Composition of Relational Features with an Application to Explaining Black-Box Predictors

Ashwin Srinivasan∗ · A Baskar · Tirtharaj Dash† · Devanshu Shah

Received: date / Accepted: date

Abstract Three key strengths of relational machine learning programs like those developed in Inductive Logic Programming (ILP) are: (1) The use of an expressive subset of first-order logic that allows models that capture complex relationships amongst data instances; (2) The use of domain-specific relations to guide the construction of models; and (3) The models constructed are human-readable, which is often one step closer to being human-understandable. The price for these advantages is that ILP-like methods have not been able to capitalise fully on the rapid hardware, software and algorithmic developments fuelling current developments in deep neural networks. In this paper, we treat relational features as functions and use the notion of generalised composition of functions to derive complex functions from simpler ones. Motivated by the work of McCreath and Sharma (McCreath and Sharma, 1998a; McCreath, 1999) we formulate the notion of a set of M-simple features in a mode language M and identify two composition operators (ρ1 and ρ2) from which all possible complex features can be derived. We use these results to implement a form of “explainable neural network” called Compositional Relational Machines, or CRMs. CRMs are labelled directed-acyclic graphs. The vertex-label for any vertex j in the CRM contains a feature-function fj and a continuous activation function gj. If j is a “non-input” vertex, then fj is the composition of features associated with vertices in the direct predecessors of j. Our focus is on CRMs in which input vertices (those without any direct predecessors) all have M-simple features in their vertex-labels. We provide a randomised procedure for constructing the structure of such CRMs, and a procedure for esti-

A. Srinivasan · T Dash
APPCAIR, BITS Pilani, India

A. Srinivasan · A. Baskar · T Dash · D Shah
Department of CS & IS
BITS Pilani, K.K. Birla Goa Campus
Goa 403726, India
E-mail: {ashwin,abaskar,tirtharaj,f20180240}@goa.bits-pilani.ac.in

∗AS is currently visiting TCS Research. He is also a Visiting Professor at the Centre for Health Informatics, Macquarie University, Sydney; and a Visiting Professorial Fellow at the School of CSE, University of New South Wales, Sydney.

†TD is currently at the University of California, San Diego, CA, USA.
mating the parameters (the $w_{ij}$’s) using back-propagation and stochastic gradient descent. Using a notion of explanations based on the compositional structure of features in a CRM, we provide empirical evidence on synthetic data of the ability to identify appropriate explanations; and demonstrate the use of CRMs as ‘explanation machines’ for black-box models that do not provide explanations for their predictions.

**Keywords** Explainable Neural Networks, Relational Features, Inductive Logic Programming, Neuro-Symbolic Learning

1 Introduction

It has long been understood that choice of representation can make a significant difference to the efficacy of machine-based induction. A seminal paper by Quinlan (1979) demonstrates the increasing complexity of models constructed for a series of problems defined on a chess endgame, using a fixed representation consisting of 25 features (called properties in the paper). These features were identified manually (by him), and captured relations between the pieces and their locations in the endgame. Commenting on the increasing complexity of the models, he concludes:

“This immediately raises the question of whether these and other properties used in the study were appropriate. The answer is that they were probably not; it seems likely that a chess expert could develop more pertinent attributes ... If the expert does his job well the induction problem is simplified; if a non-expert undertakes the definition of properties (as was the case here) the converse is true.”

Although Quinlan assumed the representation would be identified manually, the possibility of automatic identification of an appropriate representation was already apparent to Amarel (1968) a decade earlier: “An understanding of the relationship between problem formulation and problem solving efficiency is a prerequisite for the design of procedures that can automatically choose the most ‘appropriate’ representation of a problem (they can find a ‘point of view’ of the problem that maximally simplifies the process of finding a solution)”. In fact, by ‘choose' what is probably meant is ‘construct’, if we are to avoid kicking Feigenbaum’s famous knowledge-acquisition bottleneck down the road from extracting models to extracting representations.

It has also been known that one way to construct representations automatically is through the use of neural networks. But extraction and re-use of these representations for multiple tasks have become substantially more common only recently, with the routine availability of specialised hardware. This has allowed the construction of networks in which adding layers of computation is assumed to result in increasingly complex representations (10s of layers are now common, but 100s are also within computational range). In principle, while a single additional layer is all that is needed (due to the Universal approximation theorems for neural networks (Hornik et al., 1989; Cybenko, 1989; Pinkus, 1999)), it is thought that the main benefit of the additional layers lies in constructing representations that are multiple levels of abstractions, allowing more efficient modelling. There are however 3 important issues that have surfaced: (1) Automatic construction of abstractions in this way requires a lot of data, often 100s of 1000s of examples;
(2) The kinds of neural networks used for constructing abstractions depend on the type of data. For example, certain kinds of networks are used for text, others for images and so on: there is apparently no single approach for representation learning that can automatically adjust to the data type; and (3) The internal representations of data are opaque to human-readability, making it difficult to achieve the kind of understanding identified by Amarel.

Recent results with neural-based learning suggest the possibility of viewing representation learning as program synthesis. Of particular interest are methods like Dreamcoder (Ellis et al., 2021) that automatically construct programs for generating data, given some manually identified primitive functions represented in a symbolic form (Dreamcoder uses $\lambda$-expressions for the primitive functions). A combination of generative and discriminative network models is used to direct the hierarchical construction of higher-level functions by compositions of lower-level ones. Compositions are generated and assessed for utility in interleaved phases of generate-and-test (called “Dream” and “Wake” phases), until the neural-machinery arrives at a small program, consisting the sequential composition of primitive- and invented functions. The final result is some approximation to the Bayesian MAP program for generating the data provided, assuming a prior preference for small programs. There are good reasons to look at this form of program-synthesis as a mechanism for automated representation learning: (a) Empirical results with programs like Dreamcoder show that it is possible to identify programs with small numbers of examples (the need for large numbers of examples is side-stepped by an internal mechanism for generating data in the Dream phase); (b) In principle, the symbolic language adopted for primitive functions ($\lambda$-expressions) and the mechanism of function composition is sufficiently expressive for constructing programs for data of any type; (c) The intermediate representations have clearly defined interpretations, based on functional composition. There are however some shortcomings. First, the primitive functions have to be manually identified. Secondly, the construction of new representations requires a combinatorial discrete search that is usually less efficient than those based on continuous-valued optimisation. Thirdly, the representation of $\lambda$-expressions, although mathematically powerful, can prove daunting as a language for encoding domain-knowledge or for interpreting the results. Finally, the Dreamcoder-like approach for representation learning has only been demonstrated on very simple generative tasks of a geometric nature.

In this paper, we partially address these shortcomings by drawing on, and extending some results on representation developed in the area of Inductive Logic Programming (ILP). The main contributions of this paper are as follows:

(a) **Conceptual** We develop the conceptual basis for a class of ‘simple’ relational features using a well-known specification language in ILP (mode-declarations). Additionally, we develop composition operators for deriving more complex relational features, and prove some completeness properties that apply to the use of the operators;

(b) **Implementation** We use the concepts developed to specify and implement a form of neural network called Compositional Relational Machines, or CRMs. An important feature of these networks is that each node is identified with a clearly defined relational feature. This allows us to associate structured ‘explanations’ with each node in the network;
(c) Application We present empirical results on 2 synthetic data that demonstrate the ability of CRMs to construct appropriate explanations; and results on using CRMs to act as 'explanation machines' for a state-of-the-art black-box predictor on 10 real-world datasets.

The rest of the paper is organised as follows: In Section 2 we provide a conceptual framework for relational features and their compositions. We use this framework to implement CRMs in Section 3. We provide empirical evaluation of CRMs as explanation machines in Section 4. Related work of immediate relevance to this paper are presented in Section 5. Concluding remarks are in in Section 6. The paper has several appendices that act as supporting material.

2 Relational Features and their Composition

In this paper, we are principally interested in specifying and combining relational features. For us, a \( k \)-ary relational feature will be a function with \( k \) terms as arguments. We will specify such a feature by \( f: A_1 \times A_2 \times \cdots \times A_k \rightarrow B \), where \( A_1, A_2, \ldots, A_k, B \) are some sets. For the most part in this paper, we will restrict ourselves to \( k = 1 \) and \( B = \{0, 1\} \), although the results here can be generalised. We will denote this setting as \( f: A \rightarrow \{0, 1\} \), for some set \( A \).

A relational feature is defined in 2 steps. First, we represent the conditions under which the feature takes the value 1 using a clause of the form:

\[
C: \forall X (p(X) \leftarrow \exists Y Body(X, Y))
\]

or, simply:

\[
C: (p(X) \leftarrow Body(X, Y))
\]

to mean the quantification as shown earlier. Here, the fixed predicate symbol \( p(X) \) is called the head of \( C \), and \( Body(X, \cdot) \)–the body of clause \( C \)–is a conjunction of literals \( l_1, l_2, \ldots, l_k \) containing some existentially quantified variables collectively represented here as \( Y \). We assume the body of \( C \) does not contain a literal of the form \( p(\cdot) \) (that is, \( C \) is not self-recursive) and call clauses like these feature-clauses.\(^2\) The clausal representation does not tell us how to obtain the value (0 or 1) of the feature itself for any \( X = a \). For this we assume an additional set of clausal formulae \( B \) (“background”) which does not contain any occurrence of the predicate-symbol \( p/1 \) and define the feature-function associated with the clause \( C \) as follows. Let \( \theta_a \) denote the substitution \( \{X/a\} \) for \( a \in A \). Then:

\[
f_{C,B}(a) = \begin{cases} 1 & \text{if } B \cup (C\theta_a) \models p(a) \\ 0 & \text{otherwise} \end{cases}
\]

\(^1\) See Appendix A for a summary of logical syntax and concepts needed for this paper. We assume a logic with equality (=/2).

\(^2\) This clause is equivalent to the disjunct \( l_1 \lor \neg l_2 \lor \cdots \lor \neg l_k \). It will sometimes also be written as the set of literals \( \{l_1, \neg l_2, \ldots, \neg l_k\} \). The literals can contain predicate symbols representing relations: hence the term “relational”. The requirement that all feature-clauses have the same predicate symbol \( p/1 \) in the head is a convenience that will be helpful in what follows. It may be helpful to read the symbol \( p/1 \) as a proposition defined on \( X \).
In general, given feature-clauses $C_1, C_2, \ldots, C_j$ we will write $f_{C_i, B}(\cdot)$ as $f_i(\cdot)$ ($1 \leq i \leq j$), when the context is obvious. If $f_i(x) = 1$ for $x = a$, we will say “the feature $f_i$ is true for $x = a$”.

**Example 1** An early example of a problem requiring relational features was the problem of discriminating amongst goods trains (Michalski, 1980), which has subsequently served as a touchstone for the construction and use of relational features (see for example, the “East-West Challenge” (Michie et al., 1994)). In its original formulation, the task is to distinguish eastbound trains from westbound ones using properties of the carriages and their loads (the engine’s properties are not used), using pictorial descriptions like these ($T_1$ is eastbound and $T_2$ is westbound):

Examples of feature-clauses are:

- $C_1 : p(X) \leftarrow (\text{has car}(X,Y), \text{short}(Y))$
- $C_2 : p(X) \leftarrow (\text{has car}(X,Y), \text{closed}(Y))$
- $C_3 : p(X) \leftarrow (\text{has car}(X,Y), \text{short}(Y), \text{closed}(Y))$
- $C_4 : p(X) \leftarrow (\text{has car}(X,Y), \text{has car}(X,Z), \text{short}(Y), \text{closed}(Z))$

Here, we will assume that predicates like $\text{has car}/2$, $\text{short}/1$, $\text{closed}/1$, $\text{long}/1$, $\text{has load}/3$, etc., are defined as part of the background $B$, and capture the situation shown diagrammatically. That is,

$$B = \{ \text{has car}(t_1, c_1, 1), \text{has car}(t_1, c_1, 2), \ldots, \text{long}(c_1, 1), \text{closed}(c_1, 1), \text{has load}(c_1, \text{square}, 3), \ldots, \text{has car}(t_2, c_2, 1), \text{has car}(t_2, c_2, 2), \ldots \}.$$

Then the corresponding feature-function values are:

$$f_{C_1, B}(t_1) = f_1(t_1) = 1; \quad f_2(t_2) = 1;$$
$$f_{C_2, B}(t_1) = f_2(t_1) = 1; \quad f_2(t_2) = 1;$$
$$f_{C_3, B}(t_1) = f_3(t_1) = 1; \quad f_3(t_2) = 0;$$
$$f_{C_4, B}(t_1) = f_4(t_1) = 1; \quad f_4(t_2) = 1.$$

Although not used in this paper, feature-clauses need not be restricted to descriptions of single objects. An example of a feature-clause about train-pairs for example is:

- $C_5 : p((X_1, X_2)) \leftarrow (\text{has length}(X_1, L_1), \text{has length}(X_2, L_2), L_1 \geq L_2)$.

(The corresponding feature-function will then also be defined over pairs of objects.)

We intend to describe a mechanism for automatically enumerating feature-clauses like these, as well as mechanisms for combining simpler feature-clauses to give more complex ones. We start with some preliminary definitions needed.
2.1 Preliminaries

It will be necessary for what follows to assume an ordering over literals in a feature-clause.

**Definition 1 (Ordered Clause)** Let $C$ be a feature-clause with 1 head literal and $k-1$ body literals. We assume an ordering over the literals that maps the set of literals in the clause to a sequence $\langle C \rangle = \langle \lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_k \rangle$, where $\lambda_1$ is the head literal and the $\lambda_2, \ldots, \lambda_k$ are literals in the body of the feature-clause.

**Definition 2 (Ordered Subclause)** Let $\langle C \rangle = \langle \lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_k \rangle$ be an ordered clause. Then an ordered subclause $\langle C' \rangle$ is any clause $\langle \lambda_1', \lambda_2', \ldots, \lambda_j' \rangle$ where $\lambda_1' = \lambda_1$ and $\langle \lambda_2', \ldots, \lambda_j' \rangle$ is a sub-sequence of $\langle \lambda_2, \ldots, \lambda_k \rangle$.

From now on, we will use the term “ordered clause” to emphasise an ordering on the literals is assumed. For simplicity, we will assume that the intended ordering is reflected in a left-to-right reading of the clause. Given an ordered clause, it is possible to recover trivially the set of literals constituting the feature-clause. We use $\text{Set}(\langle C \rangle) = \text{Set}(\langle \lambda_1, \ldots, \lambda_k \rangle) = \{\lambda_1, \neg \lambda_2, \ldots, \neg \lambda_k\}$. Usually, we will further use the set-notation interchangeably with $\lambda_1 \leftarrow \lambda_2, \ldots, \lambda_k$ to denote the feature-clause $C$.

2.2 Feature-Clauses in a Mode Language

The field of Inductive Logic Programming (ILP) has extensively used the idea of a *mode language* to specify a set of acceptable clauses.\(^3\) We use this approach here to specify the set of feature-clauses. We provide details of mode-declarations and the definition of a mode-language based on such declarations in Appendix B. We need the following concepts from the Appendix: (a) type-names and their definitions; (b) set of mode declarations and clauses in the mode-language; (c) input term of type $\gamma$ in some literal; and (d) output term of type $\gamma$ in some literal. With these notions in place, we will require the mode-language for specifying feature-clauses to satisfy the following constraints:

- **MC1.** The set of modes $M$ contains exactly one mode-declaration for every predicate occurring in a feature-clause;
- **MC2.** All modes in $M$ for predicates which appear in the body of a feature-clause contain at least 1 “input” argument; and
- **MC3.** If $\mu = \text{modeh}(p) \in M$, then $p$ is an unary predicate and $\text{modeb}(p)$ does not occur in $M$.

These constraints extend to $p/k$ if the features are defined over a product-space. We note that the restriction MC1 is more strict than the mode language allowed by ILP implementations like Progol (Muggleton, 1995) or Aleph (Srinivasan, 2001). Effectively, it prevents a predicate being called in multiple ways, which is allowed

---

\(^3\) The notion of associating *modes* for predicates has its origins in typed logics and functional programming. Even in ILP, modes are not the only way of specifying acceptable sets of clauses; they are used here as they provide a straightforward way of specifying the notion of *simple features* that follows in a later section.
in logic programming languages like Prolog. Here, to achieve the same effect, we will need to use different predicate symbols for each mode of call. Now, variables (and ground-terms) are constrained by the type-restrictions. Given a set of mode-declarations, feature-clauses in the mode-language are therefore more constrained than we have presented thus far. Henceforth we will use $M$ is a set of “constrained mode-declarations” to mean that $M$ satisfies MC1–MC3.

The categorisation of variables in a literal as being inputs or outputs allows a natural association of an ordered clause with a graph.

