Skeptical Abduction: A Connectionist Network

Technische Universität Dresden

Luis Palacios Medinacelli

Supervisors:

Prof. Steffen Hölldobler
Emmanuelle-Anna Dietz

To my family.

Abstract

A key research area in Artificial Intelligence is human thought. Providing theoretical models for human reasoning is of great interest in AI as it sheds light on the ongoing processes in our minds and brains. Two fields studying these processes are Machine Learning and Knowledge Representation, whose approaches are usually incompatible. Thus a model that can take advantage of both fields is desirable. In this thesis we provide a Neural-Symbolic realization for computing Skeptical Abduction. The resulting network is a prerequisite to model some human reasoning tasks presented by Byrne.

Some of the afore mentioned human reasoning tasks were modeled in a cognitive-computational approach by Stenning and van Lambalgen. In particular they show that Byrne's tasks can be adequately modeled by logic programming. Their approach was modified by Hölldobler and Kencana Ramli using Łukasiewicz three-valued logic and weak completion semantics, and extended to a Neural-Symbolic representation. Their approach covered most of the tasks, but not all, since some of them require skeptical abduction. This thesis solves the problem of how to obtain a Neural-Symbolic system for computing skeptical abduction, and thus completing the work by the above mentioned previous approaches, which constitutes our main contribution.

Our approach uses abductive logic programming to formally specify skeptical abduction. Then we provide an algorithm to compute skeptical inferences, that is encoded as a logic program. Once the problem is solved at a logic level, we use the Core method to translate the resulting logic programs into a neural network. Both, the algorithm and the construction of the network heavily rely on the properties of semantic operators for logic programs.

As a result we obtain a neural network that adequately models Byrne's human reasoning tasks that require abduction. The network can also be used for solving additional problems related to abduction. It can as well be trained, meaning that the knowledge represented by the network can be learned.

Contents

Chapter 1

Introduction

Artificial intelligence studies, among other areas, how humans think and how we can build artificial systems with similar capabilities.

This involves how we obtain, encode, store, process, retrieve and use information. Since the main source of information about the world surrounding us are our senses, a lot of focus has been made on how we perceive and interact with the world through them: e.g. how does vision work? How do we perceive a smell? How do we feel cold or warm?

Considering a higher level of information processing, beyond our senses, we can think of several tasks: how do we make inferences? How do we learn things? How do we recognize objects? Furthermore, assume you witness an event, say you observe there is smoke on the horizon and you want to find out why. How do we find explanations like: there must have been a wildfire, or an accident, or maybe the fire was intentional to explain what we observe? How do we decide, from the immense knowledge available, which facts are relevant and which ones are useless for explaining that we see smoke on the horizon? The process of finding such explanations, given an observation is known in cognitive science and computer science as abduction.

In this thesis we explore how logic and artificial neural networks can be used together to model human reasoning. Specifically, we provide a connectionist model for skeptical abduction, where we want to obtain all possible consequences of all possible explanations, given an observed event.

The network is a prerequisite to model some human reasoning tasks. In [5] Byrne provided some empirical results that showed the non-monotonic nature of reasoning, and thus classical logic is not adequate to model them. In a cognitive-computational approach [16] Stenning and van Lambalgen discussed that under some considerations, on how we interpret information and how we process it, the different scenarios presented by Byrne could be adequately modeled by a

non-monotonic logic approach. Stenning and van Lambalgen's approach provided some specifications which were not fully accurate, corrected by Hölldobler and Kencana Ramli in [21]. Within this new framework, called weak completion semantics, a neural-symbolic approach was presented that covered most of the scenarios considered in Byrne's suppression task, but not all since some of them require skeptical abduction. Consider an example from the suppression task, given the premise:

if she has an essay to write then she will study late in the library.

If we know that *she will study late in the library*, according to Byrne [5] most of the subjects conclude that *she has an essay to write*. This inference process can be modeled using clasical logic and abduction, but consider now we additionaly know:

if she has a textbook to read then she will study late in the library.

In this modified scenario, consisting of two premises, there exist multiple explanations for the fact that *she will study late in the library*, and the conclussion *she has an essay to write* follows no more. To adequately model the latter scenario, skeptical abduction and non-monotonic reasoning is necessary.
In [24] the above mentioned approach from Hölldobler and Kencana Ramli based on weak completion semantics was extended to consider abduction. They showed that all scenarios considered by Byrne's suppression task can be adequately modeled by logic, although a connectionist model for it was still missing. In [16] an inspiring neural-symbolic approach was presented to compute abduction, but not considering skeptical reasoning neither capable of handling weak completion semantics. In this thesis, we provide formal specification of a neural-symbolic system to compute skeptical abduction. Our approach can be used to model the remaining above mentioned tasks, completing the modeling of Bayern's tasks in a fully connectionist realization, which constitutes our main contribution.
This thesis is structured as follows: In chapter 2 all necessary notions and background are introduced. In chapter 3 we provide two algorithms to compute skeptical abduction, along with their respective proofs. Then in chapter 4 we specify how to encode the algorithms into logic programs and how to translate them into a neural network. It is at the end of chapter 4 where the main results of this thesis are given. Finally, in chapter 6 we discuss the approach and its contributions, highlight the most relevant characteristics and weaknesses, and provide our conclusions.

Chapter 2

Preliminaries

2.1. Syntax

In the following we assume the reader to be familiar with logic, and logic programming.

2.1.1. Syntax of Logic Programs

Definition 1 (Atoms and literals)**:**
An atom *A is a propositional variable. A* literal *L is an atom A or its negation $\neg A$. A literal L of the form A is called a* positive literal*, whereas if it is of the form $\neg A$ it is called a* negative literal*.*

Definition 2:
A (program) clause *is an expression of the form:*

$$A \leftarrow L_1, \ldots, L_n \qquad (n > 0)$$

where A is an atom and $L_1, \ldots, L_n$ is a conjunction of literals. The atom A to the left of the implication is called the head *and $L_1, \ldots, L_n$ is called the* body *of the corresponding clause. A clause of the form:*

$$A \leftarrow \top$$

is called a positive fact *(or simply a* fact*) whereas a clause of the form:*

$$A \leftarrow \bot$$

is called a negative fact[1].

[1]Negative facts may seem counterintuitive at first, since a clause with *body* $= \bot$ will always be satisfiable. But as we will see, programs will be interpreted under weak completion semantics, where the implications are replaced by equivalences.

We also refer to a clause of the form: $A \leftarrow L_1, \ldots, L_n$ as $A \leftarrow body$, where $body = L_1, \ldots, L_n$.

Definition 3 (Logic program)**:**
A logic program $\mathcal{P}$ *is a finite set of clauses.*

Definition 4 (Atoms of a logic program)**:**
Let $\mathcal{P}$ *be a logic program, then* $\mathsf{atoms}(\mathcal{P})$ *is the set of all atoms occurring in* $\mathcal{P}$*.*

Definition 5 (Defined and undefined atoms)**:**
Given a logic program $\mathcal{P}$*, the set* $\mathsf{def}(\mathcal{P}) = \{A \mid A \leftarrow body \in \mathcal{P}\}$ *is the set of all defined atoms, that is atoms that are the head of a clause in* $\mathcal{P}$*. Likewise the set of undefined atoms in* $\mathcal{P}$ *is* $\mathsf{undef}(\mathcal{P}) = \mathsf{atoms}(\mathcal{P}) \setminus \mathsf{def}(\mathcal{P})$*.*

2.1.2. Weak Completion

Definition 6 (Weak completion of a program)**:**
Consider the following transformation, given a program $\mathcal{P}$*:*

1. *All clauses with the same head:* $A \leftarrow body_1,\ A \leftarrow body_2,\ A \leftarrow body_3 \ldots$ *, are replaced by:* $A \leftarrow body_1 \vee body_2 \vee body_3 \ldots$
2. *For all* A *in* $\mathsf{undef}(\mathcal{P})$ *add* $A \leftarrow \bot$*.*
3. *All occurrences of* $\leftarrow$ *are replaced by* $\leftrightarrow$*.*

The resulting set of equivalences is called Clark's completion *[2] of* $\mathcal{P}$ *and is denoted as* $\mathsf{c}\,\mathcal{P}$*. If we omit step 2 on the previous transformation, the resulting program is called the* weak completion *of* $\mathcal{P}$*, and is denoted by* $\mathsf{wc}\,\mathcal{P}$*.*

