write $C, D \vdash C'$ to indicate that C' is the resolvent of C and D by SLD-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 $S^*(T) = \bigcup_{n \in \mathbb{N}} R^n(T)$. We say that 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 $|b(C)|$. A clause C is *redundant* in the theory $T \cup \{C\}$ if and only if $T \vdash^* C$. By extension, we write that 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 now state the *reduction problem*:

Definition 2.1 (Reduction problem) Given a 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 *minimal core* of T.

[1]The quantification of predicate variables does not matter in this paper because we are not concerned with the truth-value of the clauses. In practice, ILP approaches typically use existentially quantified predicate variables [3], however this has no impact on the results presented here.

Theorem 2.2 (Undecidability) *The derivation reduction problem is undecidable.*

Proof sketch. To prove this theorem, we reduce the problem of testing entailment between first-order Horn clauses to it, using the subsumption theorem for SLD-resolution [10]. Since this second problem is known to be undecidable [13], derivation reduction must also be undecidable.

These results show that it is impossible to compute a minimal core for arbitrary theories. Therefore, we instead focus on specific second-order theories that are used by ILP systems. Following is a description of some of these theories.

3 Fragments of Interest in $\mathscr{H}$

From Section 4 onwards we study whether derivationally reduced theories exist for various *fragments* of $\mathscr{H}$, where a fragment of a theory is a syntactically-restricted subset of that theory [2]. In this section, we describe the fragments of interest and introduce additional fragment-related notion.

The first fragments are based on simple syntactic restrictions. The most simple syntactic restrictions that can be imposed on clauses in $\mathscr{H}$ are on the arity of the predicates and on the number of literals in the clauses. Let us consider a fragment $\mathscr{F}$ of $\mathscr{H}$. We write $\mathscr{F}_{a,b}$ to denote clauses in $\mathscr{F}$ that contain predicates of arity at most a and clauses of body size at most b. When one of these restrictions is not imposed, the symbol ∞ replaces the corresponding number. For example, the clause $P_0(x_1) \leftarrow P_1(x_2,x_3,x_4)$ belongs to $\mathscr{H}_{3,1}$. When restrictions are imposed on a fragment that is already restricted, the stricter restrictions are kept. For example, $\mathscr{H}_{3,1} = (\mathscr{H}_{4,1})_{3,\infty} = \mathscr{H}_{4,1} \cap \mathscr{H}_{3,\infty}$ because the stricter restriction on arity is 3, not 4, and the restriction to clauses of body size 1 is stricter than no body size restriction (∞). We rely on the body size restriction to bound the minimal cores of the studied fragments.

Definition 3.1 (Reducible fragment) A fragment $\mathscr{F}$ of $\mathscr{H}$ is reducible to $\mathscr{F}_{\infty,b}$ when, for all $C \in \mathscr{F}$ such that $|b(C)| = b' > b$, there exists $b'' < b'$ such that $\mathscr{F}_{\infty,b''} \vdash C$, i.e. C is the resolvent of an inference with premises in $\mathscr{F}_{\infty,b''}$.

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

Proposition 3.2 (Reduciblility) *If a fragment $\mathscr{F}$ is reducible to $\mathscr{F}_{\infty,b}$ then $\mathscr{F}$ is redundant to $\mathscr{F}_{\infty,b}$.*

Theorem 3.3 (Cores of reducible fragments) *If a fragment $\mathscr{F}$ is reducible to $\mathscr{F}_{\infty,b}$ then the solutions of the reduction problem for $\mathscr{F}$ and $\mathscr{F}_{\infty,b}$ coincide, i.e. if there are any, the minimal cores of $\mathscr{F}$ and $\mathscr{F}_{\infty,b}$ coincide.*

Since our work is largely motivated by applications in ILP [3, 6, 10] we focus on connected clauses:

Definition 3.4 (Connected fragment) A clause is connected if the literals in the clause cannot be partitioned into two 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 $\mathscr{H}^c$, is the subset of $\mathscr{H}$ where all clauses are connected.