**Definition 3 (Clause Dependency-Graph)** Let $M$ be a set of constrained mode-declarations and $T$ be a set of type definitions for the type-names in $M$. Let $\langle C \rangle$ be an ordered clause ($\lambda_1 \leftarrow \lambda_2, \ldots, \lambda_k$) in the mode-language $L_{M,T}$ (see Appendix B). The clause dependency-graph $G_{M}(\langle C \rangle)$ associated with $\langle C \rangle$ is the labelled directed graph $(V,E,\psi)$ defined as follows:

- $V = \{v_1, v_2, \ldots, v_k\}$;
- for each $i$, $\psi(v_i) = \lambda_i$;
- $(v_i, v_j) \in E$ iff:
  - $i = 1, 2 \leq j \leq k$, and there exists a variable $X$ such that $\lambda_i$ has $X$ as an input variable of type $\gamma$ in $M$ and $\lambda_j$ has $X$ as an input variable of type $\gamma$ in $M$; or
  - $1 < i < j$, $\lambda_i$ has an output variable $X$ of type $\gamma$ in $M$ and $X$ occurs in $\lambda_j$ as an input variable of type $\gamma$ in $M$.

**Example 2** Let us assume the set of mode-declarations $M$ contain at least the following: \{modeh$(p(+\text{train}))$, modeb$(\text{has}_\text{car}(+\text{train},-\text{car}))$, modeb$(\text{short}(+\text{car}))$, modeb$(\text{closed}(+\text{car}))$\} where train and car are type-names, with definitions in $T$. The ordered clause $p(X) \leftarrow (\text{has}_\text{car}(X,Y), \text{has}_\text{car}(X,Z), \text{short}(Y), \text{closed}(Z))$ is in the mode-language $L_{M,T}$. The clause dependency-graph for this ordered clause is given below and $\psi$ is defined as follows: $\psi(v_1) = p(X)$, $\psi(v_2) = \text{has}_\text{car}(X,Y)$, $\psi(v_3) = \text{has}_\text{car}(X,Z)$, $\psi(v_4) = \text{short}(Y)$, $\psi(v_5) = \text{closed}(Z)$.

**Remark 1** We note the following about the clause dependency-graph:

- The clause dependency-graph for an ordered clause is a directed acyclic graph. This is evident from the definition: if $(v_i, v_j) \in E$ then $i < j$.
- The clause dependency-graph for an ordered clause is unique.

Given a set of modes $M$ we introduce the notion $M$-simple clauses in a manner similar to (McCreath, 1999).

---

4 That is, a feature-clause of the form $\forall X (p(X) \leftarrow \exists Y \text{Body}(X,Y))$ should be read as $\forall X \in \Lambda \ [p(X) \leftarrow \exists Y \in \Lambda \ \text{Body}(X,Y)]$ where $\Lambda$ and $\Lambda$ informally denote the sorts of $X$ and the $Y$’s. For simplicity, we will not refer to the type-restrictions on variables and terms when we say that a feature-clause is in a mode-language. The restrictions are taken as understood, and to be enforced during inference.
Definition 4 (Source- and Sink- Vertices and Literals) Given a set of constrained mode-declarations $M$, type-definitions $T$, and an ordered clause $(C)$ in $\mathcal{L}_{M,T}$, let $G_M(⟨C⟩) = (V,E,ψ)$ be the clause dependency-graph of $⟨C⟩$. A vertex $v \in V$ is said to be a sink vertex if there is no outgoing edge from $v$. The corresponding literal, $ψ(v)$, is called a sink literal. A vertex $v \in V$ is said to be a source vertex if there is no incoming edge to $v$. The corresponding literal, $ψ(v)$, is called a source literal.

Example 3 The clause in Example 2 has one source vertex $v_1$ and two sink vertices: $v_4$ and $v_5$. Correspondingly, there is one source literal $p(X)$, and two sink literals: $\text{short}(Y), \text{closed}(Y)$.

Remark 2 Let $M$ satisfy MC1–MC3. Then:

- The clause dependency-graph of any ordered clause in $\mathcal{L}_{M,T}$ will have exactly 1 source-vertex $v_1$ (and exactly 1 source-literal).
- For every vertex $v$ in the clause dependency-graph, there exists at least one path from $v_1$ to $v$. Also the union of all the paths from $v_1$ to $v$ will be a directed acyclic graph and this is unique. We will denote this directed acyclic graph by $DAG_{⟨C⟩}(v)$.
- For every vertex $v$ in the clause dependency-graph, either $v$ will be a sink vertex or it will be on a path from the source vertex ($v_1$) to a sink vertex.

Of these the third observation is not obvious. Suppose a vertex $v$ is not a sink vertex. Then it will have at least one outgoing edge from it. By following outgoing edges forward, we will end in a sink vertex. If $v \neq v_1$, then there is at least one incoming edge to $v$. By following incoming edges backward, we will end in a source vertex. Since there is only one source vertex, this will be $v_1$. So $v$ will be a sink vertex or it will be on a path from $v_1$ to a sink vertex.

Definition 5 (M-Simple Feature-Clause) Given a set of constrained mode-declarations $M$, type-definitions $T$, an ordered feature-clause $⟨C⟩$ in the mode-language $\mathcal{L}_{M,T}$ is said to be an M-simple feature-clause, or simply a M-simple clause iff the clause dependency-graph $G_M(⟨C⟩)$ has exactly one sink literal.

Example 4 We continue Example 2. The ordered clauses $p(X) \leftarrow \text{has} \_\text{car}(X,Y)$, $\text{short}(Y)$ and $p(X) \leftarrow \text{has} \_\text{car}(X,Y)$ are M-simple clauses as both have only one sink literal $\text{short}(Y)$ and $\text{has} \_\text{car}(X,Y)$ respectively. The ordered clause $p(X) \leftarrow \text{has} \_\text{car}(X,Y), \text{short}(Y), \text{closed}(Y)$ is not a M-simple clause as it has two sink literals $\text{short}(Y)$ and $\text{closed}(Y)$.

Definition 6 (Maximal M-simple subclause) Given a set of constrained mode-declarations $M$, type-definitions $T$, an ordered clause $⟨C⟩$ in the mode-language $\mathcal{L}_{M,T}$ is said to be a maximal M-simple subclause of an ordered clause $(C)$ in $\mathcal{L}_{M,T}$ iff: (a) $⟨C‘⟩$ is an ordered subclause of $⟨C⟩$; and (b) there is an isomorphism from the clause dependency-graph $G_M(⟨C⟩)$ to $DAG_{⟨C⟩}(v)$ for some sink vertex $v$ in $G_M(⟨C⟩)$.

Example 5 Continuing Example 2, the ordered clause $p(X) \leftarrow \text{has} \_\text{car}(X,Y), \text{short}(Y)$ is a maximal M-simple subclause of $p(X) \leftarrow \text{has} \_\text{car}(X,Y), \text{short}(Y), \text{closed}(Y)$. The ordered clause $p(X) \leftarrow \text{has} \_\text{car}(X,Y), \text{short}(Y), \text{closed}(Y)$ is not a maximal M-simple subclause of $p(X) \leftarrow \text{has} \_\text{car}(X,Y), \text{short}(Y), \text{closed}(Y)$. 

A. Srinivasan et al.
Example 6

Composition of Relational Features

Note that if the ordered clause provided by axioms of the equality logic. For more details see Appendix A. We

\[ \langle \text{some clause} \rangle \]

Let \( G \) be a set of mode-declarations, \( M \) be a set of type-declarations, and \( T \) be the set of type-definitions, and \( M' \) be an extension of \( M \) as above. Let \( T \) be a set of type-declarations, and \( L \in T \) using \( \{p_1, p_2\} \) and clauses in \( L.M.T \).

These operators allow us to establish a link between the derivability of clauses in \( L.M.T \).

1. Let \( \langle C \rangle \) be in \( L.M.T \). Then \( C \) has \( X \), \( Y \) are output variables of the same type in \( \langle C \rangle \) and \( X \subseteq Y \). We define operators \( p_1 \) as follows:

\[ p_1(C) = \langle p_1(X) \rangle = \langle p_1(Y) \rangle \]

2. Let \( \langle C \rangle \) be in \( L.M.T \). Then \( p_2(C) = \langle p_2(X) \rangle = \langle p_2(Y) \rangle \).

Next we consider \( C \subseteq C' \). Let \( M' \) be a set of constrained mode-declarations and \( T \) be a set of type-declarations. \( T \) be a set of type-definitions, and \( M' \) be an extension of \( M \) as above. Let \( T \) be a set of type-declarations, and \( L \in T \) using \( \{p_1, p_2\} \) and clauses in \( L.M.T \).

These operators allow us to establish a link between the derivability of clauses in \( L.M.T \).

We define operators \( p_1 \) as follows:

\[ p_1(C) = \langle p_1(X) \rangle = \langle p_1(Y) \rangle \]

3. Let \( \langle C \rangle \) be in \( L.M.T \). Then \( C \) has \( X \), \( Y \) are output variables of the same type in \( \langle C \rangle \) and \( X \subseteq Y \). We define operators \( p_2 \) as follows:

\[ p_2(C) = \langle p_2(X) \rangle = \langle p_2(Y) \rangle \]

Lemma 1 (Basis Lemma) Let \( M \) be a set of constrained mode-declarations, \( T \) be a set of type-declarations, and \( M' \) be an extension of \( M \) as above. Let \( T \) be a set of type-declarations, and \( L \in T \) using \( \{p_1, p_2\} \) and clauses in \( L.M.T \).

Lemma 2 (Basis Lemma) Let \( M \) be a set of constrained mode-declarations, \( T \) be a set of type-declarations, and \( M' \) be an extension of \( M \) as above. Let \( T \) be a set of type-declarations, and \( L \in T \) using \( \{p_1, p_2\} \) and clauses in \( L.M.T \).

We prove \( \bar{U} \subseteq \bar{S} \) and \( \bar{S} \subseteq \bar{U} \) as follows:

\[ \bar{U} \subseteq \bar{S} \]

Next we consider \( C \subseteq C' \). Let \( M' \) be a set of constrained mode-declarations and \( T \) be a set of type-declarations. \( T \) be a set of type-definitions, and \( M' \) be an extension of \( M \) as above. Let \( T \) be a set of type-declarations, and \( L \in T \) using \( \{p_1, p_2\} \) and clauses in \( L.M.T \).

These operators allow us to establish a link between the derivability of clauses in \( L.M.T \).

We define operators \( p_1 \) as follows:

\[ p_1(C) = \langle p_1(X) \rangle = \langle p_1(Y) \rangle \]

3. Let \( \langle C \rangle \) be in \( L.M.T \). Then \( C \) has \( X \), \( Y \) are output variables of the same type in \( \langle C \rangle \) and \( X \subseteq Y \). We define operators \( p_2 \) as follows:

\[ p_2(C) = \langle p_2(X) \rangle = \langle p_2(Y) \rangle \]
\( \Omega \subseteq \{ \rho_1, \rho_2 \} \). Let \( \Phi \) be a set of feature-clauses in \( \mathcal{L}_{M', T} \). A sequence of feature-clauses \( \langle C_1, C_2, \ldots, C_n \rangle \) is said to be a derivation sequence of \( \langle C_n \rangle \) from \( \Phi \) using \( \Omega \) iff each clause \( \langle C_i \rangle \) in the sequence is either: (a) an instance of an element of \( \Phi \) such that no variables other than \( X \) occur earlier in this sequence; or (b) an element of the set \( \rho_1(\langle C_j \rangle) \) for some \( j < i \), if \( \rho_1 \in \Omega \); or (c) an element of the set \( \rho_2(\langle C_j \rangle, \langle C_k \rangle) \) for some \( j, k < i \), if \( \rho_2 \in \Omega \). We will say \( \langle C_n \rangle \) is derivable from \( \Phi \) using \( \Omega \) if there exists a derivation sequence of \( \langle C_n \rangle \) from \( \Phi \) using \( \Omega \).

**Example 7** Let us assume the set of mode-declarations \( M \) contain the following:

\[
\{ \text{mode}(p(+\text{train})), \text{modeb}(\text{has\_car}(+\text{train}, -\text{car})), \text{modeb}(\text{short}(+\text{car})), \\
\text{modeb}(\text{closed}(+\text{car})), \text{modeb}(\text{smaller}(+\text{car}, +\text{car})) \}
\]

where \( \text{train} \) and \( \text{car} \) are type-names, with definitions in \( T \). Here is a derivation sequence of

\[
p(X) \leftarrow \text{has\_car}(X, U), \text{has\_car}(X, V), \text{smaller}(U, V), U = V, \\
\text{has\_car}(X, Y), \text{short}(Y), U = Y
\]

from

\[
\{ p(X) \leftarrow \text{has\_car}(X, Y), \text{short}(Y), \\
p(X) \leftarrow \text{has\_car}(X, U), \text{has\_car}(X, V), \text{smaller}(U, V) \}
\]

using \( \{ \rho_1, \rho_2 \} \).

1. \( p(X) \leftarrow \text{has\_car}(X, U), \text{has\_car}(X, V), \text{smaller}(U, V) \) \quad Given
2. \( p(X) \leftarrow \text{has\_car}(X, U), \text{has\_car}(X, V), \text{smaller}(U, V), U = V \) \quad 1, \rho_1, U, V
3. \( p(X) \leftarrow \text{has\_car}(X, Y), \text{short}(Y) \) \quad Given
4. \( p(X) \leftarrow \text{has\_car}(X, U), \text{has\_car}(X, V), \text{smaller}(U, V), U = V, \\
\text{has\_car}(X, Y), \text{short}(Y) \) \quad 2, 3, \rho_2
5. \( p(X) \leftarrow \text{has\_car}(X, U), \text{has\_car}(X, V), \text{smaller}(U, V), U = V, \\
\text{has\_car}(X, Y), \text{short}(Y), U = Y \) \quad 4, \rho_1, U, Y

It is useful to define the notion of a \( \rho \)-derivation graph from a set of feature-clauses \( \Phi \).

**Definition 9 (\( \rho \)-derivation graph given \( \Phi \))** Let \( \gamma = (V, E, \phi) \) be a labelled DAG with vertices \( V \), edges \( E \) and vertex-labelling function \( \phi \). Let \( \text{Pred}(v) \) denote the set of immediate predecessors of any \( v \in V \). Let \( F_M \) be a set of feature-clauses given modes \( M \) and \( I \subseteq F_M \). Let \( \Phi \) be a set of feature-clauses in \( \mathcal{L}_{M', T} \). Then \( \gamma \) is a \( \rho \)-derivation graph given \( \Phi \) if:

- For each vertex \( v_i \in V \), \( \phi(v_i) = C_i \), where \( C_i \in F_M \);
- \( 0 \leq |\text{Pred}(v)| \leq 2 \) for all \( v \in V \);
- For each \( v \in V \):
  - If \( \text{Pred}(v) = \emptyset \) then \( \phi(v) \in \Phi \);
  - If \( \text{Pred}(v) = \{ u \} \) then \( \phi(v) \in \rho_1(\phi(u)) \);
  - If \( \text{Pred}(v) = \{ u_1, u_2 \} \) then \( \phi(v) \in \rho_2(\phi(u_1), \phi(u_2)) \)
Since we are only concerned with $\rho_1, \rho_2$ in this paper, we will usually call this the derivation graph given \( \Phi \) or even just the derivation graph, when \( \Phi \) is understood.

**Remark 4** We note that \( \rho_1 \) and \( \rho_2 \) preserve equivalence, in the following sense:

- If \( C' \equiv C \) then \( \rho_1(C) \equiv \rho_1(C') \); and
- If \( C' \equiv C_1 \) and \( C'' \equiv C_2 \) then \( \rho_2(C_1, C_2) \equiv \rho_2(C', C'') \)

Here, equivalence across sets has the usual conjunctive meaning. That is, for sets \( A, B, A \equiv B \iff \bigwedge x \in A \equiv \bigwedge y \in B \). Two ordered clauses \( \langle C_1 \rangle \) and \( \langle C_2 \rangle \) are equivalent iff the \( \text{Set}(\langle C_1 \rangle) \) is equivalent to the \( \text{Set}(\langle C_2 \rangle) \).

**Definition 10 (Closure)** Let \( \Phi \) be a set of feature-clauses and \( \Omega \subseteq \{\rho_1, \rho_2\} \). We define the closure of \( \Phi \) using \( \Omega \) as the set of ordered clauses \( \langle C \rangle \) which has a derivation sequence from \( \Phi \) using \( \Omega \). We use \( \text{Closure}_\Omega(\Phi) \) to denote the closure of \( \Phi \) using \( \Omega \).