Example 2.1.1:
As an example consider program $\mathcal{P}$:

$$\mathcal{P} = \left\{ \begin{array}{lcl} p & \leftarrow & q \end{array} \right\}$$

Its completion is:

$$\mathsf{c}\,\mathcal{P} = \left\{ \begin{array}{lcl} p & \leftrightarrow & q \\ q & \leftrightarrow & \bot \end{array} \right\}$$

Its weak completion is:

$$\mathsf{wc}\,\mathcal{P} = \left\{ \begin{array}{lcl} p & \leftrightarrow & q \end{array} \right\}$$

2.2. Semantics

2.2.1. Interpretations and Models

Two-valued Interpretations and Models

Definition 7 (Two-valued interpretation)**:**
Given a logic program $\mathcal{P}$, a two-valued interpretation *I is a mapping from the set of atoms* $\mathsf{atoms}(\mathcal{P})$ *to the truth values $\{\top, \bot\}$. We represent two-valued interpretations as a set of atoms I, where each atom $A \in I$ is mapped to $\top$, and is mapped to $\bot$ otherwise.*

A literal L of the form A is mapped to $\top$ iff $A \in I$. If L is of the form $\neg A$, it is mapped to $\top$ iff $A \notin I$. The body of a clause containing literals is mapped to $\top$ under I, written as $I(body) = \top$, if all the literals in the body are mapped to $\top$. The body of a clause containing literals is mapped to $\bot$ under I, written as $I(body) = \bot$, if at least one literal in the body is mapped to $\bot$. A clause is mapped to $\bot$ under I if its body is mapped to $\top$ and the head mapped to $\bot$. It is mapped to $\top$, otherwise.

Definition 8 (Two-valued interpretation of formulas)**:**
A two-valued interpretation I is extended to formulas by applying the usual truth tables for two-valued logics. Given a formula F and a two-valued interpretation I we write $I(F) = \top$ iff F is mapped to true under I and $I(F) = \bot$ iff F is mapped to false under I.

Definition 9 (Two-valued model)**:**
Given a formula F, we say that a two-valued interpretation I is a two-valued model *for F if $I(F) = \top$. Given a program $\mathcal{P}$, I is a two-valued model for $\mathcal{P}$ if I maps each clause occurring in $\mathcal{P}$ to $\top$.*

Three-valued Logic

In this section two-valued logic is extended to three-valued Łukasiewicz logic, where additionally to true and false, atoms can be assigned a third truth value: unknown, represented by the symbol U. Table 2.1 shows the semantics of the conjunction ($\wedge$), disjunction ($\vee$), implication ($\leftarrow$), equivalence ($\leftrightarrow$) and negation ($\neg$) under three-valued Łukasiewicz logic.

Definition 10 (Three-valued interpretation)**:**
Given a logic program $\mathcal{P}$, a three-valued interpretation *I is a mapping from* $\mathsf{atoms}(\mathcal{P})$ *to the truth values $\top, \bot$ or U. We represent three-valued interpretations as a tuple of disjoint sets: $I = \langle I^{\top}, I^{\bot} \rangle$ where all atoms in $I^{\top}$ are mapped to $\top$, all atoms in $I^{\bot}$ are mapped to $\bot$ and all other atoms are mapped to U. We write $I(F) = \mathsf{U}$ iff F is mapped to unknown under I.*

F	$\neg$
$\top$	$\bot$
$\bot$	$\top$
U	U

F	G	$\wedge$	$\vee$	$\leftarrow$	$\leftrightarrow$
$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\top$	$\top$	$\bot$
$\top$	U	U	$\top$	$\top$	U
$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\top$
$\bot$	U	$\bot$	U	U	U
U	$\top$	U	$\top$	U	U
U	$\bot$	$\bot$	U	$\top$	U
U	U	U	U	$\top$	$\top$

Table 2.1: *Three-valued Łukasiewicz truth table*

In the following, interpretations are assumed to be three-valued, unless stated otherwise.

Definition 11 (Interpretation of formulas)**:**
An interpretation I is extended to formulas using the semantics provided in Table 2.1. Given a formula F and an interpretation I we write $I(F) = \top$ iff F is mapped to true under I, $I(F) = \bot$ iff F is mapped to false under I, and $I(F) = \mathsf{U}$ iff F is mapped to unknown under I.

Definition 12 (Model)**:**
Given a formula F, we say that an interpretation I is a model *for F iff $I(F) = \top$. Given a program $\mathcal{P}$, I is a model for $\mathcal{P}$ iff I maps each clause occurring in $\mathcal{P}$ to $\top$.*

Given two interpretations J and I, we want to determine when one of the interpretations is smaller. In the literature there are two common approaches measuring different aspects of interpretations: truth-ordering and knowledge-ordering. In the former the number of atoms mapped to $\top$ is meant to be minimized, whereas the number of atoms mapped to $\bot$ is maximized. In the latter, the number of atoms mapped to either $\top$ or $\bot$ is meant to be minimized. Here we focus on knowledge-ordering.

Definition 13 (Knowledge-ordering)**:**
Given two interpretations I and J we write $I \prec J$ iff:

$$I^{\top} \subset J^{\top} \text{ and } I^{\bot} \subset J^{\bot}$$

Accordingly we write $I \preceq J$ iff:

$$I^{\top} \subseteq J^{\top} \text{ and } I^{\bot} \subseteq J^{\bot}$$

Given the partial order $\preceq$ we are now ready to define minimality: we prefer models that minimize the number of atoms mapped to either $\top$ or $\bot$.

Definition 14 (Minimal and least model)**:**
Given a program $\mathcal{P}$ and a model I for $\mathcal{P}$, we say that I is a minimal model *for $\mathcal{P}$, if there exists no other model J for $\mathcal{P}$ with $J \preceq I$. If $\mathcal{P}$ has only one minimal model then it is the* least model *of $\mathcal{P}$.*

2.2.2. Semantic Operators

Two-valued Semantic Operator $T_\mathcal{P}$

Semantic operators provide a way to obtain the immediate consequences of a logic program. They take a two-valued interpretation I as an input, and yield and two-valued interpretation J as output.

Given a two-valued interpretation I and a program $\mathcal{P}$ the operator $T_\mathcal{P}$ is defined as follows:

Definition 15 (Two-valued semantic operator)**:**

$$T_\mathcal{P}(I) = \{A \mid A \leftarrow body \in \mathcal{P} \text{ and } I(body) = \top\}$$

Definition 16 ($T_\mathcal{P}$ operator fixpoint)**:**
A fixpoint of $T_\mathcal{P}$ is a two-valued interpretation I such that $T_\mathcal{P}(I) = I$. If in addition there exists no other fixpoint J with $J \subseteq I$, then I is called the least fixpoint *and is denoted as $T_\mathcal{P}^{\mathsf{lfp}}$.*

For *acceptable* programs [8], The least fixpoint of $T_\mathcal{P}$ correspond to the least model of $\mathcal{P}$, if it exists.
Given a program $\mathcal{P}$ and a formula F, we define the consequence relation:

$$\mathcal{P} \models^{T_\mathcal{P}} F \text{ iff } T_\mathcal{P}^{\mathsf{lfp}}(F) = \top$$

Three-valued Semantic Operator Φ

Stenning and van Lambalgen [16] introduced a semantic operator Φ for three-valued logic programs. If we iterate the operator starting with $\langle\emptyset, \emptyset\rangle$, it will eventually reach a point where the output and the input are equal. This so called least fixpoint, corresponds to the least model for $\mathsf{wc}\,\mathcal{P}$.