Example 3.5 The clause $C_1 = P_0(x_1,x_2) \leftarrow P_1(x_3,x_1), P_2(x_2), P_3(x_3)$ belongs to $\mathscr{H}^c$, but the clause $C_2 = P_0(x_1,x_2) \leftarrow P_1(x_3,x_4), P_2(x_2), P_3(x_3)$ does not because none of the variables occurring under P_0 and P_2, i.e. x_1 and x_2, occur under P_1 and P_3 and vice versa.

A stricter version of connectedness, denoted here as 2-connectedness, is also of importance in ILP [3]. It essentially eliminates singleton occurrences of variables:

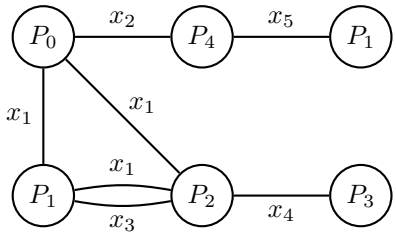


Figure 1: Encoding of $C = P_0(x_1,x_2) \leftarrow P_1(x_1,x_3), P_2(x_1,x_3,x_4), P_3(x_4), P_4(x_2,x_5), P_1(x_5,x_6)$ in the graph $\mathscr{G}_C$, where vertices correspond to literals and edges represent variables shared by two literals.

Definition 3.6 (2-connected fragment) The *2-connected* fragment, denoted as $\mathscr{H}^{2c}$, is the subset of $\mathscr{H}^{c}$ such that all the term variables occur at least twice in distinct literals. A term variable that does not follow this restriction is denoted as *pending*.

Example 3.7 The clause C_1 from Example 3.5 belongs to $\mathscr{H}^{2c}$ because x_1 occurs under P_0 and P_1, x_2 occurs under P_0 and P_2, and x_3 occurs under P_1 and P_3. In contrast, the clause $C_3 = P_0(x_1,x_2) \leftarrow P_1(x_3,x_1), P_2(x_1), P_3(x_3)$ belongs to $\mathscr{H}^{c}$ but not to $\mathscr{H}^{2c}$ because x_2 occurs only once and is thus pending.

Note that the simple restrictions can be combined with both connectedness and 2-connectedness. In the following sections we consider the reduction problem for $\mathscr{H}^{c}$ (Sect. 4) and for $\mathscr{H}^{2c}_{2,\infty}$ (Sect. 5).

4 The Fragment $\mathscr{H}^{c}$ is Reducible to $\mathscr{H}^{c}_{\infty,2}$

We now study whether certain fragments can be reduced. Our first focus is on the fragment $\mathscr{H}^{c}$ containing al connected clauses. We are primarily interested in seeing whether this fragment can be reduced using SLD-resolution to a fragment with only two literals in the body ($\mathscr{H}^{c}_{\infty,2}$). Our general approach is to use concepts from graph theory and we assume reader familiarity with basic notions of this field, in particular, notions of *circuits* and *length* of circuits, *spanning trees*, *connected graphs*, *degree* of vertices and *outgoing* edges (from a set of vertices). To prove the reducibility of $\mathscr{H}^{c}$, we consider $\mathscr{H}^{c}_{a,\infty}$ for any $a \in \mathbb{N}^*$, and show that it can be reduced to $\mathscr{H}^{c}_{a,2}$. To reduce all clauses in $\mathscr{H}^{c}_{a,\infty}$ of body size greater than two, we rely on the following graph encoding to create fitting premises to infer C.

Definition 4.1 (Graph encoding) Let C be a clause in $\mathscr{H}^{c}_{m,\infty}$. The undirected graph $\mathscr{G}_C$ is such that:

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

Example 4.2 The clause $C = P_0(x_1,x_2) \leftarrow P_1(x_1,x_3), P_2(x_1,x_3,x_4), P_3(x_4), P_4(x_2,x_5), P_1(x_5,x_6)$ is mapped to $\mathscr{G}_C$ as illustrated in Fig. 1. In C, there are two occurrences of P_1, thus there are also two vertices with this label in $\mathscr{G}_C$. Moreover the variables x_1 and x_3 occur both under P_1 and P_2 in C, thus there are two edges linking the corresponding nodes in $\mathscr{G}_C$. Note also that since the variable x_6 occurs only in P_4, it is not present in $\mathscr{G}_C$. In fact $\mathscr{G}_C$ also represents many other clauses, e.g., $P_1(x_1,x_3) \leftarrow P_0(x_2,x_1)$, $P_2(x_4,x_3,x_1), P_3(x_4), P_4(x_2,x_5), P_1(x_5,x_5)$.