We will say \( \theta \) is a type-consistent substitution if for every variable \( U \), the substitution \( U/t \in \theta \) (that is, \( \theta(U) = t \)), then \( U, t \) have the same type in \( M \). It follows that if \( \theta \) is a type-consistent substitution for variables in an ordered clause \( \langle C \rangle \) in \( \text{L}_M, T \) and \( \theta(u) = \theta(v) \) for \( u, v \) in \( \langle C \rangle \), then \( u, v \) have the same type in \( M \).

**Lemma 2 (Derivation Lemma)** Given \( M, M' \) and \( \Omega = \{\rho_1, \rho_2\} \) as before. Let \( \langle C \rangle \) be an ordered clause in \( \text{L}_M, T \), with head \( p(X) \). Let \( S \) be a set of ordered \( M \)-simple clauses in \( \text{L}_M, T \), with heads \( p(X) \) and all other variables of clauses in \( S \) standardised apart from each other and from \( C \). If there exists a substitution \( \theta \) s.t. Basis(\( \langle C \rangle \)) \( \subseteq S \theta \) then there exists an ordered clause \( \langle C' \rangle \) in \( \text{L}_M, T \) such that \( \langle C' \rangle \) is equivalent to \( \langle C \rangle \) and derivable from \( S \) using \( \Omega \).

**Proof** See Appendix C.

**Remark 5** Let \( \langle C_1 \rangle, \langle C_2 \rangle, \ldots, \langle C_n \rangle \) be a derivation from a set of ordered clauses \( S \) using \( \{\rho_1, \rho_2\} \). Also, for any clause \( \langle C_i \rangle \) in the derivation sequence, let \( f_i \) denote the corresponding feature-function as defined in Section 2 using background knowledge \( B \). Let \( a \) denote a data-instance. We note the following consequences for \( 1 \leq i < j \leq n \):

1. \( C_i \) subsumes \( C_j \)
2. If \( f_i(a) = 0 \) then \( f_j(a) = 0 \);
3. If \( f_i(a) = 1 \) then \( f_j(a) = 1 \); and
4. If \( \langle C_{i+1} \rangle \in p_t(\langle C_i \rangle) \), \( f_i(a) = 1 \) and \( f_{i+1}(a) = 0 \) then there exists a clause \( C'_{i+1} \) s.t. \( C_i \equiv C_{i+1} \lor C'_{i+1} \) and \( f_{B,C_i \cup C_{i+1}}(a) = 1 \)

(1) follows straightforwardly since for any \( \langle C_i \rangle \), subsequent clauses in the derivation only result in the addition of literals (that is, \( C_i \subseteq C_j \) for \( i < j \)). For (2), we note that since \( C_i \subseteq C_j \) and both \( C_i, C_j \) have the same head literal \( (p(X)) \) we can take \( C_i = \forall (p(X) \lor l_1 \lor \cdots \lor l_k) \) and \( C_j = \forall (p(X) \lor l_1 \lor \cdots \lor l_k \lor l_{k+1} \lor \cdots \lor l_m) \). If \( f_i(a) = 0 \) then \( B \cup C_i \not\subseteq \{a\} \). That is, there exists some interpretation \( I \) that is a model for \( B \cup C_i \)

\(^5\) Here subsumption is used in the sense described by Plotkin (1972). That is, representing clauses as sets of literals, clause \( C \) subsumes clause \( D \) iff there exists a substitution \( \theta \) s.t. \( C\theta \subseteq D \).
s.t. \( p(a) \) is false in \( I \). If \( I \) is a model for \( B \cup C_i \) then it is a model for \( C_i \). Further, if \( I \) is a model for \( C_i \) and \( p(a) \) is false in \( I \) then \( I \) is a model for \( \forall (l_i \lor \ldots \lor l_k) \). But then \( I \) is a model for \( C_j = \forall (p(X) \lor l_1 \lor \ldots \lor l_k \lor \ldots \lor l_m) \). Thus \( I \) is a model for \( B \cup C_j \) and not a model for \( p(a) \). That is, \( B \cup C_j \not\models p(a) \) and \( f_j(a) = 0 \). (3) follows from the fact that if \( C_i \) subsumes \( C_j \) then \( C_i \models C_j \) (Gottlob, 1987). Therefore, if \( B \cup C_j \not\models p(a) \) then \( B \cup C_i \not\models p(a) \). That is, if \( f_j(a) = 1 \) then \( f_i(a) = 1 \). For (4), let \( C_1 : p(X) \leftarrow \text{Body}_1(X,Y) \). Then \( C_{i+1} : p(X) \leftarrow \text{Body}_i(X,Y), Y_i = Y_j \) for \( y_{i,j} \in Y \). Since \( f_i(a) = 1 \) and \( f_{i+1}(a) = 0 \), it must be the case that \( Y_i = Y_j \) does not hold for \( x = a \). Let \( C_{i+1} : p(X) \leftarrow \text{Body}_i(X,Y), Y_i \neq Y_j \). It is evident that, \( f_{B,C_{i+1}}(a) = 1 \) and \( C_i \equiv C_{i+1} \lor C_{i+1}' \).

A specialised form of derivation results from the repeated use of \( \rho_2 \) first, followed by the repeated use of \( \rho_1 \). We call this form of derivation a linear derivation. We describe this next (relevant proofs are in Appendix D).

**Definition 11 (Linear Derivation of Feature-Clauses)** Let \( M \) be a set of mode-declarations, and \( M' \) be an extension \( M \) as earlier. Let \( T \) be a set of type-definitions. Let \( \Phi \) be a set of feature-clauses in \( \mathcal{L}_{M,T} \) and \( \rho_1, \rho_2 \) be the operators defined earlier. A sequence of feature-clauses \( \langle C_1 \rangle, \langle C_2 \rangle, \ldots, \langle C_n \rangle \) is said to be a linear derivation sequence of \( \langle C_n \rangle \) from \( \Phi \) using \( \{ \rho_1, \rho_2 \} \) iff there exists \( j \) such that \( 1 \leq j \leq n \) and:

- For \( i \leq j \):
  - Clause \( \langle C_i \rangle \) in the sequence is either an element of \( \Phi \) or an element of the set \( \rho_2(\langle C_{i-1} \rangle, \langle C_k \rangle) \) where \( \langle C_k \rangle \in \Phi \) and \( k < i \).
- For \( i > j \):
  - Clause \( \langle C_i \rangle \) is an element of the set \( \rho_1(\langle C_{i-1} \rangle) \).

We will say \( \langle C_i \rangle \) is linearly derivable from \( \Phi \) using \( \{ \rho_2 \} \); and \( C_n \) is linearly derivable from \( \Phi \) using \( \{ \rho_1, \rho_2 \} \).

**Example 8** We continue Example 7. Below is a linear derivation sequence of

\[
\begin{align*}
p(X) &\leftarrow \text{has}_\text{car}(X,U), \text{has}_\text{car}(X,V), \text{smaller}(U,V), \\
\text{has}_\text{car}(X,Y), \text{short}(Y), U = V, U = Y
\end{align*}
\]

from

\[
\{ \begin{array}{l}
p(X) \leftarrow \text{has}_\text{car}(X,Y), \text{short}(Y), \\
p(X) \leftarrow \text{has}_\text{car}(X,U), \text{has}_\text{car}(X,V), \text{smaller}(U,V)
\end{array} \}
\]

using \( \{ \rho_1, \rho_2 \} \).

1. \( p(X) \leftarrow \text{has}_\text{car}(X,U), \text{has}_\text{car}(X,V), \text{smaller}(U,V) \) \quad \text{Given}
2. \( p(X) \leftarrow \text{has}_\text{car}(X,Y), \text{short}(Y) \) \quad \text{Given}
3. \( p(X) \leftarrow \text{has}_\text{car}(X,U), \text{has}_\text{car}(X,V), \text{smaller}(U,V), \\
\text{has}_\text{car}(X,Y), \text{short}(Y), U = V \) \quad 1, \rho_2
4. \( p(X) \leftarrow \text{has}_\text{car}(X,U), \text{has}_\text{car}(X,V), \text{smaller}(U,V), \\
\text{has}_\text{car}(X,Y), \text{short}(Y), U = V \) \quad 3, \rho_1, U, V
5. \( p(X) \leftarrow \text{has}_\text{car}(X,U), \text{has}_\text{car}(X,V), \text{smaller}(U,V), \\
\text{has}_\text{car}(X,Y), \text{short}(Y), U = V, U = Y \) \quad 4, \rho_1, U, Y
There is no way to derive the clause $p(X) \leftarrow \text{has\_car}(X, U), \text{has\_car}(X, V), \text{smaller}(U, V), U = V, \text{has\_car}(X, Y), \text{short}(Y), U = Y$ using linear derivation from the given set, but we can derive an equivalent clause $p(X) \leftarrow \text{has\_car}(X, U), \text{has\_car}(X, V), \text{smaller}(U, V), \text{has\_car}(X, Y), \text{short}(Y), U = V, U = Y$ using linear derivation. We would like to point out that the positions of equality literal $U = V$ in the first clause and the second clause are different.

**Lemma 3 (Linear Derivation Lemma)** Given $M, M'$ and a set of ordered clauses $\Phi$. If an ordered clause $\langle C \rangle$ is derivable from $\Phi$ using $\{\rho_1, \rho_2\}$ then there exists an equivalent ordered clause $\langle C' \rangle$ and it is linearly derivable from $\Phi$ using $\{\rho_1, \rho_2\}$.

**Proof** See Appendix D.

Feature-clauses and their composition using the $\rho$-operators provide the tools for the development of a particular kind of neural network that we describe next.

### 3 Compositional Relational Machines (CRMs)

Formally, a CRM is defined as follows:

**Definition 12 (CRM)** A CRM is a 7-tuple $(V, I, O, E, \phi, \psi, h)$ where:

- $V$ denotes a set of vertices;
- $I \subseteq V$ is a set of “input” vertices;
- $O \subseteq V$ is a set of “output” vertices;
- $E \subseteq \{(v_i, v_j) : v_i, v_j \in V, v_i \notin O, v_j \notin I\}$;
- A vertex-labelling function $\phi : V \rightarrow \mathcal{F}_M \times \mathcal{G}$, where $\mathcal{F}_M$ is the set of feature-clauses given a set of modes $M$; and $\mathcal{G}$ denotes a set of activation functions.
- $\psi : E \rightarrow \mathbb{R}$ assigns some real-valued labels to edges in the graph; and
- $h : \mathbb{R}^{|O|} \rightarrow \mathbb{R}^n$ is a computation function, for some fixed $n$ such that $(V, E, \phi')$ is a derivation graph (Definition 9) given $\Phi$ where $\phi'(v) = C$ if $\phi(v) = (C, \cdot)$ and $\Phi = \{\phi'(v) \mid v \in I\}$.

We note 2 important features of CRMs: (1) Each vertex has a feature-clause associated with it; and (2) Edges between vertices in a CRM are required to satisfy the constraints on edges imposed by a derivation graph. That is, the only edges allowed are those that result from $\rho_1$ or $\rho_2$ operations on the feature-clauses associated with the vertices.

#### 3.1 CRMs as Explainable Neural Networks

We describe a use of CRMs as a form of neural network capable of generating logical explanations relevant to its prediction. The architecture of the neural network is inspired by Turing’s idea of unorganised machines (Turing, 1948) (see

---

$^6$ We assume activation functions in $\mathcal{G}$ are $\mathbb{R} \rightarrow \mathbb{R}$. 

Each “neuron” has 2 parts, implementing the vertex-label specification of a CRM’s node: (i) An arithmetic part that is concerned with the $g$-function in the CRM’s vertex-label; and (ii) A logical part that acts as a switch, depending on the feature-clause associated with the CRM’s vertex-label. We call neurons in such a network “arithmetic-logic neurons” or ALNs for short.

![Neural network diagram](image)

Fig. 1: A neural network implementation of a CRM inspired by Turing’s gated neural networks (Turing, 1948). A neuron $n_i$ corresponds to a vertex $v_i$ in the CRM, with vertex-label $\psi(v_i) = (C_i, g_i)$. In the figure, $n_i$ is connected to neurons $n_j$ and $n_k$, implying $(v_j, v_i)$ and $(v_k, v_i)$ are in the edge-set of the CRM. $f_i$ (correctly $f_{C_i, g_i}$) is the feature-function obtained from the feature-clause $C_i$ (see Section 2), and acts as a gate. For a data instance $a$, $g_i(w_{ji}h_j(a) + w_{ki}h_k(a))$ passes through the gate if and only if $f_i(a) = 1$. In general, $h_i(a)$ is thus $g_i(\sum_{k=\text{Pred}(n_i)} w_{ki}h_k(a))$ or 0, where $\text{Pred}(n_i)$ is the set of immediate predecessors of $n_i$ in the neural network.

In the rest of the paper, we will use “CRM” synonymously with this form of neural-network implementation. The 7-tuple defining a CRM $(V, I, O, E, \phi, \psi, h)$ corresponds to the following aspects of the neural implementation: (a) The structure of the network is defined by $V, I, O, E, \phi$; (b) The parameters of the network are defined by $\psi$; and (c) the computation of the network is defined by $h$. We consider each of these in turn.

### 3.1.1 Structure Selection

Procedure 1 is an enumerative procedure for obtaining a 5-tuple $(V, I, O, E, \phi)$, given a set of feature-clauses $\Phi$. For simplicity, the procedure assumes a single activation function $g$.

Procedure 1 has an important practical difficulty:

- We are interested in a class of CRMs that can be constructed using a set of $M$-simple feature-clauses $\{C_1, \ldots, C_n\}$. Now, it may be impractical to obtain all possible $M$-simple feature-clauses in a mode-language. Even if this were not the case, it may be impractical to derive all non-simple clauses in the manner shown in Procedure 1.

Procedure 2 describes a randomised implementation to address this. The procedure also uses the result in the Linear Derivation Lemma (Lemma 3 in Section 2) to construct a CRM structure that first uses the $\rho_2$ operator, followed by the $\rho_1$ operator.
Procedure 1: (ConstructCRM) Depth-bounded construction of a CRM’s structure

Input: A set \( \Phi \) of feature-clauses \( \{C_1, C_2, \ldots, C_n\} \) with heads \( p(X) \) and all other variables are standardised apart from each other; an activation function \( g \); and a bound \( d \) on the depth of composition

Output: A CRM structure

1. \( I = \{v_1, v_2, \ldots, v_n\} \)
2. \( V = I \)
3. \( E = \emptyset \)
4. for \( i = 1 \) to \( n \) do
5. \( \phi(v_i) = (C_i, 1) \)
6. for \( j = 1 \) to \( d \) do
7. \( V_{j,1} = \{(C', v) : v \in V, \phi(v) = (C', \cdot), C' = \rho_1(C)\} \)
8. \( V_{j,2} = \{(C', v_1, v_2) : v_1, v_2 \in V, \phi(v_1) = (C_1, \cdot), \phi(v_2) = (C_2, \cdot), C' = \rho_2(C_1, C_2)\} \)
9. Let \( V_1 \) be a set of new vertices s.t. there exists \( (C', v') \) in 1 and \( \phi(v) = (C', g) \)
10. Let \( V_2 \) be a set of new vertices s.t. there exists \( (C', v_1, v_2) \) in 2 and \( \phi(v) = (C', g) \)
11. Let \( V' = V_1 \cup V_2 \)
12. \( V = V \cup V' \)
13. \( E = E \cup \{(v_1, v) : (C, v_1) \in V_{j,1}, \phi(v) = (C, g)\} \cup \{(v_1, v), (v_2, v) : (C, v_1, v_2) \in V_{j,2}, \phi(v) = (C, g)\} \)
14. \( O = \{v : v \in V, \exists v' s.t. (v, v') \in E\} \)
15. return \((V, I, O, E, \phi)\)

In the rest of the paper, we will use the term Simple CRM to denote a CRM constructed by either Procedure 1 or Procedure 2 in which the input clauses \( C_1, C_2, \ldots, C_n \) are M-simple feature-clauses.

3.1.2 Parameter Estimation

Procedure 2 does not completely specify a CRM. Specifically, neither the edge-labelling \( \psi \) nor \( h \) are defined. We now describe a procedure that obtains a \( \psi \) given the partial-specification returned by Procedure 2 and a pre-specified \( h \) suitable for the usual task of using the neural network for function approximation. That is, given a partial specification of an unknown function \( \delta : A \rightarrow Y \) in the form of sample data \( D = \{(a_i, y_i)\}_1^N \). We want the neural network to construct an approximation \( \hat{\delta} : A \rightarrow Y \) that is reasonably consistent with \( D \). In order to estimate the goodness of the approximation, we need to define a loss function, that computes the penalty of using \( \hat{\delta} \). We will take \( \hat{\delta} \) to be synonymous with \( h \), the computation function of the CRM. Recall \( h : \mathbb{R}^{|O|} \rightarrow \mathbb{R}^n \) for some fixed \( n \). In this paper, we will therefore take \( Y = \mathbb{R}^n \) and define \( h \) in the usual manner adopted by neural networks, namely as a function of “local” computations performed at each of the \( O \) vertices of the CRM.