Definition 17 (Three-valued semantic operator)**:**
Given a logic program $\mathcal{P}$ and an interpretation I, we define:

$$\Phi_\mathcal{P}(I) = \langle J^\top, J^\bot\rangle$$

where:

$$J^{\top} = \{A | \textbf{\textit{there exists}} \textit{ a clause of the form } A \leftarrow body \in \mathcal{P} \textit{ with } I(body) = \top\}$$
$$J^{\bot} = \{A | \textit{there exists a clause of the form } A \leftarrow body \in \mathcal{P}, \textit{ and } \textbf{\textit{for all clauses}} \textit{ of the form } A \leftarrow body \textit{ we find that } I(body) = \bot\}$$

Stenning and van Lambalgen use Clark's completion of a logic program and three-valued Fitting semantics to model the suppression task and the selection task [16]. Unfortunately there is a technical bug in this approach that is solved by Hölldobler and Kencana Ramli in [9]. They show that, by using three-valued Łukasiewicz logic and weak completion semantics, each program $\mathcal{P}$ has exactly one least model, which can be computed using the Φ operator:

Definition 18 (Φ operator fixpoint)**:**
A fixpoint *of Φ given a program $\mathcal{P}$ is an interpretation I such that $\Phi_{\mathcal{P}}(I) = I$. If in addition there exists no other fixpoint J with $J \preceq I$, then I is called the* least fixpoint *and is denoted as $\Phi^{\mathsf{lfp}}_{\mathcal{P}}$.*

The $\Phi^{\mathsf{lfp}}_{\mathcal{P}}$ of any program $\mathcal{P}$ can be reached by iterating $\Phi_{\mathcal{P}}$ starting from the empty interpretation $I = \langle \emptyset, \emptyset \rangle$. Moreover, as shown in [21], the least fixpoint of Φ under three-valued Łukasiewicz logic, is equal to the least model for the weak completion of the program.
Given a program $\mathcal{P}$ and a formula F, we define the following consequence relation:

$$\mathcal{P} \models_{wcs} F \quad \text{iff} \qquad \Phi^{\mathsf{lfp}}_{\mathcal{P}}(F) = \top$$

Example 2.2.1:
Consider now:

$$\mathcal{P} = \left\{ \begin{array}{lcl} p & \leftarrow & q \\ q & \leftarrow & \top \\ r & \leftarrow & s \end{array} \right\}$$

Iterating the associated $\Phi_{\mathcal{P}}$ operator, starting from the empty interpretation $I_0 = \langle \emptyset, \emptyset \rangle$, yields:

$$\begin{array}{rcccl} I_1 & = & \Phi_{\mathcal{P}}(I_0) & = & \langle \{q\}, \emptyset \rangle \\ I_2 & = & \Phi_{\mathcal{P}}(I_1) & = & \langle \{q, p\}, \emptyset \rangle \\ I_2 & = & \Phi_{\mathcal{P}}(I_2) & = & \langle \{q, p\}, \emptyset \rangle \end{array}$$

Here, $I_2 = \langle \{q, p\}, \emptyset \rangle$ is the least fixpoint of $\Phi_{\mathcal{P}}$, since $I_2 = \Phi_{\mathcal{P}}(I_2)$, and corresponds to the least model for $\mathcal{P}_2$. Additionally we have:

$$\Phi^{\mathsf{lfp}}_{\mathcal{P}_2}(\{p\}) = \top \, , \, \Phi^{\mathsf{lfp}}_{\mathcal{P}_2}(\{q\}) = \top$$

$$\Phi^{\mathsf{lfp}}_{\mathcal{P}_2}(\{r\}) = \mathsf{U} \text{ , } \Phi^{\mathsf{lfp}}_{\mathcal{P}_2}(\{s\}) = \mathsf{U}$$

As we will see in Section 2.3.2 later in this chapter, semantic operators are strongly related to the process of constructing an artificial neural network equivalent to a logic program, in the sense that the network preserves the program semantics.

2.3. Neural Networks

2.3.1. Artificial Neural Networks

Artificial neural networks (in short neural networks) are systems composed of interconnected units where each unit performs a simple computation. The architecture, the connections and the functions computed by each unit define the behavior of the network as a whole.
The input and output of a neural network are vectors of real numbers. In this sense, a neural network is a system which computes a function $f : \mathbb{R}^n \to \mathbb{R}^m$, and is often provided with a mechanism to adjust its internal connections in such a way that the output vector matches an unknown function, which allows the network to approximate it.

Formally:

Definition 19 (Unit)**:**
Each unit *u consists of:*

- *An* input value (vector) $\vec{i} = (i_1, i_2, \ldots, i_n) \in \mathbb{R}^n$
- *An* activation function $\theta : \mathbb{R}^n \to \mathbb{R}$
- *A* potential $p = \theta(i_1, i_2, \ldots, i_n) \in \mathbb{R}$
- *An* output function $\zeta : \mathbb{R} \to \mathbb{R}$
- *An* output value $v = \zeta(p) \in \mathbb{R}$

Definition 20 (Weighted and directed connection)**:**
Given two units u_k, u_j, a weighted and directed connection $w_{kj} \in \mathbb{R}$ *(or simply a* connection*) states that there exists a connection to unit u_k from unit u_j with value w_{kj}. This value is called the* weight *of the connection.*

Each input value $i = w_{kj}v_j$ of an input vector $\vec{i}$, corresponds to a weighted connection from unit u_j to unit u_k, where v_j is the output value of unit u_j and w_{kj} is the weight of the connection.

Definition 21 (State)**:**
Let $U = \{u_i \mid 1 \leq i \leq n\}$ be an artificial neural network. The state *of U at time t is $(v_1(t), \ldots, v_n(t))$, where $v_i(t)$ is the output of unit u_i at time t.*

Units can have several activation functions. Here we consider so called *binary threshold units*. Such units are characterized by their activation and output function.

Definition 22 (Weighted sum activation function)**:**
Let $u_k \in U$ be a unit with input $\vec{i} = (i_1, \ldots, i_n)$ and let $i_j = w_{kj}v_j$, then the weighted sum activation function *θ of u_k is of the form:*

$$\theta(i_1, \ldots, i_n) = \sum_{j=1}^{n} i_j$$

Definition 23 (Binary threshold output function)**:**
Let $\Omega_i \in \mathbb{R}$ be a threshold *value for unit u_i, and let p_i be the current potential of u_i. The* binary threshold output function *ζ of u_i is of the form:*

$$\zeta = \begin{cases} 1 & \text{if } p_i \geq \Omega_i \\ 0 & \text{otherwise} \end{cases}$$

Definition 24 (Active and passive units)**:**
Given a binary threshold unit u, it is said to be active *if its output is 1; it is said to be* passive *if its output is 0. Unit u* flips *if its output changes from 1 to 0 or from 0 to 1 between two consecutive timesteps.*

Units make sequential computations. Meaning that if an input is given to a unit u at time t, the corresponding output of u is obtained at time $t + 1$. Formally:

Definition 25 (Updates)**:**
If a unit u is updated *then its potential p and value v are computed given the current inputs. Updates yield potentials and values at time $t + 1$, given inputs at time t.*

Definition 26 (Artificial neural network)**:**
An artificial neural network *is a tuple $\langle U, W \rangle$ where:*

- *U is a finite set of* units
- *W is a finite set of* connections

In this Thesis we will only consider networks where all units are updated synchronously.
Since the information propagates through the network, we are interested in how this process evolves from a initial state, where no units are active:

Definition 27 (Relaxed network)**:**
A neural network $N = \langle U, W \rangle$ *is* relaxed *if all units in* U *are passive.*

Units can be arranged in *layers*, where all units in layer B receive their inputs only from layer A, and all units in layer A output their values only to units in layer B. Neural Networks can be embedded in an *environment* in which case some units may interact with it.

Definition 28:
A 3-layered feed forward network *consists of:*

- *An input layer*
- *A hidden layer*
- *An output layer*

Units in the input layer, don't get their inputs from other units in the network, but externally. Units in the hidden layer receive their input values from the input layer, and output values to the output layer. Units in the output layer don't provide outputs for other units but to the environment of the network.