This graph encoding allows us to abstract the characteristics of the clauses that are irrelevant to our problem and concentrate only on the one notion that matters in this case, namely connectivity, as shown in the following proposition.

Proposition 4.3 *Let $C \in \mathscr{H}$. The graph $\mathscr{G}_C$ is connected if and only if $C \in \mathscr{H}^c$.*

In other words, the notion of connectedness that we introduced for clauses in Def. 3.4 is equivalent to graph connectedness when encoding the clauses in the graph form from Def. 4.1. Since we are only interested in connected clauses, this means that we only handle connected graphs.

In the proofs of this section we assume that (1) in a clause no two literals share more than one variable and (2) no predicate variable occurs more than once. This is done w.l.o.g. to simplify the notations and the reasoning. 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. Condition (2) allows us to name vertices with the predicates they represent. Clauses that do not respect these criteria are trivially derivable from clauses respecting these conditions with clauses of body size 1. For example $C = P_0(x_1,x_2) \leftarrow P_1(x_1,x_2),P_0(x_2,x_3)$ is derivable from $P_0(x_1,x_2) \leftarrow P_1(x_1,x_4),P_2(x_4,x_3)$ and $P_2(x_2,x_3) \leftarrow P_0(x_2,x_3)$, both in $\mathscr{H}^c_{2,\infty}$, by unifying x_2 and x_4.

The following result (Prop. 4.6) is the main intermediary step in the proof of reducibility of the connected fragment (Th. 4.7). We start with a reminder of a classical result in graph theory used in the subsequent proof.

Theorem 4.4 (Th. 2 from [4]) *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 4.5 *Let $\mathscr{G}$ be a connected graph containing a circuit of length 3. If $\mathscr{S}$ is a spanning tree of $\mathscr{G}$ containing two edges of the circuit then replacing in $\mathscr{S}$ one of these edges by the non-used one from the circuit yields another spanning tree of $\mathscr{G}$.*

Proof. Let $\mathscr{S}$' be $\mathscr{S}$ 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 $\mathscr{S}$ and that the edge (v_2,v_3) is replaced by (v_1,v_3) in $\mathscr{S}$'. Wherever there exists a path between two vertices in $\mathscr{S}$, there also exists a path between them in $\mathscr{S}$':

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

Let us assume the existence of a circuit in $\mathscr{S}$'. Since $\mathscr{S}$ is a spanning tree, the circuit in $\mathscr{S}$' necessarily goes through (v_1,v_3). Let us consider a path from v_1 to itself in $\mathscr{S}$'. Then by inserting $v2$ in any contiguous occurrences of v_1 and v_3, a path from v_1 to itself in $\mathscr{S}$ is obtained, a contradiction. ∎

Proposition 4.6 (Spanning tree) *For any clause $C \in \mathscr{H}^c_{a,\infty}$, $a \in \mathbb{N}^*$, there exists a spanning tree of $\mathscr{G}_C$ 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. 4.4 there exists at least one vertex v of degree 1 in the spanning tree, because by definition it has no circuit. The vertice 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'). Since there are at most a arguments in all predicates, there are at least two edges e_1, e_2 related to one or two variables that is/are also behind the existence of some other edge or edges. We want to remove one of these edges and replace it with an edge not connected to v'. There are two possibilities to examine:

1. there is one variable that connects v' with at least three other distinct vertices,
2. there are two distinct variables such that each one connects 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 $\mathscr{G}_C$ because the corresponding predicates 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 $\mathscr{G}_C$ by Prop. 4.5.