Definition 13 (Local Computation in a CRM) Let \((V, I, O, E, \phi, \psi)\) be a partially-specified CRM, where \( O = \{o_1, o_2, \ldots, o_k\} \). For each vertex \( v_i \in V \) let \( \phi(v_i) = (C_i, g_i) \) and for each edge \((v_i, v_j)\) \( \psi(v_i, v_j) = w_{ij} \). Let \( f_i \) denote \( f_{C_i} \). For any \( a \in A \) we define \( h_i : A \rightarrow \mathbb{R} \) as follows:

\[
h_i(a) = \begin{cases} f_i(a) & \text{if } v_i \in I \\ f_i(a)g_i \left( \sum_{(v_j, v_i) \in E} w_{ij}h_j(a) \right) & \text{if } v_i \notin I \end{cases}
\]
Procedure 2: (RandomCRM) Randomised construction of a CRM structure, with linear derivation of feature-clauses.

Input: A set $\Phi$ of feature-clauses $\{C_1, C_2, \ldots, C_n\}$ with heads $p(X)$ and all other variables are standardised apart from each other; an activation function $g$; a sample size $s$; bounds $d_{\rho_1}, d_{\rho_2}$ on the depth of application of the $\rho_1$ and $\rho_2$ operators

Output: A CRM structure

1. $I = \{v_1, v_2, \ldots, v_n\}$
2. $V_0 = I$
3. $V = V_0$
4. $E = \emptyset$
5. for $i = 1$ to $n$ do
   6. $\phi(v_i) = (C_i, 1)$
   7. for $j = 1$ to $d_{\rho_1} + d_{\rho_2}$ do
      8. $V_j = \emptyset$
      9. $E_j = \emptyset$
     10. if $j \leq d_{\rho_2}$ then
        11. $op = \rho_2$
        12. else
        13. $op = \rho_1$
        14. for $i = 1$ to $s$ do
           15. if $op = \rho_2$ then
              16. Sample a vertex $v_1$ from $I$ using a uniform distribution and sample a vertex $v_2$ from $V_{j-1}$ using the uniform distribution;
              17. Create a vertex $v'$ such that $\phi(v') = (C', g)$ where $C' = \rho_2(C_1, C_2)$, $\phi(v_1) = (C_1, g)$ and $\phi(v_2) = (C_2, g)$;
              18. $V_j = V_j \cup \{v'\}$
              19. $E_j = E_j \cup \{(v_1, v'), (v_2, v')\}$
           20. else
           21. Sample a vertex $v$ from $V_{j-1}$ using the uniform distribution;
           22. Create a vertex $v'$ such that $\phi(v') = (C', g)$ where $C' = \rho_1(C)$ and $\phi(v) = (C, g)$;
           23. $V_j = V_j \cup \{v'\}$
           24. $E_j = E_j \cup \{(v, v')\}$
        25. $V = V \cup V_j$
        26. $E = E \cup E_j$
     27. $O = \{v : v \in V, \exists v' s.t. (v, v') \in E\}$
     28. return $(V, I, O, E, \phi)$

Then $\hat{\delta}(a) = h(h_{o_1}(a), \ldots, h_{o_k}(a))$.

For a multi-class classification task, function $h$ computes the probability distribution over the classes, for example, a softmax function. Similarly, for a regression task, $h$ computes a real number, for example, a linear function.

Procedure 3 estimates the parameters of the neural network using a standard weight-update procedure based on stochastic gradient descent (SGD) (Rumelhart et al., 1986; Goodfellow et al., 2016), given the structure obtained from Procedure 2, a pre-defined computation function $h$, and a loss function $L$.

3.1.3 Predictions and Explanations

We denote the prediction of a CRM $\gamma = (V, I, O, E, \phi, \psi, h)$ by $\hat{\delta}(a) = h(h_{o_1}(a), \ldots, h_{o_k}(a))$, where $O = \{o_1, \ldots, o_k\}$ and the $h_{o_i}$ are as defined in Definition 13.
Procedure 3: (TrainCRM) Parameter estimation of a CRM, given its structure, using stochastic gradient descent (SGD). The training is done until some stopping criterion is reached, which refers to the condition when the number of training epochs reaches some pre-specified maximum value.

**Input:** A CRM structure $\gamma = (V, I, O, E, \phi)$, a dataset $D = \{(a_i, y_i)\}_{i=1}^N$, where $a_i \in A$ and $y_i \in \mathbb{R}^n$, a computation function $h : \mathbb{R}^{|O|} \rightarrow \mathbb{R}^n$, a loss function $L : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$.

**Output:** A CRM

1. Let $O = \{o_1, \ldots, o_k\}$;
2. Initialise $\psi$;
3. while stopping criterion is not met do
4. Randomly draw an instance $(a_i, y_i)$ from $D$;
5. Let $\hat{y}_i = h(h_{o_1}(a), \ldots, h_{o_k}(a))$ (see Definition 13);
6. Error $= L(y_i, \hat{y}_i)$;
7. Update $\psi$ using SGD to minimise Error;
8. return $(V, I, O, E, \phi, \psi, h)$;

The association of feature-clauses with every vertex of the CRM allows us to construct “explanations” for predictions. For this we introduce the notion of ancestral graph of a vertex and explanation graph of an output vertex in a CRM.

**Definition 14 (Ancestral Graph of a Vertex)** Let $\gamma = (V, I, O, E, \phi, \psi, h)$ be a CRM. The set of ancestors of a vertex $v \in V$ in $\gamma$, denoted by $\text{Ancestors}(v, \gamma)$, is defined as follows:

$$\text{Ancestors}(v, \gamma) = \begin{cases} \{v\} & \text{if } v \in I \\ \bigcup_{(u, v) \in E} \text{Ancestors}(u, \gamma) \cup \{v\} & \text{otherwise} \end{cases}$$

The ancestral graph of a vertex $v$ in $\gamma$ is $(V', E')$ where $V' = \text{Ancestors}(v, \gamma)$ and $E' = \{(u', u'') : u', u'' \in V', (u', u'') \in E\}$.

**Definition 15 (Explanation Graph)** Let $\gamma = (V, I, O, E, \phi, \psi, h)$ be a CRM, and $a \in A$ be a data instance. Let $O = \{o_1, \ldots, o_k\}$ and let $\delta(a) = h(h_{o_1}(a), \ldots, h_{o_k}(a))$ be the prediction of the CRM for $a$. For $o_i \in O$, let $C_i$ be the feature-clause associated with $o_i$ (that is, $\phi(o_i) = (C_i, \cdot)$). Let $f_i$ be the corresponding feature-function (as defined in Section 2), and $(V', E')$ be an ancestral graph of $o_i$ in $\gamma$. Then the explanation graph of $a$ from vertex $o_i$, denoted by $\text{Explain}_{\gamma, o_i}(a)$, is as follows:

$$\text{Explain}_{\gamma, o_i}(a) = \begin{cases} (V', E', \phi') & \text{if } f_i(a) = 1 \\ \emptyset & \text{otherwise} \end{cases}$$

where $\phi' : V' \rightarrow \mathcal{F}_M$ is a vertex-labelling function, $\phi'(v) = C\theta_a$, where $\theta_a$ the substitution $\{X/a\}$ for the variable $X$ in the head literal and $\phi(v) = (C, \cdot)$.

**Remark 6** $\text{Explain}_{\gamma, o_i}(a)$ consists of a (labelled) tree of feature-clauses extracted from the derivation graph of feature-clauses. The root of the tree is the feature-clause at $o_i$, and sub-trees contain simpler feature-clauses. If the CRM is a Simple CRM, then the leaves of the explanation-tree are M-simple feature-clauses.
Example 9 In the Trains problem, suppose the data instance is the train shown on the left below. The explanation graph, associated with an output vertex of the CRM is shown on the right.

By definition, we know that the feature-function value associated with $p(t_1) \leftarrow \text{has\_car}(t_1, A), \text{short}(A), \text{has\_car}(t_1, B), \text{closed}(B), A = B$ has the value 1. Also, we know that feature-function values with all other clauses in the explanation will also be 1.

The notion of an explanation graph from a vertex extends naturally to the explanation graph from a set of vertices which we do not describe here. It will be useful in what follows to introduce the notion of a feature-clause being “contained in an explanation graph”.

Definition 16 (Feature-clause Containment) Let $\gamma$ be a CRM, $o$ be an output vertex of $\gamma$. Let $a$ be a data instance, Let $\text{Explain}_{\gamma,o}(a) = (V, E, \alpha)$, and $F_{\gamma,o}(a) = \{\alpha(v) | v \in V\}$ be the set of feature-clauses in the explanation graph. We will say a feature-clause $C$ is contained in $\text{Explain}_{\gamma,o}(a)$, or $C \sqsubseteq \text{Explain}_{\gamma,o}(a)$ iff there exists $C' \in F_{\gamma,o}(a)$ s.t. $C \theta_a \equiv C'$ (where $\theta_a = \{X/a\}$ is a substitution for the variable $X$ in the head of $C$).

Example 10 The feature-clause $C : p(X) \leftarrow \text{has\_car}(X, Y), \text{short}(X), \text{closed}(X)$ is contained in the explanation graph shown for train $t_1$ in Figure 6 because:

- $F_{\gamma,o}(t_1) = \{C_1, C_2, C_3, C_4\}$ where:
  - $C_1 : p(t_1) \leftarrow \text{has\_car}(t_1, A), \text{short}(A), \text{has\_car}(t_1, B), \text{closed}(B), A = B$;
  - $C_2 : p(t_1) \leftarrow \text{has\_car}(t_1, A'), \text{short}(A'), \text{has\_car}(t_1, B'), \text{closed}(B')$;
  - $C_3 : p(t_1) \leftarrow \text{has\_car}(t_1, A''), \text{short}(A'')$; and
  - $C_4 : p(t_1) \leftarrow \text{has\_car}(t_1, A''), \text{closed}(A'')$

- With $\theta_a = \{X/t_1\}$, $C \theta_a = (p(t_1) \leftarrow \text{has\_car}(t_1, A), \text{short}(A), \text{closed}(A))$; and
- $C \theta_a \equiv C_1$
Explanatory Fidelity

Explanatory fidelity refers to how closely the CRM’s explanation matches the “true explanation”. Of course, in practice, explanatory fidelity will be a purely notional concept, since the true explanation will not be known beforehand. However it is useful for us to calibrate the CRM’s explanatory performance when it is used for problems where true explanations are known (the synthetic problems considered in experiments below are in this category).

For a prediction \(\hat{\delta}(h_0(a), \ldots, h_0(a))\) by a CRM, suppose we have a relevance ordering over the output vertices \(o_1, \ldots, o_k\). Let \(o^*\) be the most relevant vertex in this ordering. Then we will call the explanation graph from \(o^*\) as the most-relevant explanation graph for \(a\) given the CRM.\(^7\)

For a classification task, we use clause containment and the most-relevant explanation graph to arrive at a notion of explanatory fidelity of a CRM to a set of feature-clauses \(T_c\) that are known to be ‘acceptable’ feature-clauses for class \(c\) (if no such acceptable clauses exist for class \(c\), then \(T_c = \emptyset\)). Let \(\gamma\) be a CRM used to predict the class-labels for a set of data-instances. For any instance \(a\), let \(o^*\) denote the most relevant output vertex of the CRM. We will say that a data instance \(a\) is consistently explained iff: (i) the CRM predicts that \(a\) has the class-label \(c\); and (ii) there exists a \(C \in T_c\) s.t. \(C \subseteq \text{Explain}_{\gamma, o^*}(a)\); and (iii) for \(c'^* \neq c\), there does not exist \(C' \in T_{c'}\) s.t. \(C' \subseteq \text{Explain}_{\gamma, o^*}(a)\).

Given a set of data-instances \(E\), let \(CE\) denote the set of instances in \(E\) explained consistently and \(IE\) denote the set of instances in \(E\) not explained consistently. Then the explanatory fidelity of the CRM (correctly, this is only definable w.r.t. the \(T_c\)’s) is taken to be \(\frac{|CE|}{|CE| + |IE|}\), provided \(|CE| + |IE| \neq 0\) (and undefined otherwise).

3.2 CRMs as Explanation Machines

CRMs can be used as ‘explanation machines’ for black-box predictors that do not intrinsically include an explanatory component. The approach, sometimes called post hoc explanation generation, is shown in Figure 2.

To assess the utility of using the CRM in this manner, we will change the usual assessment of predictive accuracy to one of ‘predictive fidelity’, which refers to how closely the CRM matches the prediction of the target model.

4 Empirical Evaluation of CRMs as Explanation Machines

4.1 Aims

We consider two kinds of experiments with Simple CRMs:

\(^7\) In implementation terms, one way to obtain such a relevance ordering over output vertices of the CRM is to use the \(h_i(\cdot)\) values for vertices in \(O\) to select a vertex \(o^*\) that has the highest magnitude (this is the same as selecting the best vertex after one iteration of the layer-wise relevance propagation, or LRP (Binder et al., 2016), procedure).
Fig. 2: Using a CRM as an explanation machine. The target model is a “black box” that does not have an explanatory output. The CRM model is trained using training data labelled with the prediction from the black box (and not the ‘true label’). The multiplexer (MUX) selects between the CRM’s explanation graph and the “empty” explanation $\emptyset$ depending on whether the CRM’s prediction does or does not match the target model’s prediction. By using the setup as shown here, we are able to get a prediction $P$ and a corresponding explanation $E$.

**Synthetic data.** Using tasks for which both target-model predictions and acceptable feature-clauses are available, we intend to investigate the hypothesis that: (a) Simple CRMs can construct models with high predictive fidelity to the target’s prediction; and (b) Simple CRMs have high explanatory fidelity to the set of acceptable feature-clauses.

**Real data.** Using real-world datasets, for which we have predictions from a state-of-the-art black-box target model, we investigate the hypothesis that Simple CRMs can construct models with high predictive fidelity to the target’s prediction. We also provide illustrative examples of using the CRM to provide explanations for the predictions.

We clarify what is meant by ‘acceptable feature-clauses’ for the synthetic data in Section 4.3. For real data, the target-model is the state-of-the-art (SOTA, which in this case is a graph-based neural network). That is, the CRM is being used here to match the SOTA’s predictions (and not the ‘true value’), and to provide proxy explanations. No acceptable feature-clauses are known for classes in the real data.\(^8\)

### 4.2 Materials

#### 4.2.1 Data and Background Knowledge

**Synthetic Data.** We use two well-known synthetic datasets. The first dataset is the “Trains” problem of discriminating between eastbound (class = +) and westbound trains (class = −) (Michalski, 1980). The original data consists only of 10 instances (5 in each class). We generate a dataset of 1000 instances with a class-distribution of approximately 50% + and 50% −, using the data.

---

\(^8\) The CRM can of course be used to predict the true value directly. We will comment on this later, but that is not the primary goal of the experiment here.
generator (Michie et al., 1994). We use 700 instances as training data and 300 instances as test-data. The second dataset consists of the task of discriminating between illegal (class = +) and legal (class = −) chess positions in the King-Rook-King endgame (Bain, 1994; Michie, 1976). The class-distribution is approximately 33% + and 67% −. We use 10000 instances of board-positions as training data and 10000 instances as test-data. Examples of + instances are showed pictorially in Figure 3.

![Trains and Chess](image)

**Fig. 3:** Pictorial examples of positive instances in the synthetic data. The actual data are logical encodings of examples like these. The instance on the left is an example of a train classified as “eastbound” (+). The instance on the right is of a board position classified as “illegal” (+), given that it is White’s turn to move. For both problems, we have a target model that is complete and correct. We also know a set of feature-clauses that are acceptable as explanations for instances that are correctly predicted as +.

**Real Data.** Our real data consists of 10 datasets obtained from the NCI\(^9\). Each dataset represents extensive drug evaluation with the concentration parameter GI50, which is the concentration that results in 50% growth inhibition of cancer cells (Marx et al., 2003). A summary of the dataset is presented in Figure 4. Each relational data-instance in a dataset describes a chemical compound (molecule) with atom-bond representation: a set of bond facts. The background knowledge consists of logic programs defining almost 100 relations for various functional groups (such as amide, amine, ether, etc.) and various ring structures (such as aromatic, non-aromatic etc.). There are also higher-level domain-relations that determine presence of connected, fused structures. Some more details on the background knowledge can be seen in these recent studies: (Dash et al., 2021, 2022).