2.3.2. CORE Method

The CORE method [9, 13] translates programs into neural networks. It is based on the idea that feed-forward networks can approximate almost any function arbitrarily well, and that the semantics of a logic program is captured by the $T_{\mathcal{P}}$ operator. The following definition is based on [9]:

Definition 29 (The CORE method)**:**
We represent propositional variables by natural numbers; let $n \in \mathbb{N}$ *be the largest number occurring in* $\mathcal{P}$*. The network associated with* $\mathcal{P}$ *is constructed as follows:*

1. *The input and output layer is a vector of binary threshold units (with threshold* .5*) of length* n*, where the* i*-th unit in each layer represents the variable* i*,* $1 \leq i \leq n$*.*
2. *For every clause of the form* $A \leftarrow L_1 \wedge \cdots \wedge L_k$ *with* $k \geq 0$ *occurring in* $\mathcal{P}$ *do:*
   (a) *Add a binary threshold unit* c *to the hidden layer.*
   (b) *Connect* c *to the unit representing* A *in the output layer with weight 1.*
   (c) *For each* L *occurring in* $L_1 \wedge \cdots \wedge L_k$ *connect the unit representing* L *in the input layer to* c*. If* L *is an atom, then set the weight to* 1*, otherwise to* -1*.*

(d) Set threshold θ_c of c to $l - 0.5$, where l is the number of positive literals occurring in $L_1 \wedge \cdots \wedge L_k$.

Lemma 1 ([9])**:**
For each program $\mathcal{P}$ there exists a 3-layer network $\mathsf{Net}(\mathcal{P})$ that computes $T_\mathcal{P}$.

Lemma 2 ([11])**:**
For each acceptable[2] program $\mathcal{P}$ there exists a 3-layer recurrent $\mathsf{Net}(\mathcal{P})^{\circlearrowleft}$ network such that each computation starting with an arbitrary initial input converges and yields the unique fixpoint of $T_\mathcal{P}$.

Additionally, if each unit in the output layer is connected to the corresponding unit in the input layer with weight 1, we obtain the corresponding recurrent network.

It is important to remark, that the $T_\mathcal{P}$ associated to a non-acceptable program $\mathcal{P}$ under two-valued logic, is not always guaranteed to have a least fixpoint. Although this property may imply that iterating $T_\mathcal{P}$ may never end, it is desirable to model some of the components of our network, as we will see in chapter 4.

The Extended Core Method

Since the Core method was only defined for two-valued logic, an extension to multi-valued logic was necessary to apply it to a wider range of problems. This was attained independently in [10] and [12] where the former extends the Core method for three-valued logic and the latter makes a generalization for multi-valued logics.

The following definition is based on [17] where the resulting network computes the three-valued semantic operator Φ associated to $\mathcal{P}$:

Definition 30 (The extended Core method)**:**
Given a program $\mathcal{P}$, the following algorithm translates $\mathcal{P}$ into a feed-forward core. Let m be the number of propositional variables occurring in $\mathcal{P}$. Without loss of generality, we may assume that the variables are denoted by natural numbers in the range $[1, m]$. Let $\omega \in \mathbb{R}^+$.

1. *The input and output layers are vectors of binary threshold units of length $2m$ representing interpretations. The $(2i-1)$-th unit in the layers, denoted by $i^\top$, is active iff the i-th variable is mapped to $\top$. The $2i$-th unit in the layers, denoted by $i^\bot$, is active iff the i-th variable is mapped to $\bot$. Both, the $(2i-1)$-th and the $2i$-th unit, are passive iff the i-th variable is mapped to U. The case where both, the*

[2] Acceptable programs are a superclass of acyclic programs. For details the reader is referred to [8].

$(2i-1)$-th and the $2i$-th unit, are active is not allowed.
The threshold of each unit occurring in the input layer is set to $\frac{\omega}{2}$ .The threshold of each $(2i-1)$-th unit occurring in the output layer is set to $\frac{\omega}{2}$. The threshold of each $2i$-th unit occurring in the output layer is set to $max\left\{\frac{\omega}{2}, l-\frac{\omega}{2}\right\}$, where l is the number of clauses with head i in $\mathcal{P}$.
In addition, two units representing $\top$ and $\bot$ are added to the input layer. The threshold of these units is set to -$\frac{\omega}{2}$.

2. *For each clause of the form $A \leftarrow B_1, \ldots, B_k$ occurring in $\mathcal{P}$, do the following:*
   (a) *Add two binary threshold units $h^\top$ and $h^\bot$ to the hidden layer.*
   (b) *Connect $h^\top$ to the unit $A^\top$ in the output layer. Connect $h^\bot$ to the unit $A^\bot$ in the output layer.*
   (c) *For each B_j, $1 \leq j \leq k$, do the following:*
      i. *If B_j is an atom, then connect the units $B_j^\top$ and $B_j^\bot$ in the input layer to $h^\top$ and $h^\bot$, respectively.*
      ii. *If B_j is the literal $\neg B$, then connect the units $B^\bot$ and $B^\top$ in the input layer to $h^\top$ and $h^\bot$, respectively.*
      iii. *If B_j is $\top$, then connect the unit $\top$ in the input layer to $h^\top$.*
      iv. *If B_j is $\bot$, then connect the unit $\bot$ in the input layer to $h^\bot$.*
   (d) *Set the threshold of $h^\top$ to $kw - \frac{w}{2}$, and the threshold of $h^\bot$ to $\frac{\omega}{2}$.*
3. *Set the weights associated with all connections to ω.*

In this thesis, when constructing a network from a logic program we will use $\omega = 1$.

Definition 31 (Network corresponding to a logic program)**:**
The network resulting from applying the above algorithm to a program $\mathcal{P}$ is called the core *and is denoted as $Net(\mathcal{P})$. Additionally, if we connect each unit in the output layer with its corresponding unit in the input layer we obtain the associated* recurrent network, *denoted by $\mathsf{Net}(\mathcal{P})^{\circlearrowleft}$.*

Details can be found in [18]. Here we are mainly interested in two properties:

Proposition 1 ([17])**:**
For each program $\mathcal{P}$, there exists a core $\mathsf{Net}(\mathcal{P})$ of binary threshold units computing $\Phi_\mathcal{P}$.

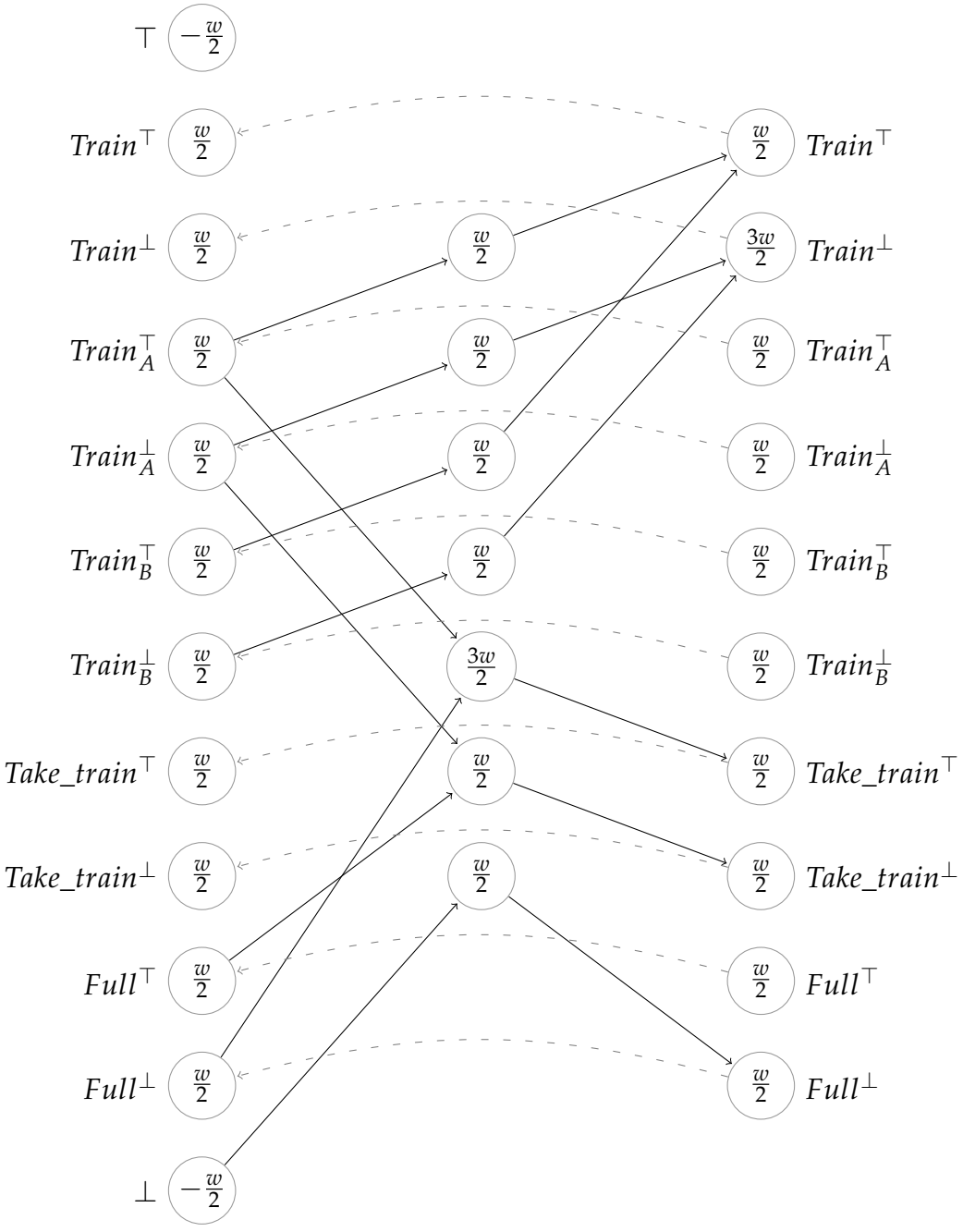
Proposition 2 ([17])**:**
For each program $\mathcal{P}$, the corresponding recurrent network $\mathsf{Net}(\mathcal{P})^{\circlearrowleft}$ initialized by the empty interpretation converges to a stable state which corresponds to the least fixpoint of $\Phi_\mathcal{P}$.