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. 4.5 guarantees that the obtained graph is also a spanning tree of $\mathscr{G}_C$.

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. ∎

The main result of this section is the following theorem, that concerns any connected fragment with constrained arity. As implied by Th. 20 it turns out that any such fragment admits a minimal core containing clauses of body size at most two.

Theorem 4.7 (Reducibility of connected fragments) *For any $a \in \mathbb{N}^*$, $\mathscr{H}^c_{a,\infty}$ is reducible to $\mathscr{H}^c_{a,2}$.*

Proof. Let $a \in \mathbb{N}^*$ be fixed and $C = P_0(..) \leftarrow P_1(..),..,P_k(..) \in \mathscr{H}^c_{a,\infty}$ ($k \geq 3$). By applying Prop. 4.6, it is possible to identify two adjacent vertices v and v' in $\mathscr{G}_C$ such that there exists a spanning tree $\mathscr{S}$ of $\mathscr{G}_C$ 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 predicates respectively associated with 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: $C_1 = P_0(..) \leftarrow P'_0(x_1,..,x_{a'}),P_1(..),..,P_k(..)\setminus\{P_v(..),P_{v'}(..)\}$ and $C_2 = P'_0(x_1,..,x_{a'}) \leftarrow P_v(..),P_{v'}(..)$. These clauses are such that $C_1,C_2 \vdash_S C$ and $C_1,C_2 \in \mathscr{H}^c_{a',\infty}$. Thus, C is reducible. ∎

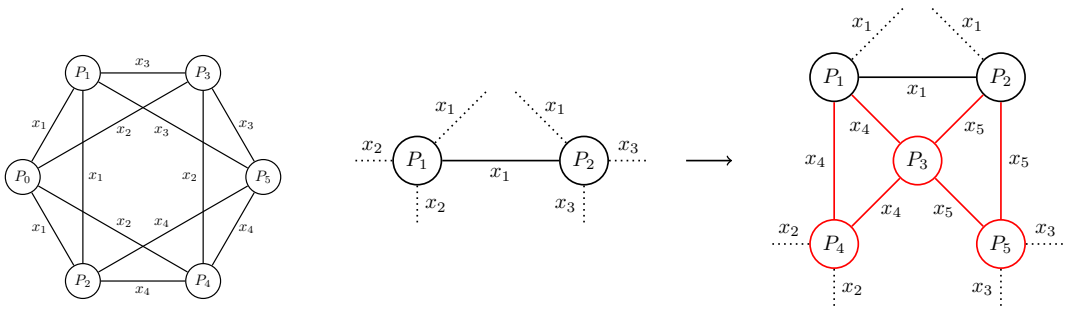
This result is straightforwardly extended to the whole connected fragment.

Lemma 4.8 (Reducibility of the whole connected fragment) *The fragment $\mathscr{H}^c$ is reducible to $\mathscr{H}^c_{\infty,2}$*

Note that this last result does not imply that $\mathscr{H}^c$ has minimal cores. In fact, since it is not possible to increase the arity of predicates through SLD-resolution, any fragment where this arity is not constrained is guaranteed to have no minimal core since at least one predicate of each arity must occur in it and the number of literals that occur in a clause is finite.

5 Reducibility of $\mathscr{H}^{2c}_{2,\infty}$

We now consider the Reduction for the $\mathscr{H}^{2c}_{2,\infty}$ fragment of $\mathscr{H}$. Although this fragment is only slightly more constrained than $\mathscr{H}^c_{2,\infty}$, itself reducible to $\mathscr{H}^c_{2,2}$, the results in this section are completely different from the ones from the previous section. We show that it is not possible to reduce $\mathscr{H}^{2c}_{2,\infty}$ to any size-constrained sub-fragment. To do so we exhibit a set $\mathscr{H}^{nr}$ of clauses in $\mathscr{H}^{2c}_{2,\infty}$ that cannot be reduced. This



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

(b) Graph of a clause before and after a non-red preserving transformation

Figure 2: Graph encoding of $\mathscr{H}^{nr}$ base and construction rule

set contains clauses that are of arbitrary size. In practice, this means that in $\mathscr{H}^{2c}_{2,\infty}$ 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 $\mathscr{H}^{2c}_{2,\infty}$ itself to be reducible to $\mathscr{H}^{2c}_{2,k}$ no matter how big k is.