### 4.2.2 Algorithms and Machines

We use the ILP system Aleph (Srinivasan, 2001) for constructing the feature-clauses. The CRMs are implemented using PyTorch (Paszke et al., 2019). The

\(^9\) The National Cancer Institute ([https://www.cancer.gov/](https://www.cancer.gov/))
Fig. 4: Summary of the NCI-50 datasets (Total no. of instances is approx. 30,200).
The graph neural network predictor described in (Dash et al., 2021) is taken as
the target model. No acceptable feature-clauses are known for these tasks.

4.3 Method

The experiments are in two parts: an investigation on synthetic data to examine
the predictive performance and explanatory fidelity of CRMs; and an investigation
on real data, to compare the predictive performance of CRMs against state-of-
the-art deep networks. Some examples of explanations are also provided for the
explanations generated by a CRM on real data. We describe the method used for
each part in turn.

4.3.1 Experiments with Synthetic Data

For both synthetic datasets, we have access to symbolic descriptions of the true
concepts involved. The ‘target model’ in each case is taken to be equivalent to
a classifier that labels instances consistent with the corresponding true concept.
This allows us to judge the fidelity of explanations generated. The method used
in each case is straightforward:

For each problem:
(a) Construct the dataset \( D \) of instances labelled by the target model;
(b) Generate a subset of M-simple feature-clauses in the mode-language for
the problem;
(c) Randomly split \( D \) into training and test samples;
(d) Construct a CRM using Procedure 2 with the M-simple features. The
weights for the CRM are obtained using the training data and the SGD-
based weight-update steps described in Procedure 3 (see below for addi-
tional details);
(e) Obtain an estimate of the predictive and the explanatory fidelity of the
CRM using the test data (again, see below for details).

The following additional details are relevant to the method just described:
– For both datasets, the composition depth of CRMs is at most 3. Also, the mode-declarations for Chess allow the occurrence of equalities in M-simple features (see Appendix F), additional compositions using $\rho_1$ are not used in this problem;
– We use the rectified linear (relu) activation function for the local computation in the neurons of the internal (hidden) layers of the CRMs.
– We use Adam optimiser (Kingma and Ba, 2015) to minimise the training cross-entropy loss between the true classes and the predicted classes by the network;
– We provide as input feature-clauses only a subset of all possible M-simple feature-clauses. The subset is constrained by the following: (i) At most 2 body literals; (ii) Minimum support of at least 10 instances$^{10}$; and (iii) Minimum precision of at least 0.5. All subsequent feature-clauses obtained by composition are also required to satisfy the same support and precision constraints. The learning rate for the Adam optimiser is set to 0.001 while keeping other hyperparameters to their defaults within PyTorch;
– The number of training epochs is set to 5 for the Trains dataset and 10 for the Chess dataset;
– Both synthetic problems are binary classification tasks. We call the classes + and − for convenience. Predictive fidelity is estimated in the usual manner, namely as the proportion of correctly predicted test-instances;
– Explanatory fidelity is estimated as described in Section 3.1.3. For this, we need to pre-define sets of feature-clauses that are acceptable in explanations. For the synthetic datasets, we are able to identify sets of acceptable feature-clauses from the literature: These feature-clauses are obtained from a target model that is known to be complete and correct (see (Michie et al., 1994) for the target model for Trains and (Bain, 1994) for Chess). The acceptable clauses in $T_+$ are as follows:

| Problem | Acceptable Feature Clause |
|---------|---------------------------|
| Trains  | $p(X) \leftarrow \text{has\_car}(X,Y), \text{short}(X), \text{closed}(X)$ |
| Chess   | $p((A,B,C,D,E)) \leftarrow \text{true}$ |
|         | $p((A,B,C,D,E,D)) \leftarrow \text{true}$ |
|         | $p((A,B,A,B,C,D)) \leftarrow \text{true}$ |
|         | $p((A,B,C,D,E,F)) \leftarrow \text{adj}(A,E), \text{adj}(B,F)$ |

In Trains, the feature clauses apply to descriptions of trains. For Chess these are descriptions of the board in an endgame. The board is a 6-tuple that denotes the file and rank of the position of the White King, White Rook and Black King, respectively. In all cases $T_- = \emptyset$. Explanatory fidelity will be estimated by checking clause-containment of the feature-clauses above in the explanation graph for the most-relevant output vertex of the CRM (see Section 3.1.3);
– The acceptable feature-clause for the Trains is a direct rewrite of the function used to generate the labels. For Chess, the (set of) feature-clauses are direct

$^{10}$ In principle, increasing the number of input clauses will increase the size (width and breadth) of the CRM (measured by the number of layers and neurons in each layer of the CRM). Furthermore, since complex features are more specific than the features represented by simple clauses, the coverage of the complex features (that is, instances for which the features have the value 1) will usually be lower than those of simple clauses. Thus, if we restrict complex features to those having a positive coverage of at least $p$, simple features will have also have a coverage of at least $p$. Thus, simple features with a lower positive coverage will not be part of any connection in the CRM, and therefore need not appear in the inputs.
rewrites of an approximate symbolic description taken from (Srinivasan et al., 1992). These approximate description isn’t identical to the correct description, but is very closely related to it (the approximation differ from the correct description only in about 40 of 10,000 cases). For our purposes therefore, high explanatory fidelity, w.r.t. the set of feature-clauses shown, will taken to be sufficient; and

- We provide a baseline comparison for predictive fidelity against a ‘majority class’ predictor. A baseline comparison is also provided for explanatory fidelity against random selection of a feature-clause from the set of feature-clauses associated with the output vertices of the CRM that have a feature-function value of 1 for the data instance being predicted.

4.3.2 Experiments with Real Data

For the real-world datasets, the current state-of-the-art predictions are from a Graph Neural Network (GNN) constructed using the background knowledge described earlier (Dash et al., 2022). However, the GNN model constitutes a black-box model, since it does not produce any explanations for its predictions. We investigate equipping this black-box model with CRM model for explanation. The method is as follows:

For each problem:
(a) Construct the dataset \( D \) consisting of problem instances and their predictions of the target model;
(b) Generate a subset of M-simple feature-clauses in the mode-language for the problem. The restrictions used for synthetic data are used to constrain the subset;
(c) Construct a CRM using Procedure 2 with the M-simple features and the dataset \( D \). The weights for the CRM are obtained using the training data used by the state-of-the-art methods and the SGD-based weight-update procedure; and
(d) Obtain the predictive fidelity of the CRM model to the predictions of the target model.

The following additional details are relevant:

- As with the synthetic data, the compositional depth for the CRMs is set to 3. Again, we do not use \( \rho_1 \) operations, since the mode-declarations allow equalities. The constraints on input feature-clauses is the same as those used for synthetic data;\(^{11}\)
- The CRM implementation is the same as the one used for synthetic data. We perform a grid-search of the learning rate for the Adam optimiser using the parameter grid: \( \{0.01, 0.001, 0.005, 0.0001\} \). The total number of training epochs is 10, with early-stopping mechanism (Prechelt, 1998) with a patience period of 3;

\(^{11}\) Our choice for the compositional depth of 3 is also loosely-based on our previous work on Deep Relational Machines (DRMs: (Dash et al., 2018)). However, we note that a depth higher than 3 will increase the complexity of a CRM (due to increase in number of layers and neurons), which might result in better predictive performance of a CRM. We expect that in practice the depth bound will be treated as a hyperparameter and subject to the usual forms of hyperparameter optimisation.
– As with the synthetic data, we provide a baseline comparison against the ‘majority class’ predictor;
– Unlike the synthetic data, no pre-defined set of acceptable feature-clauses exists, and therefore no estimate of explanatory fidelity is possible. Correspondingly, there is no baseline provided either.

4.4 Results

Figure 5 tabulates the results used to compute estimates of predictive and explanatory fidelity on synthetic and real data. The main details in these tabulations are these: (a) For the synthetic data, Simple CRMs models able to match the target’s prediction perfectly (predictive fidelity of 1.0); (b) The high explanatory fidelity values show that for instances labelled +, the maximal explanation for the most-relevant vertex contains at least 1 clause from the set of acceptable feature-clauses; and for instances labelled −, the maximal explanation of the most relevant vertex does not contain any clauses from the target theory; and (c) On the real datasets predictive fidelity of CRMs is reasonably high: suggesting that about 81% of the time, the CRM’s prediction will match that of the state-of-the-art model.

| Dataset | Pred. Fidelity | CRM | Baseline |
|---------|----------------|-----|----------|
| Trains  | 1.0            | 0.5 | 0.4      |
| Chess   | 1.0            | 0.7 | 0.7      |
|         |                | 0.77| 0.79     |
| A498    | 0.85           | 0.73| 0.58     |
| A549_ATCC | 0.78         | 0.73| 0.58     |
| ACHN    | 0.81           | 0.78| 0.51     |
| BT_549  | 0.81           | 0.78| 0.51     |
| CAKI_1  | 0.81           | 0.78| 0.51     |
| CCRF_CEM| 0.82           | 0.78| 0.51     |
| COLO_205| 0.77           | 0.78| 0.51     |
| DLD_1   | 0.90           | 0.78| 0.51     |
| DMS_114 | 0.89           | 0.78| 0.51     |
| Avg.    | 0.81 (0.05)    | 0.66 (0.17)|

Fig. 5: Estimates of fidelity for Simple CRMs on the synthetic datasets and real datasets. Explanatory fidelity is assessable on synthetic data since we have access to the “correct” explanation. Baseline for prediction is the majority class predictor. For explanations, Baseline refers to random selection of a feature-clause from the set of feature-clauses associated with the output vertices of the CRM with function value 1 for the data instance being predicted by the majority class predictor.

We now turn to examine the results in greater detail.
4.4.1 Predictive Fidelity

Although we obtain perfect predictive fidelity to the target model on synthetic data, fidelity on the real datasets clearly has room for improvement. Improvements in fidelity are possible simply by considering ensembles of CRMs, obtained simply due to the sampling variation arising in Step (c) (refer Section 4.3.2). Below, we tabulate changes in predictive fidelity on 1 of the real-world problems (786_0), using a sample consisting of up to 3 CRMs. With multiple CRMs, for a data-instance to be correctly predicted it is sufficient for any one of the CRMs to predict the same class as the target-model. Recall the primary purpose of the CRM is to explain the target-model’s prediction in terms of its feature-clauses. Any CRM that matches the target-model’s prediction can be used to explain the prediction. More on this under “Explanation” below.

| No. of CRMs | Predictive Fidelity |
|-------------|---------------------|
| 1           | 0.75                |
| 2           | 0.83                |
| 3           | 0.85                |

4.4.2 Explanatory Fidelity

For the synthetic datasets we show below in Figure 6 a representative instance (shown pictorially for ease of understanding), along with the predictions of both target and the CRM. The last column shows an acceptable feature-clause along with a stylised English translation. In both instances, an equivalent form of the acceptable feature-clause is contained in the CRM’s explanation graph.

For the Chess data, the CRM’s explanatory fidelity is less than 1. This means that there are instances for which the CRM’s explanation graph does not contain an acceptable feature-clause. We discover 19 different kinds of ‘buggy’ explanations are found by the CRM: a full listing is in Appendix F. Here we provide illustrative instances of two kinds of errors: those that are close to the correct explanation; and those that are an artifact of the specific data instance being explained (see Figure 7). Besides these, in many cases, we find explanation errors arise from the fact that definitions in background knowledge of file and rank adjacency hold when ranks and files are the same (that is, if \( A \) and \( B \) are ranks (or files) and \( A = B \), then \( \text{adj}(A, B) \) is true).12 In many instances inconsistent explanations result from feature-clauses contain literals that use equality instead of adjacency (that is, the CRM’s explanation contains \( A = E \), rather than \( \text{adj}(A, E) \); see Section F.2 in Appendix F). However, even accounting for this, the CRM’s explanations can be more specific than the correct explanation; and in some cases, incorrect (an example of each is shown in Figure 7).

What about explanations on real data? At this point, we do not have any independent source of acceptable feature-clauses for this data. We nevertheless show a representative example of the explanation for a test-instance (see Figure 8).

12 M. Bain, the author of the background definitions, confirms that this is the intended meaning of the \( \text{adj}/2 \) predicate for this problem (personal communication).
We close this examination by drawing the reader's attention to an important aspect of a CRM's explanation. The feature-clauses are defined in terms of relations provided as prior knowledge. This makes them potentially intelligible to a person familiar with the meanings of these relations. This makes it easier in
Instance Explanation Graph

\[
p((A, B, C, D, E, F)) \leftarrow \text{adj}(A, E), \text{adj}(B, F)
\]

White King’s file is adjacent to Black King’s file and White King’s rank is adjacent to Black King’s rank.

Fig. 7: Examples where the target model and the CRM’s prediction are both +, but the CRM’s explanation graph does not contain an acceptable feature-clause. For simplicity, we do not show the substitutions for \(A \ldots F\).

Fig. 8: A CRM explanation for a prediction on the real data. The class-label predicted by both CRM and the black-box model for this data instance is +.

principle at least—to perform a human-based assessment of the feature-clauses in the explanation graph (this is apparent from the ‘debugging’ of explanations that we have been to accomplish with the Chess data).
4.4.3 Additional Results: CRMs as Prediction Machines

The tabulations of fidelity and the subsequent assessments above provide a measure of confidence in the use of CRMs as explanation machines. But it is evident that CRMs can be used as ‘white-box’ predictors in their own right. We provide an indicative comparison of a CRM predictor against the state-of-the-art predictors (for the real-data, the prediction is by majority-vote from an ensemble of 3 CRMs):

| Dataset | Predictive accuracy |
|---------|---------------------|
|         | CRM     | GNN     | DRM (500) | CILP++ | Baseline |
| 786_0   | 0.66 (0.01) | 0.69 (0.01) | 0.69 (0.01) | 0.67 (0.01) | 0.55 (0.01) |
| A498    | 0.67 (0.01) | 0.72 (0.01) | 0.70 (0.01) | 0.66 (0.01) | 0.52 (0.01) |
| A549_ATCC | 0.64 (0.01) | 0.67 (0.01) | 0.70 (0.01) | 0.60 (0.01) | 0.51 (0.01) |
| ACHN    | 0.64 (0.01) | 0.70 (0.01) | 0.70 (0.01) | 0.64 (0.01) | 0.51 (0.01) |
| BT_549  | 0.66 (0.01) | 0.68 (0.01) | 0.70 (0.01) | 0.65 (0.01) | 0.53 (0.01) |
| CAIL1   | 0.63 (0.01) | 0.68 (0.01) | 0.66 (0.01) | 0.64 (0.01) | 0.54 (0.01) |
| CCRF_CEM | 0.65 (0.01) | 0.71 (0.01) | 0.71 (0.01) | 0.68 (0.01) | 0.63 (0.01) |
| COLO_205 | 0.60 (0.01) | 0.69 (0.01) | 0.67 (0.01) | 0.66 (0.01) | 0.56 (0.01) |
| DLD_1   | 0.69 (0.02) | 0.69 (0.02) | 0.70 (0.02) | 0.72 (0.02) | 0.69 (0.02) |
| DMS_114 | 0.68 (0.02) | 0.74 (0.02) | 0.75 (0.02) | 0.75 (0.02) | 0.76 (0.02) |

Fig. 9: Predictive performance comparison of an ensemble of 3 Simple CRMs against some leading black-box predictors. The numbers are estimates of predictive accuracy obtained on test data. The number in parentheses is the estimated standard deviation. The techniques being compared are: GNN, the graph-based neural network approach described in (Dash et al., 2022); DRM (500), is a form of MLP called a Deep Relational Machine that has, as input, Boolean feature-vectors resulting from a stochastic selection of 500 relational features (Dash et al., 2019); and CILP++, an MLP that has input Boolean feature-vectors resulting from an exhaustive feature construction technique called Bottom-Clause Propositionalisation (França et al., 2014). Baseline is the majority class predictor.

The results in Figure 9 indicate that Simple CRMs perform approximately as well as CILP++ (França et al., 2014), but are worse than either the GNN (Dash et al., 2022) or DRM (Dash et al., 2019). However, Figure 9 are best treated as preliminary. Variations in CRMs arise in Procedure 2 purely due to sampling, of course. However, the CRM obtained is also affected by the following: (1) The subset constraints on support and precision all feature-clauses in the CRM; (2) bounds on the depth of compositions $\rho_2$ followed by $\rho_1$ operators; (3) The number of feature-clauses drawn in each layer of the CRM. Additional variation can arise from the initialisation of weights for the SGD-based estimation of parameters. This suggests that substantially more experimentation is needed to see if the predictive performance of Simple CRMs can be improved. We note also that the DRM uses substantially more complex features than the Simple CRM, and that CILP++ constructs substantially more features than the Simple CRM (anywhere between 30,000 to 50,000 compared to about 330 M-simple features for the CRMs). Of
course none of GNN, DRM or CILP++ have any intrinsic mechanism of associating explanations with their prediction.