Example 2.3.1:
Let us illustrate the CORE method by another example, which is used throughout this Thesis to show how networks are constructed. Consider a passenger that is waiting for a train. There are only two possible trains that arrive at the station: $Train_A$ and $Train_B$. Whenever a train arrives there is a signal *Train* activated, but he would only *Take_train* if he is certain that it is indeed $Train_A$, and it is not *Full*. From where he is standing, he can only see the train is not *Full*. The following program $\mathcal{P}_{Train}$ represents this knowledge:

$$\mathcal{P}_{Train} = \left\{ \begin{array}{lcl} Train & \leftarrow & Train_A \\ Train & \leftarrow & Train_B \\ Take\_train & \leftarrow & Train_A, \neg Full \\ Full & \leftarrow & \bot \end{array} \right\}$$

Figure 2.1 shows the network $\mathsf{Net}(\mathcal{P}_{Train})^{\circlearrowleft}$ obtained by applying the CORE method to program $\mathcal{P}_{Train}$. The input as well as the output layer, each have two copies for every atom $A \in \mathsf{atoms}(\mathcal{P})$. One positive $A^{\top}$ and one negative $A^{\bot}$. The input layer has two additional units representing $\top$ and $\bot$ respectively. For each unit associated to an atom $A \in \mathsf{atoms}(\mathcal{P})$, a recurrent connection is added, showed in dashed lines.

Figure 2.1: *Figure 2.1 shows the recurrent network* $\mathsf{Net}(\mathcal{P}_{Train})^{\circlearrowleft}$ *obtained from program* $\mathcal{P}_{Train}$*. Recurrent connections are shown in dashed lines.*



2.4. Abduction

Abduction has been formalized by Kakas, Kowalski and Toni in [7] where abductive frameworks are introduced. The definitions given here are based on their approach. Before introducing abductive frameworks, let us first present integrity constraints.

2.4.1. Integrity Constraints

When looking for models for a program $\mathcal{P}$, we would like to impose some restrictions on how these models should look like, we do this by introducing integrity constraints. In the literature, there are several definitions for understanding integrity constraints in abduction [7, 24, 25] which mostly differ on when an integrity constraint is considered to be satisfied. We present a definition suitable for weak completion semantics:

Definition 32 (Set of integrity constraints)**:**
An integrity constraint *is an expression of the form* $\mathsf{U} \leftarrow L_1, \ldots, L_n$*, where* $L_1, \ldots, L_n$ *is a conjunction of literals. A* set of integrity constraints *is denoted by* $\mathcal{IC}$*.*

We see an integrity constraint $\mathsf{U} \leftarrow$ *body* as a way of rejecting any model for a program $\mathcal{P}$ where the literals in *body* are all true at the same time. That is, whenever the body of a clause is mapped to U or $\bot$ by a model I for $\mathcal{P}$, the integrity constraint will be mapped to $\top$, and thus the model will be accepted.

2.4.2. Abductive Framework

Definition 33 (Abductive framework)**:**
An abductive framework *AF is a tuple* $\langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$ *where:*

- $\mathcal{P}$ *is a logic program.*
- $\mathcal{A}_\mathcal{P} = \{A \leftarrow \top | A \in \mathsf{undef}(\mathcal{P})\} \cup \{A \leftarrow \bot | A \in \mathsf{undef}(\mathcal{P})\}$*, is called the set of abducibles.*
- $\models_{wcs}$ *is the consequence relation defined in Section 2.2.2.*
- $\mathcal{IC}$ *is a finite set of integrity constraints.*

Definition 34 (Explanation)**:**
Consider an abductive framework $\langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$*. Let* $\mathcal{O}$ *be a non-empty finite set of*

literals called observations. *Then given $\mathcal{P}$, we say that $\mathcal{O}$ is* explained by $\mathcal{E} \subseteq \mathcal{A}_\mathcal{P}$ *(or $\mathcal{E}$ is an explanation for $\mathcal{O}$) iff:*

$$\mathcal{P} \cup \mathcal{E} \models_{wcs} \mathcal{O}$$

Additionally an explanation $\mathcal{E}$ is said to be minimal, *if there is no $\mathcal{E}' \subset \mathcal{E}$ such that $\mathcal{E}'$ is an explanation for $\mathcal{O}$.*

Definition 35 (Integrity constraint satisfaction)**:**
Given an abductive framework $\langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$, a set of integrity constraints $\mathcal{IC}$ and an observation $\mathcal{O}$, we say that an explanation $\mathcal{E} \subseteq \mathcal{A}_\mathcal{P}$ satisfies *an integrity constraint $\mathsf{U} \leftarrow body$ iff $\Phi^{\mathsf{lfp}}_{\mathcal{P}\cup\mathcal{E}}(\mathsf{U} \leftarrow body) = \top$. A set of integrity constraints $\mathcal{IC}$ is satisfied by $\mathcal{P} \cup \mathcal{E}$ iff all of its elements are satisfied.*

In the following, we consider only explanations that satisfy the integrity constraints of the corresponding abductive framework.

Intuitively, given an AF and an observation $\mathcal{O} = \{L_1, \ldots, L_n\}$, the task of abductive logic programming is to find an explanation $\mathcal{E}$ for $\mathcal{O}$. This is done by adding a combination of the abducibles to the program: $\mathcal{P} \cup \mathcal{E}$, and then verify if the observation $\mathcal{O}$ follows from it.
Note that under the weak completion semantics, every program $\mathcal{P}$ has a least fixpoint. This means programs are always satisfiable otherwise Definition 34 would be trivial, i.e if $\mathcal{P}$ is unsatisfiable any $\mathcal{E} \subseteq \mathcal{A}_\mathcal{P}$ would be an explanation for $\mathcal{O}$. For the same reason we don't require explanations to be consistent since negative facts will be overwritten by positive facts. In other words, an explanation $\mathcal{E}_1 = \{A \leftarrow \top\ ,\ A \leftarrow \bot\}$, will have the same effect as $\mathcal{E}_2 = \{A \leftarrow \top\}$.