We start by defining the clause $C_{base} \in \mathscr{H}^{nr}$ that is the foundation on which $\mathscr{H}^{nr}$ is build.

Definition 5.1 (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 the others, i.e. they all have two direct neighbors (literals that share a common variable) from their first variable and two other from their second variable, and there is one literal that does not share any variable with them but is a neighbor to all the other literals. This is better seen on the graphical representation of C_{base} in Fig. 2a, e.g. P_0 does not share literals with P_5 but does with all other predicates.

Proposition 5.2 (Non-reducibility of C_{base}) *C_{base} is not reducible.*

Proof. To derive the clause 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 $\mathscr{G}_{C_{base}}$ in two sets and count the number of edges with distinct labels that link vertices from the two sets. These 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 $\mathscr{H}^{2c}_{2,\infty}$. 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. The symmetries in $\mathscr{G}_{C_{base}}$ are exploited to reduce the number of cases to consider to only four that vary along two axes: the cardinalities of the subsets, either 2-4 or 3-3; and the connectedness of the subsets. Only the following cases are possible. In the 2-4 case, one of the subset is fully connected and the other is not. In the 3-3 case, the subsets are either both fully connected or both not fully connected. 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 not reducible. 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. 5.3) in such a way that when this extension is done on a non-reducible clause, the resulting clause is also not reducible (Prop.5.4).

Definition 5.3 (Non-red preserving extension) Let the body of a clause $C \in \mathscr{H}^{2c}_{2,\infty}$ contain two dyadic predicates 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 replacing 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 fresh predicate and term variable symbols.

By observing on Fig. 2b the neighboring structure of the literals after the non-red preserving transformation, one can see a symmetry between the ordered pairs of vertices (P_1,P_4) and (P_2,P_5) that resembles the symmetry between the original vertices P_1 and P_2.

Proposition 5.4 (Non-red preserving extension) *If a clause C is non-reducible and all the term variables it contains occur three times then any non-red preserving extension of C is also non-reducible.*

Proof sketch. We assume the existence of a reducible non-red extension C_{ext} of a non-reducible clause C in which all term variables occur three times. We use the existence of premises C_{ext1}, C_{ext2} that derive C_{ext} to build C_1, C_2 such that $C_1, C_2 \vdash C$, a contradiction to the fact that C is non-reducible. The number of cases to consider is greatly reduced due to the symmetries of the non-red preserving extension.

Starting from C_{base} and using this extension, we define $\mathscr{H}^{nr}$ formally (Def. 5.5) and as a consequence of the previous proposition, the whole $\mathscr{H}^{nr}$ fragment contains only non-reducible clauses (Prop. 5.6).

Definition 5.5 (Non-reducible fragment) The fragment $\mathscr{H}^{nr}$ is a subset of $\mathscr{H}^{2c}_{2,\infty}$ that contains C_{base} and all the clauses that can be obtained by applying a non-red extension to another clause in $\mathscr{H}^{nr}$.

Proposition 5.6 *For all $C \in \mathscr{H}^{nr}$, C is non-reducible.*

The non-reducibility of $\mathscr{H}^{nr}$ ensures that the body sizes of the clauses in an hypothetical minimal core of $\mathscr{H}^{2c}_{2,\infty}$ cannot be bounded, which in turn prevents the existence of this minimal core. This has direct (negative) consequences on ILP systems that rely on such results to ensure search completeness [9].

6 Conclusion

In this paper, we have introduced the derivation reduction problem for second-order clauses ($\mathscr{H}$), i.e. the problem of computing a finite subset of a set of second-order clauses from which the whole set can be derived using SLD-resolution. We have shown that the problem is undecidable. We have also considered the derivation reducibility of several fragments of $\mathscr{H}$. The first studied fragment, that of connected clauses $\mathscr{H}^c$, was proven reducible to $\mathscr{H}^c_{\infty,2}$, where clauses have at most two literals in their body. The second studied fragment, that of 2-connected clauses made of at most dyadic predicates $\mathscr{H}^{2c}_{2,\infty}$, was proven to have no reduction constraining the body size of clauses.