5 Related Work

We note first that $\rho_1$ and $\rho_2$ are closely related to the notion of refinement operators which have been studied extensively in ILP, in the context of the search through a hypothesis space (see (Tamaddoni-Nezhad and Muggleton, 2009; Nienhuys-Cheng et al., 1997)). Our motivation in this paper is, however, in the use of these operators to derive relational features. Consequently, we describe connections to related work in 3 categories: conceptual work on relational features; implementation and applied work on propositionalisation in ILP; and work on explainable deep networks.

On the conceptual understanding of relational features, perhaps the most relevant single summary is in (Saha et al., 2012). There the authors identify several feature-classes, based on somewhat similar notions of source- and sink-literals. The relationship between the different classes in that paper is shown in Figure 10(a). The relationship to these sets of the class of M-simple feature-clauses, denoted here as $F_M$, is shown in Figure 10(b) (see Appendix E for more details).

![Fig. 10: The feature classes proposed in (Saha et al., 2012): (a) and the relationship to the class of M-simple features (b).](image)

Simple clauses in (McCreath, 1999) and the corresponding set of features in $F_s$ are restricted to determinate predicate-definitions. Results in (McCreath, 1999) show that features from $F_s$ can be used to derive the subset of feature-clauses in $F_d$ that only contain determinate predicate-definitions. There is no such restriction imposed on $F_M$ and all clauses in $F_d$ (and therefore all other classes shown) can be derived using some composition of $\rho_1$ and $\rho_2$. No corresponding operators or completeness results are known for $F_r$.

Relational features have been shown to be an extremely effective form of learning with relational objects (Kramer et al., 2001; Lavrač et al., 2021). Methods that construct and use relational features, guided by some form of mode-declarations can be found in (Srinivasan and King, 1999; Lavrač et al., 2002; Ramakrishnan

---

13 Informally, a determinate predicate is one whose definition encodes a function. That is, for a given set of values for input arguments, there is exactly one set of values for output arguments.
Composition of Relational Features

et al., 2007; Joshi et al., 2008; Specia et al., 2009; Faruquie et al., 2012; Saha et al., 2012; França et al., 2014; Vig et al., 2017; Dash et al., 2018). Of these, the features in Lavrač et al. (2002) are from the feature class \( F_r \). There are no reports on the class of features used in the other reports, although the procedures for obtaining the features suggest that they are not restricted to any specific sub-class (that is, they are simply from \( F_d \)). Given our results on derivation of features in \( F_d \) from features in \( F_M \), and the class-inclusions shown, we would expect at least some features in a super-class would require additional composition operations to those in a sub-class. In terms of a CRM structure, we would expect features in \( F_i \), for example, would usually be associated with vertices at a greater depth than those in \( F_r \). Empirical results tabulated for some statistical learners in (Saha et al., 2012) suggest that relational features from the class \( F_i \) were most useful for statistical learners. If this empirical trend continues to hold, then we would expect the performance of CRMs to improve as depth increases (and features in \( F_i \) are derived), and then to flatten or decrease (as features in \( F_d \setminus F_i \) are derived).

The development and application of CRMs is most closely related to the area of self-explainable deep neural networks (Alvarez Melis and Jaakkola, 2018; Angelov and Soares, 2020; Ras et al., 2022). The structure of the CRM enforces a meaning to each node in the network, and in turn, we have shown here how to extract one form of explanation from these meanings. A different kind of neural network, also with meanings associated nodes is described in (Sourek et al., 2018). Those networks are also explainable, although not in the manner described here. In (Srinivasan et al., 2019), a symbolic proxy-explainer is constructed using ILP for a form of multi-layer perceptron (MLP) that uses as input values of relational features. The features there drawn from the class \( F_d \), and the explanations are logical rules constructed by ILP using the feature-definitions provided to the MLP. There are at least two important differences to the explanations constructed there and the ones obtained with a CRM: (i) The rules constructed in (Srinivasan et al., 2019) effectively only perform the \( \rho^2 \) operation on relational features. This can result in a form of incompleteness: some features in \( F_d \) cannot be represented by the rules, unless they are already included as input to the MLP; and (ii) The structuring of explanations in (Srinivasan et al., 2019) requires relevance information: here, the structuring is from usual functional (de)composition.

6 Conclusion

It has been long-understood in machine learning that the choice of representation can make a significant difference to the performance and efficiency of a machine learning technique. Representation is also clearly of relevance when we are interested in constructing human-understandable explanations for predictions made by the machine learning technique. A form of machine learning that has paid particular attention to issues of representation is the area of Inductive Logic Programming (ILP). A form of representation that has been of special interest in ILP is that of a relational feature. These are Boolean-valued functions defined over objects, using definitions of relations provided as prior- or background knowledge. The use of relational features forms the basis of an extremely effective form of ILP called propositionalisation. This obtains a Boolean-vector description of objects in the data, using the definition of the relational features and the background
knowledge. Despite the obvious successes of propositionalisation, surprising little is known, conceptually, about the space of relational features. In this paper, we have sought to address this by examining relational features within a mode-language $M$, introduced in ILP within the setting of mode-directed inverse entailment (Muggleton, 1995). Within a mode-language, we identify the notion of $M$-simple relational features, and two operations $\rho_1$ and $\rho_2$ that allows us to compose progressively more complex relational feature in the mode-language. In the first half of the paper, we show that $\rho_1$ and $\rho_2$ are sufficient to derive all relational features within a mode-language $M$. This generalises a previous result due to McCreath and Sharma (1998b), which was restricted to determinate definitions for predicates in the background knowledge, albeit starting from a different definition of simple features to that work.

In the second half of the paper, we use the notion of $M$-simple features and the composition operators $\rho_1$ and $\rho_2$ to construct a kind of deep neural network called a Compositional Relational Machine, or CRM. A special aspect of CRMs is that we are able to associate a relational feature with each node in the network. The structure of the CRM allows us to identify further how the feature at the node progressively decomposes into simpler features, until an underlying set of $M$-simple features are reached. This corresponds well to the intuitive notion of a structured explanation, that is composed of increasingly simpler components. We show how this aspect of CRMs allows them to be used as “explanation machines” for black-box models. Our results on synthetic and real-data suggest that CRMs can reproduce target-predictions with high fidelity; and the explanations constructed on synthetic data suggest that CRM’s explanatory structure usually also contains an acceptable explanation.

We have not explored the power of CRMs as “white-box” predictors in their own right, but early results suggest that it may be possible to obtain CRMs with good predictive accuracy. Although still significantly lower than the state-of-the-art, we believe this can change. We have also not explored other forms of CRMs, both simpler and more elaborate. For example, the identification of $M$-simple features and their subsequent compositions using the $\rho$-operators suggests an even simpler CRM structures than that used here. It is possible for example, simply to obtain all possible compositions to some depth, and use a Winnow-like parameter estimation (Littlestone, 1988) to obtain a self-explainable linear model. Equally, more complex CRMs are possible by incorporating weights on the $M$-simple features (this could be implemented simply by changing the activation function at the input nodes of the network). Taking this one step further, it is possible to associate weights with all the relational features, which will allow the use of the inference machinery of probabilistic logic programs (De Raedt et al., 2019). We think an investigation of these other kinds of compositional relational machines would contribute positively to the growing body of work in human-intelligible machine learning.

Acknowledgements AS is a Visiting Professor at Macquarie University, Sydney and a Visiting Professorial Fellow at UNSW, Sydney. He is also the Class of 1981 Chair Professor at BITS Pilani, Goa and a Research Associate at TCS Research. AS and TD would like to thank Lovekesh Vig and Gautam Shroff at TCS Research for interesting discussions on explainable neural networks; and Michael Bain at UNSW for discussions on the use of ILP for constructing symbolic explanations.
Declaration

Funding Not applicable.
Conflicts of interest Not applicable.
Ethics approval Not applicable.
Consent to participate Not applicable.
Consent for publication Not applicable.
Data and code availability All data and codes used in our research can be found at: https://github.com/tirtharajdash/CRM.

Authors’ contributions AS and AB conceived and worked on the conceptual parts related to simple features and their composition, and the specification of CRMs. TD conceived and worked on the implementation of CRMs as gated neural networks. AS and TD conceived and worked on the application of CRMs to synthetic- and real-data. DS was involved in some parts of the implementation of CRMs.
Appendices

A Logic Terminology

In this section we cover only terminology used in the paper, and further confined largely to logic programming. For additional background and further terminology see (Lloyd, 2012; Chang and Lee, 2014; Nilsson, 1991; Muggleton and de Raedt, 1994). The summary below is adapted from (Srinivasan et al., 2019).

A language of first order logic programs has a vocabulary of constants, variables, function symbols, predicate symbols, logical implication ‘←’, and punctuation symbols. A function or predicate can have a number of arguments known as terms. Terms are defined recursively. A constant symbol (or simply “constant”) is a term. A variable symbol (or simply “variable”) is a term. If $f$ is an $m$-ary function symbol, and $t_1, \ldots, t_m$ are terms, then the function $f(t_1, \ldots, t_m)$ is a term. A term is said to be ground if it contains no variables.

We use the convention used in logic programming when writing clauses. Thus, predicate, function and constant symbols are written as a lower-case letter followed by a string of lower- or upper-case letters, digits or underscores (’_’). Variables are written similarly, except that the first letter must be upper-case. This is different to the usual logical notation, where predicate-symbols start with upper-case, and variables start with lower-case: however the logic programming syntax is useful for the implementation of CRMs. Usually, predicate symbols will be denoted by symbols like $p, q, r$, etc., and symbols like $X, Y, Z$ to denote variables. If $p$ is an $n$-ary predicate symbol, and $t_1, \ldots, t_n$ are terms, then the predicate $p(t_1, \ldots, t_n)$ is an atom. Predicates with the same predicate symbol but different arities are distinguished by the notation $p/n$ where $p$ is a predicate of arity $n$.

A literal is either an atom or the negation of an atom. If a literal is an atom it is referred to as a positive literal, otherwise it is a negative literal. A clause is a disjunction of the form $A_1 \lor \ldots \lor A_i \lor \neg A_{i+1} \lor \ldots \lor \neg A_k$, where each $A_j$ is an atom. Alternatively, such a clause may be represented as an implication (or “rule”) $A_1, \ldots, A_i \leftarrow A_{i+1}, \ldots, A_k$. A definite clause $A_1 \leftarrow A_2, \ldots, A_k$ has exactly one positive literal, called the head of the clause, with the literals $A_2, \ldots, A_k$ known as the body of the clause. A definite clause with a single literal is called a unit clause, and a clause with at most one positive literal is called a Horn clause. A set of Horn clauses is referred to as a logic program. It is often useful to represent a clause as a set of literals.

A substitution $\theta$ is a finite set \{ $v_i/t_1, \ldots, v_n/t_n$ \} mapping a set of $n$ distinct variables $v_i$, $1 \leq i \leq n$, to terms $t_j$, $1 \leq j \leq n$ such that no term is identical to any of the variables. A substitution containing only ground terms is a ground substitution. For substitution $\theta$ and clause $C$ the expression $C\theta$ denotes the clause where every occurrence $C$ of a variable from $\theta$ is replaced by the corresponding term from $\theta$. If $\theta$ is a ground substitution then $C\theta$ is called a ground clause. Since a clause is a set, for two clauses $C, D$, the set inclusion $C\theta \subseteq D$ is a partial order called subsumption, usually written $C \theta$-subsumes $D$ and denoted by $C \preceq_D D$. For a set of clauses $S$ and the subsumption ordering $\preceq$, we have that for every pair of clauses $C, D \in S$, there is a least upper bound and greatest lower bound, called, respectively, the least general generalisation (lgg) and most general unifier (mgu).
of $C$ and $D$, which are unique up to variable renaming. The subsumption partial ordering on clauses enables the definition of a lattice, called the subsumption lattice.

We assume the logic contains axioms allowing for inference using the equality predicate $=$. This includes axioms for reflexivity $(\forall x (x = x))$, and substitution $(\forall x (\phi(x) \land (x = y) \rightarrow \phi(y)))$.

B Mode Language

We borrow some of the following definitions from (Dash et al., 2022). The definition of $\lambda\mu$ sequence is simplified as we are dealing only with feature clauses here and all other definitions are same as in (Dash et al., 2022).

**Definition 17 (Term Place-Numbering)** Let $\pi = \langle i_1, \ldots, i_k \rangle$ be a sequence of natural numbers. We say that a term $\tau$ is in place-number $\pi$ of a literal $\lambda$ iff: (1) $\pi \neq \langle \rangle$; and (2) $\tau$ is the term at place-number $\langle i_2, \ldots, i_k \rangle$ in the term at the $i_1$th argument of $\lambda$. $\tau$ is at a place-number $\pi$ in term $\tau'$: (1) if $\pi = \langle \rangle$ then $\tau = \tau'$; and (2) if $\pi = \langle i_1, \ldots, i_k \rangle$ then $\tau'$ is a term of the form $f(t_1, \ldots, t_m)$, $i_1 \leq m$ and $\tau$ is in place-number $\langle i_2, \ldots, i_k \rangle$ in $t_{i_1}$.

**Definition 18 (Type-Names and Type-Definitions)** Let $\Gamma$ be a set of types and $T$ be a set of ground-terms. For $\gamma \in \Gamma$ we define a set of ground-terms $T_\gamma = \{\tau_1, \tau_2, \ldots\}$, where $\tau_i \in T$. We will say a ground-term $\tau_i$ is of type $\gamma$ if $\tau_i \in T_\gamma$, and denote by $T_\Gamma$ the set $\{T_\gamma : \gamma \in \Gamma\}$. $T_\Gamma$ will be called a set of type-definitions.

**Definition 19 (Mode-Declaration)**

(a) Let $\Gamma$ be a set of type names. A mode-term is defined recursively as one of:
(i) $+\gamma$, $-\gamma$ or $\#\gamma$ for some $\gamma \in \Gamma$; or (ii) $\phi(mt_1, mt_2, \ldots, mt_j)$, where $\phi$ is a function symbol of arity $j$, and the $mt_k$s are mode-terms.
(b) A mode-declaration $\mu$ is of the form $\text{modeh}(\lambda')$ or $\text{modeb}(\lambda')$. Here $\lambda'$ is a ground-literal of the form $p(mt_1, mt_2, \ldots, mt_n)$ where $p$ is a predicate name with arity $n$, and the $mt_k$ are mode-terms. We will say $\mu$ is a modeh-declaration (resp. modeb-declaration) for the predicate-symbol $p/n$. In general there can be several modeh or modeb-declarations for a predicate-symbol $p/n$. We will use $\text{ModeLit}(\mu)$ to denote $\lambda'$.
(c) $\mu$ is said to be a mode-declaration for a literal $\lambda$ iff $\lambda$ and $\text{ModeLit}(\mu)$ have the same predicate symbol and arity.
(d) Let $\tau$ be the term at place-number $\pi$ in $\mu$, We define

$$\text{ModeType}(\mu, \pi) = \begin{cases} 
(+, \gamma) & \text{if } \tau = +\gamma \\
(-, \gamma) & \text{if } \tau = -\gamma \\
(\#, \gamma) & \text{if } \tau = \#\gamma \\
\text{unknown} & \text{otherwise}
\end{cases}$$

(e) If $\mu$ is a mode-declaration for literal $\lambda$, $\text{ModeType}(\mu, \pi) = (+, \gamma)$ for some place-number $\pi$, $\tau$ is the term at place $\pi$ in $\lambda$, then we will say $\tau$ is an input-term of type $\gamma$ in $\lambda$ given $\mu$ (or simply $\tau$ is an input-term of type $\gamma$). Similarly we define output-terms and constant-terms.
We will also say that mode $\mu$ contains an input argument of type $\gamma$ if there exists some term-place $\pi$ of $\mu$ s.t. $\text{ModeType}(\mu, \pi) = (+, \gamma)$. Similarly for output arguments.

**Definition 20 (\(\lambda\mu\)-Sequence)** Assume a set of type-definitions $T$, modes $M$. Let $\langle C \rangle = \langle \ell_1, \ell_2, \ell_3, \ldots, \ell_k \rangle$ be an ordered clause. Then $\langle (\lambda_1, \mu_1), (\lambda_2, \mu_2), \ldots, (\lambda_k, \mu_k) \rangle$ is said to be a $\lambda\mu$-sequence for $\langle C \rangle$ iff it satisfies the following constraints:

- **Match.** (i) $\lambda_1 = \ell_1$; (ii) For $j = 1, \ldots, k$, $\mu_j$ is a mode-declaration for $\lambda_j$ s.t. $\mu_j = \text{mode}(\cdot) \ (j = 1)$ and $\mu_j = \text{mode}(\cdot) \ (j > 1)$.