Example 2.4.1:
Recall our train example in Section 2.3.1. Suppose we now observe that signal *Train* is active. We are interested in finding out why it is active by performing abduction. When looking for explanations, we impose the integrity constraint that it is not the case that both trains arrive at the same time. Our abductive framework $\mathsf{AF} = \langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$ then consists of:

$$\mathcal{P}_{Train} = \left\{ \begin{array}{lcl} Train & \leftarrow & Train_A \\ Train & \leftarrow & Train_B \\ Take\_train & \leftarrow & Train_A, \neg Full \\ Full & \leftarrow & \bot \end{array} \right\}$$

$$\mathcal{A}_\mathcal{P} = \{Train_A \leftarrow \top, Train_A \leftarrow \bot, Train_B \leftarrow \top, Train_B \leftarrow \bot\}$$
$$\mathcal{IC} = \{\mathsf{U} \leftarrow Train_A, Train_B\}$$

and the observation $\mathcal{O} = \{Train\}$. There are four explanations for $\mathcal{O}$:

- $\mathcal{E}_1 = \{Train_A \leftarrow \top\}$
- $\mathcal{E}_2 = \{Train_B \leftarrow \top\}$
- $\mathcal{E}_3 = \{Train_A \leftarrow \top, Train_B \leftarrow \bot\}$
- $\mathcal{E}_4 = \{Train_A \leftarrow \bot, Train_B \leftarrow \top\}$

Here, $\mathcal{E}_1$ is an explanation for $\mathcal{O}$ since $\Phi^{\mathsf{lfp}}_{\mathcal{P}\cup\mathcal{E}_1}(Train) = \top$, and so are $\mathcal{E}_2$,$\mathcal{E}_3$,$\mathcal{E}_4$. However only $\mathcal{E}_1$ and $\mathcal{E}_2$ are minimal, since $\mathcal{E}_1 \subset \mathcal{E}_3$ and $\mathcal{E}_2 \subset \mathcal{E}_4$. Note that $\{Train_A \leftarrow \top, Train_B \leftarrow \top\}$ is not a valid explanation, because it does not satisfy IC.

2.4.3. Credulous vs. Skeptical Reasoning

In the literature relevant to human reasoning and skeptical abduction [24, 25, 26], the usual definition is as follows:

Definition 36:
Given an abductive framework $AF = \langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$, an observation $\mathcal{O}$ and a formula F:

- *F follows* skeptically *from $\mathcal{P}$, $\mathcal{IC}$ and $\mathcal{O}$ iff $\mathcal{O}$ can be explained, and* ***for all*** *minimal explanations $\mathcal{E}$ for $\mathcal{O}$ we find $\Phi^{\mathsf{lfp}}_{\mathcal{P}\cup\mathcal{E}}(F) = \top$.*
- *F follows* credulously *from $\mathcal{P}$, $\mathcal{IC}$ and $\mathcal{O}$ iff* ***there exists*** *a minimal explanation $\mathcal{E}$ for $\mathcal{O}$ and $\Phi^{\mathsf{lfp}}_{\mathcal{P}\cup\mathcal{E}}(F) = \top$.*

The above definitions are restricted to minimal explanations. Initially this restriction seems unnecessary, since we find that the formulas mapped to $\top$ or to $\bot$ under all explanations (including non-minimal ones) are the same, as imposing minimality:

Proposition 3:
Given $AF = \langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$, an observation $\mathcal{O}$ and a formula F, we have that:

F follows from all minimal explanations for $\mathcal{O}$ iff

F follows from all explanations for $\mathcal{O}$

Proof. In [24] it is shown that $\Phi(I) \subseteq \Phi(I \cup \mathcal{E})$ for $\mathcal{E} \subset \mathcal{A}_\mathcal{P}$, that is the Φ operator is monotonic, if we only add abducibles to the input of Φ. Consequently, if we extend an explanation by any subset of abducibles, no inferences are suppressed. We conclude that if a formula F follows from a minimal explanation $\mathcal{E}$, it also follows from any $\mathcal{E}' \supset \mathcal{E}$. On the other hand, if F follows from an arbitrary non-minimal explanation $\mathcal{E}'$, it follows as well from a minimal explanation $\mathcal{E} \subset \mathcal{E}'$. $\square$

To illustrate Proposition 3 consider the abductive framework $\langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$ where:

$$\mathcal{P} = \left\{ \begin{array}{lcl} p & \leftarrow & q \\ r & \leftarrow & s \end{array} \right\}$$
$$\mathcal{A}_\mathcal{P} = \{q \leftarrow \top, q \leftarrow \bot, s \leftarrow \top, s \leftarrow \bot\}$$
$$\mathcal{IC} = \emptyset$$

and the observation $\mathcal{O} = \{p\}$. There are three explanations for $\mathcal{O}$:

- $\mathcal{E}_1 = \{q \leftarrow \top\}$
- $\mathcal{E}_2 = \{q \leftarrow \top, s \leftarrow \top\}$
- $\mathcal{E}_3 = \{q \leftarrow \top, s \leftarrow \bot\}$

From which only $\mathcal{E}_1$ is minimal. Then, for each explanation we have that:

- $\mathcal{P} \cup \mathcal{E}_1 \models_{wcs} p \wedge q$
- $\mathcal{P} \cup \mathcal{E}_2 \models_{wcs} p \wedge q \wedge r \wedge s$
- $\mathcal{P} \cup \mathcal{E}_3 \models_{wcs} p \wedge q$

It is easy to see that p and q follow from both: from all explanations and from all minimal explantions.

Consequently we have that minimal explanations suffice to characterize the skeptical consequences of an abductive framework and an observation. That is, instead of considering all explanations, we can focus only on the minimal ones. But the problem is that first we have to obtain them.

We emphasize this point, because distinguishing between minimal and non-minimal explanations is not a trivial task. In [24] it is shown that determining if there exist a minimal explanation is *NP-complete*, and that computing skeptical abduction is *DP-complete*. So, why would we keep such a restriction? We show that more inferences can be made if we consider only minimal explanations.

There are possibly some atoms that do not play a role at all in abduction: neither explaining $\mathcal{O}$ nor as a consequence of the explanation. These are the atoms that remain unknown. Any non-minimal explanation would override the value of these unknown atoms, and will prevent us from detecting them. Consider the following example of imposing that explanations should be minimal:

Example 2.4.2:
Consider now:

$$\mathcal{P}_3 = \left\{ \begin{array}{lcl} p & \leftarrow & q \\ r & \leftarrow & s \end{array} \right\}$$

with:

$$\begin{array}{rcl} \mathcal{A}_{\mathcal{P}} & = & \{q \leftarrow \top, q \leftarrow \bot, s \leftarrow \top, s \leftarrow \bot\} \\ \mathcal{O} & = & \{p\} \end{array}$$

Then, there exist three explanations for $\mathcal{O}$:

$$\mathcal{E}_1 = \{q \leftarrow \top\}\ , \mathcal{E}_2 = \{q \leftarrow \top, s \leftarrow \top\} \text{ and } \mathcal{E}_3 = \{q \leftarrow \top, s \leftarrow \bot\}$$

Since only $\mathcal{E}_1 = \{q \leftarrow \top\}$ is minimal and $\Phi^{\mathsf{lfp}}_{\mathcal{P}_3 \cup \mathcal{E}_1} = \langle \{p, q\}, \emptyset \rangle$, this means that additionally to p, q being skeptically entailed (as positive literals) we can also conclude that r and s are skeptically entailed, as unknown literals, i.e under every minimal explanation r and s are unknown. Had minimality not been used, $\mathcal{E}_3 = \{q \leftarrow \top, s \leftarrow \bot\}$ would yet be another explanation whose least fixpoint $\Phi^{\mathsf{lfp}}_{\mathcal{P}_3 \cup \mathcal{E}_3} = \langle \{p, q\}, \{r, s\} \rangle$. In this case we can no longer make skeptical inferences about r and s, since neither $\Phi^{\mathsf{lfp}}_{\mathcal{P}_3 \cup \mathcal{E}_3}(r) = \mathsf{U}$ nor $\Phi^{\mathsf{lfp}}_{\mathcal{P}_3 \cup \mathcal{E}_3}(s) = \mathsf{U}$ holds.

If we understand skeptical inferences as formulas preserving its truth value under all minimal explanations for a given observation. The current definition of skeptical abduction handles only the cases when a formula is true or false under all explanations, but it doesn't tell us anything about those formulas mapped to unknown. Consequently, we extend the definition of skeptical entailment to refer to these atoms.