Concrete minimal cores for the reducible fragments $\mathscr{H}^c_{a,\infty}$ with $a \in \mathbb{N}$ could now be computed to be used as ILP templates. An open challenge for ILP is to work around the negative result for $\mathscr{H}^{2c}_{2,\infty}$, which could be done, for instance, by relaxing SLD-resolution to standard resolution or by allowing the use of triadic literals in restricted cases. The ramifications of these decisions, both theoretical and practical, need further investigation. Considering fragments of $\mathscr{H}^{2c}$ where predicates with a higher arity are allowed, e.g. $\mathscr{H}^{2c}_{3,\infty}$, $\mathscr{H}^{2c}_{4,\infty}$, etc. also remains as future work.

References

[1] Aws Albarghouthi, Paraschos Koutris, Mayur Naik & Calvin Smith (2017): *Constraint-Based Synthesis of Datalog Programs*. In: *International Conference on Principles and Practice of Constraint Programming*, Springer, pp. 689–706.

[2] Aaron R Bradley & Zohar Manna (2007): *The calculus of computation: decision procedures with applications to verification*. Springer Science & Business Media.

[3] A. Cropper & S.H. Muggleton (2015): *Logical minimisation of meta-rules within Meta-Interpretive Learning*. In: *Proceedings of the 24th International Conference on Inductive Logic Programming*, Springer-Verlag, pp. 65–78. Available at `http://www.doc.ic.ac.uk/~shm/Papers/minmeta.pdf`. LNAI 9046.

[4] G. A. Dirac (1952): *Some Theorems on Abstract Graphs*. *Proceedings of the London Mathematical Society* s3-2(1), pp. 69–81, doi:10.1112/plms/s3-2.1.69.

[5] Werner Emde, Christopher U Habel & Claus-Rainer Rollinger (1983): *The discovery of the equator or concept driven learning*. In: *Proceedings of the Eighth international joint conference on Artificial intelligence-Volume 1*, Morgan Kaufmann Publishers Inc., pp. 455–458.

[6] Richard Evans & Edward Grefenstette (2017): *Learning Explanatory Rules from Noisy Data*. *arXiv preprint arXiv:1711.04574*. To appear in JAIR.

[7] Robert A. Kowalski (1974): *Predicate Logic as Programming Language*. In: *IFIP Congress*, pp. 569–574.

[8] S.H. Muggleton (1991): *Inductive Logic Programming*. *New Generation Computing* 8(4), pp. 295–318. Available at `http://www.doc.ic.ac.uk/~shm/Papers/ilp.pdf`.

[9] S.H. Muggleton, D. Lin & A. Tamaddoni-Nezhad (2015): *Meta-Interpretive Learning of Higher-Order Dyadic Datalog: Predicate Invention revisited*. *Machine Learning* 100(1), pp. 49–73.

[10] Shan-Hwei Nienhuys-Cheng & Ronald de Wolf (1997): *Foundations of Inductive Logic Programming*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[11] Luc De Raedt & Maurice Bruynooghe (1992): *Interactive Concept-Learning and Constructive Induction by Analogy*. *Machine Learning* 8, pp. 107–150, doi:10.1007/BF00992861.

[12] John Alan Robinson (1965): *A Machine-Oriented Logic Based on the Resolution Principle*. *J. ACM* 12(1), pp. 23–41, doi:10.1145/321250.321253.

[13] Manfred Schmidt-Schauß (1988): *Implication of Clauses is Undecidable*. *Theor. Comput. Sci.* 59, pp. 287–296, doi:10.1016/0304-3975(88)90146-6.

[14] William Yang Wang, Kathryn Mazaitis & William W Cohen (2014): *Structure learning via parameter learning*. In: *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, ACM, pp. 1199–1208.