- **Terms.** (i) If $\tau$ is an input- or output-term in $\lambda_j$ given $\mu_j$, then $\tau$ is a variable in $\lambda_j$; (ii) Otherwise if $\tau$ is a constant-term in $\lambda_j$ given $\mu_j$ then $\tau$ is a ground term.

- **Types.** (i) If there is a variable $v$ in both $\lambda_i$, $\lambda_j$ then the type of $v$ in $\lambda_i$ given $\mu_i$ is the same as the type of $v$ in $\lambda_j$ given $\mu_j$; (ii) If $\tau$ is a constant-term in $\lambda_i$ and the type of $\tau$ in $\lambda_i$ given $\mu_i$ is $\gamma$, then $\tau \in T_\gamma$.

- **Ordering.** (i) If $\tau$ is an input-term in $\lambda_j$ given $\mu_j$ and $j > 1$ then there is an input-term $\tau_i$ in $\lambda_1$ given $\mu_1$; or there is an output-term $\tau$ in $\lambda_i$ ($m < i < j$) given $\mu_i$. (ii) If $\tau$ is an output-term in $\lambda_1$ given $\mu_1$, then $\tau$ is an output-term of some $\lambda_i \ (1 < i \leq k)$ given $\mu_i$.

**Definition 21 (Mode-Language)** Assume a set of type-definitions $T$ and modes $M$. The mode-language $\mathcal{L}_{M,T}$ is $\{ \langle C \rangle : \text{either } C = \emptyset \text{ or there exists a } \lambda\mu\text{-sequence for } \langle C \rangle \}$.

**C Proof of the Derivation Lemma**

**Lemma 2 (Derivation Lemma)** Given $M, M'$ and $\Omega = \{ \rho_1, \rho_2 \}$ as before. Let $\langle C \rangle$ be an ordered clause in $\mathcal{L}_{M,T}$, with head $p(X)$. Let $S$ be a set of ordered M-simple clauses in $\mathcal{L}_{M,T}$, with heads $p(X)$ and all other variables of clauses in $S$ standardised apart from each other and from $C$. If there exists a substitution $\theta$ s.t. $\text{Basis}\langle C \rangle \subseteq S \theta$ then there exists an ordered clause $\langle C' \rangle$ in $\mathcal{L}_{M,T}$ such that $\langle C' \rangle$ is equivalent to $\langle C \rangle$ and derivable from $S$ using $\Omega$.

**Proof** We prove this in 3 parts:

1. We first show: if $\text{Basis}\langle C \rangle \subseteq S \theta$, then $\theta$ is a type-consistent substitution for clauses in $S$.

   Let $\text{Basis}\langle C \rangle = \{ \langle C_1 \rangle, \ldots, \langle C_n \rangle \}$ and $S' = \{ S_1, \ldots, S_n \} \subseteq S$. Without loss of generality, let $S_j \theta = C_i$. Let $\theta = \{ Y_1/t_1, Y_2/t_2, \ldots, Y_n/t_n \}$ where the $Y_i$ are variables in $S$ and the $t_i$ are terms in $\text{Basis}\langle C \rangle$. In general, the $t_i$’s are variables, constants, or functional terms. Let $t_j$ be a constant.

   Suppose clause $S_i$ has the variable $Y_j$ in some location in $S_i$ and clause $C_i$ has the constant $t_j$ in the corresponding location. Since both $S_i$ and $C_i$ are in $\mathcal{L}_{M,T}$, this is only possible if there are multiple mode-declarations for the corresponding literals in $S_i$ and $C_i$. But constraint MC2 (in Section 2.2) ensures that there is exactly one mode-declaration for any literal. Therefore $t_i$ cannot be a constant. Reasoning similarly, $t_j$ cannot be a functional term. Therefore $t_i$ has to be a variable in $C_i$. Since $Y_j$ and $t_j$ are in the
same locations for some literal \( l \) in \( S_i \) and \( C_i \), and there is only one mode-declaration for \( l \) in \( M \), it follows that the types of \( Y_j \) and \( t_j \) must be the same.

2. If there exists a type-consistent substitution \( \theta \) for clauses in \( S \), then there is an ordered clause \( \langle C'' \rangle \) in \( \text{Closure}_{\rho_2}(S) \) s.t. \( C'' \theta \equiv C \).

   Let us fix an ordering among the simple clauses in \( S \): \( S_1, \ldots, S_k \), where \( S_i = p(X) \leftarrow \text{Body}_i(X,Y_1) \). Let us define a sequence of ordered clauses \( \langle C_1, C_2, \ldots, C_k \rangle \) where \( \langle C_1 \rangle \) is \( \langle S_1 \rangle \), and \( \langle C_i \rangle \) is in \( \rho_2((C_{i-1}), (S_i)) \) for \( i = 2 \to k \). It is easy to see that the above sequence is a derivation sequence of \( \langle C_k \rangle \) from \( \langle S_1, S_2, \ldots, S_k \rangle \) using \( \rho_2 \). That is, \( \langle C_k \rangle \) is in \( \text{Closure}_{\rho_2}(S) \).

   \( \langle C_k \rangle \) is of the form \( p(X) \leftarrow \text{Body}_1(X,Y_1), \text{Body}_2(X,Y_2), \ldots, \text{Body}_k(X,Y_k) \). That is, \( C_k = \bigcup_{i=1}^{k} S_i \). Since \( S \theta = \text{Basis}(C_k) \), then \( C = \bigcup_{i=1}^{k} S_i \theta \). Let \( \langle C'' \rangle = \langle C_k \rangle \). That is, \( C'' \theta = C \). Hence \( C'' \theta \equiv C \). Also, since \( \theta \) is a type-consistent substitution for clauses \( S_i \) in \( S \), \( C'' = \bigcup_{i=1}^{k} S_i \theta \) is a type-consistent substitution for \( C'' \).

3. From above \( \langle C'' \rangle \) and \( \theta \), we construct an ordered clause \( \langle C' \rangle \) in \( \mathcal{L}_{M',T} \) s.t. \( \langle C' \rangle \) is in \( \text{Closure}_{\rho_1}(\{\langle C'' \rangle\}) \) and \( C'' \theta \equiv C' \).

   Let \( \langle C'' \rangle = \{p(X) \leftarrow \text{Body}(X,Y)\} \), and \( \theta = \{Y_1/Y_1', \ldots, Y_m/Y_m'\} \). It is easy to see that \( \theta \) induces an equivalence relation on the variables in \( C'' \). Variables \( Y_i \) and \( Y_j \) are in the same equivalence class if both variables map to the same variable \( Y_m' \). For each equivalence class \([U]\), we earmark one element in the class as representative of the class. This element is denoted by \( \text{rep}(U) \). Let \( \langle C' \rangle = \{p(X) \leftarrow \text{Body}(X,Y)\} \). Let us fix an order among the variables occurring in \( \langle C'' \rangle \): \( Y_1', \ldots, Y_l' \). Consider the following sequence of ordered clauses \( \langle C_0 \rangle, \langle C_1 \rangle, \ldots, \langle C_l \rangle \) where \( \langle C_0 \rangle = \langle C'' \rangle \), and \( \langle C_l \rangle = p(X) \leftarrow \text{Body}(X,Y), Y_1 = \text{rep}(Y_1), \ldots, Y_l = \text{rep}(Y_l) \). The above sequence is a derivation of \( \langle C_l \rangle \) from \( \langle C'' \rangle \) using \( \rho_1 \). That is, \( \langle C_l \rangle \) is in \( \text{Closure}_{\rho_1}(\{\langle C'' \rangle\}) \). Let \( \langle C' \rangle = \langle C_l \rangle \) and it is of the form \( p(X) \leftarrow \text{Body}(X,Y), Y_1 = \text{rep}(Y_1), \ldots, Y_l = \text{rep}(Y_l) \). It is easy to see that \( \langle C' \rangle \in \mathcal{L}_{M',T} \). We now show \( C'' \theta \equiv C' \). Let \( \theta' = \theta'' \circ \theta' \), where \( \theta'' = \{Y_1/\text{rep}(Y_1), \ldots, Y_l/\text{rep}(Y_l)\} \) and \( \theta' = \{\text{rep}(Y_1)/Y_1', \ldots, \text{rep}(Y_l)/Y_l'\} \). The variables \( \text{rep}(Y_i) \) are in \( S \) and the variables \( Y_i' \) are in \( C \). Since the variables in \( S \) are standardised apart from the variables in \( C \), \( \theta' \) is a renaming substitution. Therefore \( C'' \theta'' \equiv C'' \). Therefore \( C'' \theta \equiv C'' \theta'' \). By the substitution axiom in the equality logic \( C'' \theta'' \equiv C' \). Therefore \( C'' \theta \equiv C' \).

From (1)–(3) above, the result follows.

D Properties of Derivations using \( \rho_1, \rho_2 \)

**Lemma 4 (Reordering Lemma)** Given \( M, M' \) and a set of ordered clauses \( \Phi \). If an ordered clause \( \langle C \rangle \) is derivable from \( \Phi \) using \( \{\rho_1, \rho_2\} \) then: (a) there is a derivation

---

14 Correctly, \( \theta' \) and \( \theta'' \) have to be functions. It is evident that \( \theta'' \) is a function, since each variable maps to the representative of its equivalence class. We argue informally that \( \theta' \) is a function as follows. If \( Y_i \) and \( Y_j \) are in the same equivalence class, then \( \text{rep}(Y_i) \) and \( \text{rep}(Y_j) \) are the same variables. But according to our \( \theta' \), they are mapped to \( Y_i' \) and \( Y_j' \). One might think \( \theta' \) is not a function. But the variables in the same equivalence class are mapped to same variable. So \( Y_i' \) is the same as \( Y_j' \) and hence \( \theta' \) is a function.
for \( (C_j) \) from \( \Phi \) using \( \{\rho_2\} \); (b) there is a derivation for \( (C_k) \) from \( \{(C_j)\} \) using \( \{\rho_1\} \); and (c) \( (C) \) is equivalent to \( (C_k) \).

**Proof** Let us assume there is a derivation \( \tau \) for \( (C) \) from \( \Phi \) using \( \{\rho_1, \rho_2\} \). There exists an ordered clause \( (C_k) \) equivalent to \( (C) = p(X) \leftarrow l_1, l_2, \ldots, l_j, l_{j+1}, l_{j+2}, \ldots, l_k \) such that

- the sequence of literals \( l_1, \ldots, l_j \) can be split into \( \text{Body}(X,Y_1), \ldots, \text{Body}(X,Y_m) \) such that \( p(X) \leftarrow \text{Body}(X,Y_i) \in \Phi \) for each \( i = 1, \ldots, m \), and
- \( l_{j+1}, \ldots, l_k \) are equality predicates introduced using \( \rho_1 \) operator in the derivation \( \tau \). Let \( (C_j) \) be \( p(X) \leftarrow \text{Body}(X,Y_1), \ldots, \text{Body}(X,Y_m) \).

It is easy to see that there is derivation \( \tau_1 \) for \( (C_j) \) from \( \Phi \) using \( \rho_2 \) operator. Now we can apply \( \rho_1 \) operator on \( (C_j) \) repeatedly to derive \( (C_k) \).

\[ \blacksquare \]

**Lemma 5** Given \( M, M' \) and a set of ordered clauses \( \Phi \). If an ordered clause \( (C) \) is derivable from \( \Phi \) using \( \{\rho_2\} \) then it is linearly derivable from \( \Phi \) using \( \{\rho_2\} \).

**Proof** Let us assume that \( (C) \) is derivable from \( \Phi \) using \( \{\rho_2\} \). Hence there exists a derivation sequence \( \xi \) of ordered clauses \( (C_1), (C_2), \ldots, (C_n) \) from \( \Phi \) using \( \{\rho_2\} \) and \( (C_n) = (C) \). Now we use mathematical induction to prove that for each \( 1 \leq i \leq n \), there exists a linear derivation of \( (C_i) \) from \( \Phi \) using \( \{\rho_2\} \).

**Base step:** Case \( n = 1 \) is easy as \( (C_1) \) should be in \( \Phi \) and hence there is a linear derivation for \( (C_1) \) from \( \Phi \) using \( \{\rho_2\} \).

**Induction step:** We assume there exists a linear derivation for \( (C_i) \) from \( \Phi \) using \( \{\rho_2\} \) for each \( i < j \) (induction hypothesis) and prove that there exists a linear derivation for \( (C_j) \). The clause \( (C_j) \) occurs in the derivation sequence \( \xi \). There are two cases to be considered: it is in \( \Phi \) or it is in \( \rho_2((C_l), (C_m)) \) where \( l, m < j \).

If \( (C_j) \) is in \( \Phi \), there exists a linear derivation sequence for \( (C_j) \) from \( \Phi \) (just the one step derivation containing itself). Now we prove the claim for the second case. Since \( l, m < j \), by induction hypothesis, there exist linear derivation sequences \( \xi_l \) for \( (C_l) \) and \( \xi_m \) for \( (C_m) \) from \( \Phi \) using \( \{\rho_2\} \). Let \( (C'_1), (C'_2), \ldots, (C'_k) \) be the clauses in \( \Phi \) and occurring in \( \xi_m \). Now consider the following sequence \( \xi_j \) containing \( \xi_l, (C'_1), (C'_2), (C'_3), \ldots, (C'_k), (C''_1), (C''_2), \ldots, (C''_a) \) where \( (C''_a) \in \rho_2((C_l), (C'_1)) \) and \( (C''_a) \in \rho_2((C''_{a-1}), (C''_a)) \) for \( a = 2 \) to \( k \). It is easy to see that \( \xi_j \) is a linear derivation sequence for \( (C'_j) \) from \( \Phi \) using \( \rho_2 \). Hence there exists a linear derivation sequence for \( (C_n) \) from \( \Phi \) using \( \{\rho_2\} \).

\[ \blacksquare \]

Now the linear derivation lemma (re-stated below) can be easily proved by combining the above two lemmas.

**Lemma 3 (Linear Derivation Lemma)** Given \( M, M' \) and a set of ordered clauses \( \Phi \). If an ordered clause \( (C) \) is derivable from \( \Phi \) using \( \{\rho_1, \rho_2\} \) then there exists an equivalent ordered clause \( (C') \) and it is linearly derivable from \( \Phi \) using \( \{\rho_1, \rho_2\} \).

**E Relationship of M-Simple Feature-Clauses to Known Feature Classes**

We note the following about the sets of feature-clauses in Figure 10(b) and the set \( F_M \) of M-simple feature clauses:
The set $F_d$. This is identical to the set of ordered feature-clauses in the mode-language $M$.

The set $F_i$. This set is defined based on a given mode language $M$ (but without constraints MC–MC3). An ordered feature clause $\langle C \rangle$ is in $F_i$ iff the number of connected components after removing the source vertex (corresponding to the head literal) from the clause dependency graph of $\langle C \rangle$ is exactly one. Since the clause dependency graph of a clause in $F_M$ has exactly one sink literal and any vertex in that graph has a path to that sink vertex (Remark 2). So the number of connected components after removing the source vertex is one. Hence $F_M \subseteq F_i$. It is easy to see that $F_i \not\subseteq F_M$ (see the counterexample given for $F_s \subseteq F_M$).

The set $F_s$. This set is defined based on the class of simple clauses identified by McCreath (1999) who proposed simple clauses. We note:

1. Feature-clause definitions in $F_s$ do not refer to a mode-language. The clause dependency-graph is constructed using a procedure described in (McCreath, 1999), and is based on the re-occurrence of variables (without any reference to input or output variables or types). M-simple feature-clauses require a mode-language, with the constraints MC1–MC3;

2. $F_M \not\subseteq F_s$. For example assume a mode language $p(+int), q(+int, -int), r(+int)$ and the feature-clause $p(X) \leftarrow q(X, X), r(X)$. The clause dependency-graph of this clause has only one sink vertex and so it is in $F_M$. But this is not in $F_s$ since the directed graph associated with this clause has two sink literals.

3. $F_s \not\subseteq F_M$. For example assume a mode language $p(+int), q(+int, -int), r(+int, -int)$ and the feature-clause $p(X) \leftarrow q(X, Y), r(X, Y)$. This is in $F_s$, but is not in $F_M$. The cause dependency-graph described in this paper doesn’t have an edge between the vertices for $q$ and $r$ resulting in two sink vertices. But the dependency graph constructed in (McCreath, 1999) will have an edge between the vertices for $q$ and $r$ because of the shared variable $y$.

The set $F_r$. The set $F_r$ consist of feature-clauses designed for subgroup discovery in relational data (Lavrač et al., 2002). Then:

1. Feature-clauses in $F_r$ do require a mode-language, and we can construct a clause dependency-graph as described here. The clause dependency-graph for feature-clauses in $F_r$ have exactly one component, and all new existential variables introduced by a source literal appear in source or sink literals.