Definition 37 (Skeptically unknown literals)**:**
Given $AF = \langle \mathcal{P}, \mathcal{A}_{\mathcal{P}}, \mathcal{IC}, \models_{wcs} \rangle$, an observation $\mathcal{O}$ and a formula F:
F follows skeptically as unknown *from $\mathcal{P}$, $\mathcal{IC}$ and $\mathcal{O}$ iff $\mathcal{O}$ can be explained, and* ***for all*** *minimal explanations $\mathcal{E}$ for $\mathcal{O}$ we find $\Phi^{\mathsf{lfp}}_{\mathcal{P} \cup \mathcal{E}}(F) = \mathsf{U}$.*

Chapter 3

Computing Skeptical Abduction

In this chapter we show how skeptical abduction can be computed. The task is divided in two steps: first we need to determine the set of all minimal explanations for $\mathcal{P}, \mathcal{IC}$ and $\mathcal{O}$. Then, from this set of explanations we can identify the skeptically entailed literals.

3.1. Ordering Candidate Explanations

Recall that, given an abductive framework $\mathsf{AF} = \langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$, $\mathcal{A}_\mathcal{P}$ is the set of abducibles. And that, given an observation $\mathcal{O}$, the task of an abductive logic program is to find explanations $\mathcal{E} \subseteq \mathcal{A}_\mathcal{P}$ for $\mathcal{O}$. Not all subsets of $\mathcal{A}_\mathcal{P}$ are valid explanations. Since initially we don't know which of them explain $\mathcal{O}$, we refer to all of them as candidate explanations.

Definition 38 (Candidate explanations)**:**
Given an abductive framework $\mathsf{AF} = \langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$ and an observation $\mathcal{O}$, a candidate explanation *$\mathcal{C}$ for $\mathcal{O}$ is any subset of $\mathcal{A}_\mathcal{P}$.*

Before we turn to the algorithm, we need to define an order on the candidate explanations $\mathcal{C}$. This ordering will allow us to simplify the computation of all minimal explanations for $\mathcal{O}$.

Definition 39 (Ordered set of candidate explanations)**:**
Let $\prec$ be any total ordering over $2^{\mathcal{A}_\mathcal{P}}$ such that for every $\mathcal{C}, \mathcal{C}' \in 2^{\mathcal{A}_\mathcal{P}}$ we have that $|\mathcal{C}| < |\mathcal{C}'|$ implies $\mathcal{C} \prec \mathcal{C}'$. Let $Ord(\mathcal{A}_\mathcal{P}) = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n\}$ be an ordered set with exactly the elements of $2^{\mathcal{A}_\mathcal{P}}$ such that $\mathcal{C}_1 \prec \mathcal{C}_2 \prec \cdots \prec \mathcal{C}_n$. Then $Ord(\mathcal{A}_\mathcal{P})$ is the ordered set of candidate explanations *for $\mathcal{O}$ with respect to AF.*

As an example consider:

Example 3.1.1:
Given :

$$\mathcal{A}_{\mathcal{P}} = \{A \leftarrow \top, A \leftarrow \bot\}$$

with :

1. $\mathcal{C}_0 = \emptyset$
2. $\mathcal{C}_1 = \{A \leftarrow \top\}$
3. $\mathcal{C}_2 = \{A \leftarrow \bot\}$
4. $\mathcal{C}_3 = \{A \leftarrow \top, A \leftarrow \bot\}$

we have that :

$$|\mathcal{C}_0| < |\mathcal{C}_1| \leq |\mathcal{C}_2| < |\mathcal{C}_3|$$

Consequently the following total order can be used to construct $Ord(\mathcal{A}_{\mathcal{P}})$:

$$\mathcal{C}_0 \prec \mathcal{C}_1 \prec \mathcal{C}_2 \prec \mathcal{C}_3$$

yielding:

$$Ord(\mathcal{A}_{\mathcal{P}}) = \{\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$$

Given AF, $\mathcal{O}$ and $Ord(\mathcal{A}_{\mathcal{P}})$, all minimal explanations are determined by Algorithm 1, introduced next.

3.2. Algorithm 1: Computing all Minimal Explanations

Data: $\langle \mathcal{P}, \mathcal{A}_{\mathcal{P}}, \mathcal{IC}, \models_{wcs} \rangle$ and $\mathcal{O}$
Output: Set of minimal explanations $\mathcal{M}_{\mathcal{E}}$ for $\mathcal{O}$

```
1 M_E = ∅
2 n = |Ord(A_P)|
3 for i = 1 to n, with C_i ∈ Ord(A_P) do
4 |   1) if C_i ⊉ E, for all E ∈ M_E, then
5 |   |   2) if C_i explains O then
6 |   |   |   add C_i to M_E
7 |   |   end
8 |   end
9 end
```

Algorithm 1: The computation of all minimal explanations.

Lemma 3:
Consider $\langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$ *and observation* $\mathcal{O}$*. Then for all* $\mathcal{C} \in Ord(\mathcal{A}_\mathcal{P})$ *holds: Whenever* $\mathcal{C}$ *is tested as part of condition 1) of Algorithm 1,* $\mathcal{M}_\mathcal{E}$ *contains all minimal explanations* $\mathcal{E}$ *for* $\mathcal{O}$ *where* $|\mathcal{E}| < |\mathcal{C}|$*.*

Proof. At the moment $\mathcal{C} \in Ord(\mathcal{A}_\mathcal{P})$ is tested, if there exists a minimal explanation $\mathcal{E} \subset \mathcal{C}$ then necessarily $\mathcal{E}$ was tested before $\mathcal{C}$, since $|\mathcal{E}| < |\mathcal{C}|$. As $\mathcal{E}$ is a minimal explanation, it satisfies conditions 1) and 2) of Algorithm 1 and consequently was added to $\mathcal{M}_\mathcal{E}$. Therefore there cannot exist a minimal explanation smaller than $\mathcal{C}$ that is not in $\mathcal{M}_\mathcal{E}$. □

Proposition 4:
Consider $\langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$ *an observation* $\mathcal{O}$*, and let* $\mathcal{M}_\mathcal{E}$ *be the set computed by Algorithm 1.*

1. $\mathcal{M}_\mathcal{E}$ *contains only minimal explanations for* $\mathcal{O}$*.*
2. *All minimal explanations for* $\mathcal{O}$ *are contained in* $\mathcal{M}_\mathcal{E}$*.*

Proof. 1. We know that every $\mathcal{E} \in \mathcal{M}_\mathcal{E}$ satisfies conditions 1) and 2) of Algorithm 1. By Lemma 3 and condition 1) there is no $\mathcal{C} \in Ord(\mathcal{A}_\mathcal{P})$ with $\mathcal{C} \subset \mathcal{E}$ such that $\mathcal{C}$ is a minimal explanation for $\mathcal{O}$. Since $\mathcal{E}$ also satisfies condition 2), we have that $\mathcal{E}$ explains $\mathcal{O}$. Then, every $\mathcal{E} \in \mathcal{M}_\mathcal{E}$ is a minimal explanation for $\mathcal{O}$.
2. (by contradiction) Assume that all candidate explanations have been tested. Let $\mathcal{E} \notin \mathcal{M}_\mathcal{E}$ be a minimal explanation. Then there exists no explanation $\mathcal{E}' \subset \mathcal{E}$, i.e., $\mathcal{E}$ satisfies condition 1) and, because $\mathcal{E}$ explains $\mathcal{O}$, it must have been added to $\mathcal{M}_\mathcal{E}$ by condition 2) of Algorithm 1. But then, $\mathcal{E} \in \mathcal{M}_\mathcal{E}$ (contradiction). □

Example 3.2.1:
As an example consider the following abductive framework
$\mathsf{AF} = \langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$, where:

$$\mathcal{P} = \left\{ \begin{array}{lll} a & \leftarrow & b, c \\ a & \leftarrow & \neg d, c \\ e & \leftarrow & f \end{array} \right\}$$

$$\mathcal{O} = \{a\}$$

$$\mathsf{undef}(\mathcal{P}) = \{b, c, d, f\}$$

$$\mathcal{A}_\mathcal{P} = \left\{ \begin{array}{llll} b \leftarrow \top & , & b \leftarrow \bot \\ c \leftarrow \top & , & c \leftarrow \bot \\ d \leftarrow \top & , & d \leftarrow \bot \\ f \leftarrow \top & , & f \leftarrow \bot \end{array} \right\}$$

We apply Algorithm 1 to AF where initially:

$$\mathcal{M}_{\mathcal{E}} = \emptyset$$
$$|Ord(\mathcal{A}_{\mathcal{P}})| = |2^{\mathcal{A}_{\mathcal{P}}}| = 2^{|\mathcal{A}_{\mathcal{P}}|} = 2^8 = 256$$

Notice that, even with a small number of atoms to be abduced (in our case 4) the number of possible solutions to explore is exponential, in this case 256.