2. $F_M \not\subseteq F_r$. For example assume a mode language $p(+int), q(+int, -int)$ and the clause $p(X) \leftarrow q(X, Y)$. This is a M-simple clause but not in $F_r$ as the existential variable introduced at the source literal $q$ but it is not appearing later.

3. $F_r \not\subseteq F_M$. For example assume a mode language $p(+train), has\_car(+train, -\_car), short(+\_car), closed(+\_car)$ and the clause $p(X) \leftarrow has\_car(X, Y), short(Y), closed(Y)$. This is not M-simple clause there are two sink literals in it but it is in $F_r$. 

F Additional Details Relevant to the Experiments

F.1 Examples of Mode Declarations and Simple Feature-Clauses

Figure 11–Figure 13 show examples of mode declarations used for the experiments. Also shown are examples of M-simple feature-clauses constructed automatically from the mode-declarations.

| Modes                  | M-simple feature clauses                        |
|------------------------|-------------------------------------------------|
| modeh(p(+tr))          | p(A) ← has_car(A, B)                            |
| modeb(short(+car))     | p(A) ← has_car(A, B), short(B)                  |
| modeb(closed(+car))    | p(A) ← has_car(A, B), closed(B)                 |
| modeb(long(+car))      | p(A) ← has_car(A, B), long(B)                   |
| modeb(open_car(+car))  | p(A) ← has_car(A, B), open_car(B)               |
| modeb(double(+car))    | p(A) ← has_car(A, B), double(B)                 |
| modeb(jagged(+car))    | p(A) ← has_car(A, B), jagged(B)                 |
| modeb(shape(+car,#shape)) | p(A) ← has_car(A, B), shape(B, u_shaped)      |
| modeb(load(+car,#shape,#int)) | p(A) ← has_car(A, B), shape(B, rectangle)    |
| modeb(wheels(+car,#int)) | p(A) ← has_car(A, B), shape(B, hexagon)       |
| modeb(has_car(+tr,-car)) | p(A) ← has_car(A, B), wheels(B, 2)            |
|                        | p(A) ← has_car(A, B), load(B, circle, 3)       |
|                        | p(A) ← has_car(A, B), load(B, rectangle, 3)    |
|                        | p(A) ← has_car(A, B), load(B, hexagon, 3)      |
|                        | p(A) ← has_car(A, B), load(B, circle, 2)       |

Fig. 11: Examples of mode-definitions and simple feature-clauses for the Trains problem.

F.2 Debugging Inconsistent Explanations from the CRM

For the Chess problem, 1044 instances (out of 10,000) are inconsistently explained. That is, the explanation graph from the most relevant output vertex does not contain an acceptable feature-clause. Further examination reveals: (a) for all 1044 instances, the predictions made by the CRM are correct; (b) a majority (1033/1044) of the inconsistently instances are + examples for which the White King and Black King are on adjacent files and ranks (the corresponding acceptable feature-clause is p((A, B, C, D, E, F)) ← adj(A, E), adj(B, F)). We find there are 19 distinct ‘buggy explanations’ produced by the CRM for the inconsistently explained data. The corresponding feature-clauses at the ‘root’ in the explanation graph are listed in Figure 14 (for simplicity, we do not show the simpler features and the full graph structure):
Fig. 12: Examples of mode-definitions and simple feature-clauses for the Chess problem.
Fig. 13: Examples of mode-definitions and simple feature-clauses for the NCI problem.

| Modes | M-simple feature clauses |
|-------|--------------------------|
| modeb(class(+mol,+class)) | p(A) ← has_struc(A, B, C, amine), gteq(C, 1) |
| modeb(symbond(+mol,+atomid,+atomid,#bondtype)) | p(A) ← has_struc(A, B, C, ammonium_ion), gteq(C, 1) |
| modeb(bond(+mol,-atomid1,-atomid2,#atomtype,#atomtype,#bondtype)) | p(A) ← has_struc(A, B, C, benzene_ring), gteq(C, 2) |
| modeb(atom(+mol,-atomid,#element)) | p(A) ← has_struc(A, B, C, benzene_ring), gteq(C, 3) |
| modeb(has_struc(+mol,-atomids,-length,#structype)) | p(A) ← has_struc(A, B, C, benzene_ring), gteq(C, 4) |
| modeb(connected(+mol,+atomids,+atomids)) | p(A) ← has_struc(A, B, C, benzene_ring), gteq(C, 5) |
| modeb(fused(+mol,+atomids,+atomids)) | p(A) ← has_struc(A, B, C, benzene_ring), gteq(C, 6) |
| modeb(gteq(+length,#length)) | p(A) ← has_struc(A, B, C, conjug_base_car), gteq(C, 1) |
| modeb(lteq(+length,#length)) | p(A) ← has_struc(A, B, C, conjug_base_car), gteq(C, 2) |
| modeb((+atomid = +atomid)) | p(A) ← has_struc(A, B, C, conjug_base_car), gteq(C, 3) |
| modeb((+atomidids = +atomidids)) | p(A) ← has_struc(A, B, C, diazo_group), gteq(C, 1) |
| modeb((+length = +length)) | p(A) ← has_struc(A, B, C, diazo_group), gteq(C, 2) |

...
1. \( p((A, B, C, D, E, F)) \leftarrow \text{adj}(B, F), \text{lt}(D, B), \text{adj}(C, E) \)
2. \( p((A, B, C, D, E, F)) \leftarrow \text{adj}(D, F), B = D, \text{lt}(A, E) \)
3. \( p((A, B, C, D, E, F)) \leftarrow \text{adj}(A, E), \text{lt}(A, C), \text{lt}(F, D) \)
4. \( p((A, B, C, D, E, F)) \leftarrow \text{lt}(E, C), A = C, \text{lt}(B, F) \)
5. \( p((A, B, C, D, E, F)) \leftarrow \text{lt}(E, A), A = C, \text{adj}(A, C) \)
6. \( p((A, B, C, D, E, F)) \leftarrow \text{lt}(E, A), \text{lt}(D, B), \text{adj}(A, E) \)
7. \( p((A, B, C, D, E, F)) \leftarrow B = D, A = C, \text{adj}(A, E) \)
8. \( p((A, B, C, D, E, F)) \leftarrow B = F, A = E, \text{lt}(E, C) \)
9. \( p((A, B, C, D, E, F)) \leftarrow \text{lt}(A, C), \text{lt}(B, D), \text{adj}(A, C) \)
10. \( p((A, B, C, D, E, F)) \leftarrow \text{lt}(D, F), A = E, \text{lt}(E, C) \)
11. \( p((A, B, C, D, E, F)) \leftarrow A = E, \text{adj}(D, F), \text{adj}(B, F) \)
12. \( p((A, B, C, D, E, F)) \leftarrow \text{lt}(B, F), \text{lt}(E, C), \text{adj}(D, F) \)
13. \( p((A, B, C, D, E, F)) \leftarrow \text{adj}(B, D), \text{lt}(E, A), \text{adj}(A, C) \)
14. \( p((A, B, C, D, E, F)) \leftarrow \text{adj}(A, E), \text{lt}(C, E), \text{adj}(B, F) \)
15. \( p((A, B, C, D, E, F)) \leftarrow A = E, A = C, \text{adj}(B, F) \)
16. \( p((A, B, C, D, E, F)) \leftarrow \text{lt}(A, C), \text{lt}(F, B), \text{adj}(B, F) \)
17. \( p((A, B, C, D, E, F)) \leftarrow \text{lt}(D, B), \text{lt}(A, E), \text{adj}(A, E) \)
18. \( p((A, B, C, D, E, F)) \leftarrow \text{lt}(E, A), A = C, \text{adj}(B, D) \)
19. \( p((A, B, C, D, E, F)) \leftarrow \text{adj}(A, E), B = F, \text{lt}(A, E) \)

Fig. 14: Buggy explanations produced by the CRM for the inconsistently explained data instances from the Chess problem.
References

Alvarez Melis D, Jaakkola T (2018) Towards robust interpretability with self-explaining neural networks. Advances in neural information processing systems 31

Amarel S (1968) On representations of problems of reasoning about actions. In: Michie D (ed) Machine Intelligence 3, American Elsevier Publisher, pp 131–171

Angelov P, Soares E (2020) Towards explainable deep neural networks (xdnn). Neural Networks 130:185–194

Bach S, Binder A, Montavon G, Klauschen F, Müller KR, Samek W (2015) On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PloS one 10(7):e0130140

Bain M (1994) Learning logical exceptions in chess. PhD thesis, University of Strathclyde

Binder A, Bach S, Montavon G, Müller KR, Samek W (2016) Layer-wise relevance propagation for deep neural network architectures. In: Information science and applications (ICISA) 2016, Springer, pp 913–922

Chang CL, Lee RCT (2014) Symbolic logic and mechanical theorem proving. Academic press

Cybenko G (1989) Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems 2(4):303–314, DOI 10.1007/BF02551274, URL https://doi.org/10.1007/BF02551274

Dash T, Srinivasan A, Vig L, Orhobor OI, King RD (2018) Large-scale assessment of deep relational machines. In: International Conference on Inductive Logic Programming, Springer, pp 22–37

Dash T, Srinivasan A, Joshi RS, Baskar A (2019) Discrete stochastic search and its application to feature-selection for deep relational machines. In: International Conference on Artificial Neural Networks, Springer, pp 29–45

Dash T, Srinivasan A, Vig L (2021) Incorporating symbolic domain knowledge into graph neural networks. Machine Learning 110(7):1609–1636

Dash T, Srinivasan A, Baskar A (2022) Inclusion of domain-knowledge into gans using mode-directed inverse entailment. Machine Learning 111(2):575–623

De Raedt L, Manhaeve R, Dumancic S, Demeester T, Kimmig A (2019) Neuro-symbolic= neural+ logical+ probabilistic. In: NeSy’19@ IJCAI, the 14th International Workshop on Neural-Symbolic Learning and Reasoning, pp 1–4

Ellis K, Wong C, Nye M, Sablé-Meyer M, Morales L, Hewitt L, Cary L, Solar-Lezama A, Tenenbaum JB (2021) Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, pp 835–850

Faruquie TA, Srinivasan A, King RD (2012) Topic models with relational features for drug design. In: International conference on inductive logic programming, Springer, pp 45–57

França MV, Zaverucha G, Garcez ASd (2014) Fast relational learning using bottom clause propositionalization with artificial neural networks. Machine learning 94(1):81–104

Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT press

Gottlob G (1987) Subsumption and implication. Inf Process Lett 24(2):109–111, DOI 10.1016/0020-0190(87)90103-7, URL https://doi.org/10.1016/
Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Networks 2(5):359–366, DOI https://doi.org/10.1016/0893-6080(89)90020-8, URL https://www.sciencedirect.com/science/article/pii/0893608089900208

Joshi S, Ramakrishnan G, Srinivasan A (2008) Feature construction using theory-guided sampling and randomised search. In: International Conference on Inductive Logic Programming, Springer, pp 140–157

Kingma DP, Ba J (2015) Adam: A method for stochastic optimization. In: ICLR (Poster), URL http://arxiv.org/abs/1412.6980

Kramer S, Lavrač N, Flach P (2001) Propositionalization Approaches to Relational Data Mining. Springer Berlin Heidelberg, Berlin, Heidelberg, pp 262–291. DOI 10.1007/978-3-662-04599-2_11, URL https://doi.org/10.1007/978-3-662-04599-2_11

Lavrač N, Železný F, Flach PA (2002) Rsd: Relational subgroup discovery through first-order feature construction. In: International Conference on Inductive Logic Programming, Springer, pp 149–165

Lavrač N, Podpečan V, Robnik-Šikonja M (2021) Propositionalization of relational data. In: Representation Learning, Springer, pp 83–105

Littlestone N (1988) Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. Machine learning 2(4):285–318

Lloyd JW (2012) Foundations of logic programming. Springer Science & Business Media

Marx KA, O’Neil P, Hoffman P, Ujwal M (2003) Data mining the nci cancer cell line compound gi50 values: identifying quinone subtypes effective against melanoma and leukemia cell classes. Journal of chemical information and computer sciences 43(5):1652–1667

McCreath E (1999) Induction in first order logic from noisy training examples and fixed example set size. PhD thesis, The University of New South Wales

McCreath E, Sharma A (1998a) Lime: a system for learning relations. In: International Conference on Algorithmic Learning Theory, Springer, pp 336–374

McCreath E, Sharma A (1998b) Lime: A system for learning relations. In: Richter MM, Smith CH, Wiehagen R, Zeugmann T (eds) Algorithmic Learning Theory, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 336–374

Michalski RS (1980) Pattern recognition as rule-guided inductive inference. IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-2(4):349–361

Michie D (1976) King and Rook Against King. 1. Historical Background and a Problem on the Infinite Board. University of Edinburgh. Machine Intelligence Research Unit

Michie D, Muggleton S, Page D, Srinivasan A (1994) To the international computing community: A new east-west challenge. Distributed email document available from https://www.docicacuk/~shm/Papers/ml-challpdf

Muggleton S (1995) Inverse entailment and progol. New generation computing 13(3-4):245–286

Muggleton S, de Raedt L (1994) Inductive logic programming: Theory and methods. The Journal of Logic Programming 19-20:629–679, DOI https://doi.org/10.1016/0743-1066(94)90035-3, URL https://www.sciencedirect.com/science/article/pii/0743106694900353, special Issue: Ten Years of Logic Programming
Nienhuys-Cheng SH, De Wolf R, et al. (1997) Foundations of inductive logic pro-
gramming, vol 1228. Springer Science & Business Media
Nilsson NJ (1991) Logic and artificial intelligence. Artificial intelligence 47(1-3):31–
56
Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin
Z, Gimelshein N, Antiga L, et al. (2019) Pytorch: An imperative style, high-
performance deep learning library. In: Advances in Neural Information Process-
ning Systems, pp 8024–8035
Pinkus A (1999) Approximation theory of the mlp model in neural networks. Acta
Numerica 8:143–195, DOI 10.1017/S0962492900002919
Plotkin G (1972) Automatic methods of inductive inference. PhD Thesis, The
University of Edinburgh
Prechelt L (1998) Early stopping-but when? In: Neural Networks: Tricks of the
trade, Springer, pp 55–69
Quinlan JR (1979) Discovering rules by induction from large collections of exam-
ples. Expert systems in the micro electronics age
Ramakrishnan G, Joshi S, Balakrishnan S, Srinivasan A (2007) Using ilp to con-
struct features for information extraction from semi-structured text. In: Interna-
tional Conference on Inductive Logic Programming, Springer, pp 211–224
Ras G, Xie N, van Gerven M, Doran D (2022) Explainable deep learning: A field
guide for the uninitiated. Journal of Artificial Intelligence Research 73:329–397
Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-
propagating errors. nature 323(6088):533–536
Saha A, Srinivasan A, Ramakrishnan G (2012) What kinds of relational features
are useful for statistical learning? In: International Conference on Inductive
Logic Programming, Springer, pp 209–224
Sourek G, Aschenbrenner V, Zelezny F, Schockaert S, Kuzelka O (2018) Lifted rela-
tional neural networks: Efficient learning of latent relational structures. Journal
of Artificial Intelligence Research 62:69–100
Specia L, Srinivasan A, Joshi S, Ramakrishnan G, Nunes MdGV (2009) An inves-
tigation into feature construction to assist word sense disambiguation. Machine
Learning 76(1):109–136
Srinivasan A (2001) The aleph manual. https://www.cs.ox.ac.uk/activities/
programinduction/Aleph/aleph.html
Srinivasan A, King RD (1999) Feature construction with inductive logic pro-
gramming: A study of quantitative predictions of biological activity aided by struc-
tural attributes. Data Mining and Knowledge Discovery 3(1):37–57
Srinivasan A, Muggleton S, Bain M (1992) Distinguishing exceptions from noise
in non-monotonic learning. In: Proceedings of the 2nd International Workshop
on Inductive Logic Programming, Citeseer, pp 97–107
Srinivasan A, Vig L, Bain M (2019) Logical explanations for deep relational
machines using relevance information. Journal of Machine Learning Research
20(130):1–47
Tamaddoni-Nezhad A, Muggleton S (2009) The lattice structure and refinement
operators for the hypothesis space bounded by a bottom clause. Machine learn-
ing 76:37–72
Turing A (1948) Intelligent machinery. In: The Essential Turing, Oxford University
Press
Vig L, Srinivasan A, Bain M, Verma A (2017) An investigation into the role of domain-knowledge on the use of embeddings. In: Lachiche N, Vrain C (eds) Inductive Logic Programming - 27th International Conference, ILP 2017, Orléans, France, September 4-6, 2017, Revised Selected Papers, Springer, Lecture Notes in Computer Science, vol 10759, pp 169–183, DOI 10.1007/978-3-319-78090-0_12, URL https://doi.org/10.1007/978-3-319-78090-0_12