The **for** cycle in line 4 in Algorithm 1 will test all $\mathcal{C} \in Ord(\mathcal{A}_{\mathcal{P}})$ to decide if they are minimal explanations.

Given that $Ord(\mathcal{A}_{\mathcal{P}})$ is ordered according to $\prec$, the empty set $\emptyset$ as well as all singleton sets will be tested first. As any explanation for $\mathcal{O} = \{a\}$ must contain the combination $\{b \leftarrow \top, c \leftarrow \top\}$ or $\{d \leftarrow \bot, c \leftarrow \top\}$ no set with arity 1 or smaller, can be an explanation for $\mathcal{O}$, and thus they are all discarded by condition 2).

Next, all $\mathcal{C} \in Ord(\mathcal{A}_{\mathcal{P}})$ with $|\mathcal{C}| = 2$ are tested, from which only two of them explain $\mathcal{O}$, namely: $\mathcal{C}_1 = \{b \leftarrow \top, c \leftarrow \top\}$ and $\mathcal{C}_2 = \{d \leftarrow \bot, c \leftarrow \top\}$.

Without loss of generality, we can assume $\mathcal{C}_1$ is tested first. It will satisfy conditions 1) and 2) of Algorithm 1, and consequently be added to $\mathcal{M}_{\mathcal{E}}$, becoming its first element, yielding $\mathcal{E}_1 = \mathcal{C}_1 = \{b \leftarrow \top, c \leftarrow \top\}$.
When $\mathcal{C}_2$ is tested, $\mathcal{M}_{\mathcal{E}} = \{\mathcal{E}_1\}$ and as $\mathcal{C}_2 \not\supset \mathcal{E}_1$ we have that $\mathcal{C}_2$ also satisfies conditions 1) and 2) of Algorithm 1, and it's also added to $\mathcal{M}_{\mathcal{E}}$. At this point we have $\mathcal{M}_{\mathcal{E}} = \{\mathcal{E}_1, \mathcal{E}_2\}$.

Finally, for any further discovered explanation $\mathcal{C}$ we will either have $\mathcal{E}_1 \subset \mathcal{C}$ or $\mathcal{E}_2 \subset \mathcal{C}$, thus violating condition 1) of Algorithm 1. After all 256 candidate explanations are tested:

$$\mathcal{M}_{\mathcal{E}} = \{\{b \leftarrow \top, c \leftarrow \top\}, \{d \leftarrow \bot, c \leftarrow \top\}\}$$

3.3. Algorithm 2: Computing Skeptically Entailed Literals

Algorithm 2 determines which literals follow skeptically by verifying if they are entailed (as positive, negative or unknown respectively) by all minimal explanations:

Proposition 5:
Consider $\langle \mathcal{P}, \mathcal{A}_{\mathcal{P}}, \mathcal{IC}, \models_{wcs} \rangle$ and an observation $\mathcal{O}$. Let $\mathcal{S}^{\top}, \mathcal{S}^{\bot}, \mathcal{S}^{\mathsf{U}}$ be the sets computed by Algorithm 2.

1. *$\mathcal{S}^{\top}$ contains all skeptically entailed positive atoms.*

[1]This set is computed by Algorithm 1.

Data: $\langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$, $\mathcal{O}$ and $\mathcal{M}_\mathcal{E}$[1]
Output: Sets $\mathcal{S}^\top, \mathcal{S}^\bot, \mathcal{S}^\mathsf{U}$

$j = 0$
for *each* $\mathcal{E} \in \mathcal{M}_\mathcal{E}$ **do**
  increment j by 1
  $\mathcal{F}_j^\top = \{ A \mid A \in \mathsf{atoms}(\mathcal{P})$ and $\mathcal{P} \cup \mathcal{E} \models_{wcs} A \}$
  $\mathcal{F}_j^\bot = \{ A \mid A \in \mathsf{atoms}(\mathcal{P})$ and $\mathcal{P} \cup \mathcal{E} \models_{wcs} \neg A \}$
  $\mathcal{F}_j^\mathsf{U} = \{ A \mid A \in \mathsf{atoms}(\mathcal{P}) \setminus (\mathcal{F}_j^\top \cup \mathcal{F}_j^\bot) \}$
end
if $j \geq 1$ **then**
  $\mathcal{S}^\top = \cap_{k=1}^{j} \mathcal{F}_k^\top$
  $\mathcal{S}^\bot = \cap_{k=1}^{j} \mathcal{F}_k^\bot$
  $\mathcal{S}^\mathsf{U} = \cap_{k=1}^{j} \mathcal{F}_k^\mathsf{U}$
end

Algorithm 2: The computation of skeptical consequences.

2. $\mathcal{S}^\bot$ *contains all skeptically entailed negative atoms.*

3. $\mathcal{S}^\mathsf{U}$ *contains all skeptically entailed unknown atoms.*

Proof. We show the proof for $\mathcal{S}^\top$. If $j = 0$ then $\mathcal{O}$ can not be explained, and nothing follows skeptically. Accordingly, the set $\mathcal{S}^\top$ will be empty. If at least one minimal explanation exists, we show that all elements in $\mathcal{S}^\top$ follow skeptically. Let $A \in \mathcal{S}^\top$. Then, because $\mathcal{S}^\top = \cap_{k=1}^{j} \mathcal{F}_k^\top$ it follows that $A \in \mathcal{F}_k^\top$ with $1 \leq k \leq j$. As Algorithm 2 creates a set $\mathcal{F}_k^\top$ for each $\mathcal{E}_k \in \mathcal{M}_\mathcal{E}$, we have that $A \in \mathcal{F}_k^\top$ implies $\Phi^{\mathsf{lfp}}_{\mathcal{P} \cup \mathcal{E}_k}(A) = \top$ for all $\mathcal{E}_k \in \mathcal{M}_\mathcal{E}$. By Proposition 4, the set $\mathcal{M}_\mathcal{E}$ contains exactly all minimal explanations for $\mathcal{O}$. Consequently, A follows from all minimal explanations for $\mathcal{O}$.
The proofs for $\mathcal{S}^\bot$ and $\mathcal{S}^\mathsf{U}$ can be shown similarly to $\mathcal{S}^\top$. □

Example 3.3.1:
Consider again Example 3.2.1. Now we apply Algorithm 2 to $\mathsf{AF} = \langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$. Table 3.1 shows the computations of the sets $\mathcal{F}^\top, \mathcal{F}^\bot, \mathcal{F}^\mathsf{U}$, given $\mathcal{M}_\mathcal{E}$ computed by Algorithm 1.

j	$\mathcal{E}_j$	$\mathcal{P} \cup \mathcal{E}_j$			$\mathcal{F}_j^{\top}$	$\mathcal{F}_j^{\bot}$	$\mathcal{F}_j^{\mathsf{U}}$
1	$\{b \leftarrow \top, c \leftarrow \top\}$	$\mathcal{P} \cup \mathcal{E}_1$	$\models_{wcs}$	$\{a, b, c\}$	$\{a, b, c\}$	$\emptyset$	$\{d, e, f\}$
2	$\{d \leftarrow \bot, c \leftarrow \top\}$	$\mathcal{P} \cup \mathcal{E}_2$	$\models_{wcs}$	$\{a, c, \neg d\}$	$\{a, c\}$	$\{d\}$	$\{b, e, f\}$

Table 3.1: *Computation of sets $\mathcal{F}_j^{\top}$,$\mathcal{F}_j^{\bot}$,$\mathcal{F}_j^{\mathsf{U}}$ by Algorithm 2*

Once we obtain the sets $\mathcal{F}_j^{\top}, \mathcal{F}_j^{\bot}, \mathcal{F}_j^{\mathsf{U}}$ for $1 \leq j \leq 2$ we can compute their intersections yielding the sets $\mathcal{S}^{\top}, \mathcal{S}^{\bot}$ and $\mathcal{S}^{\mathsf{U}}$:

$$\mathcal{S}^{\top} = \mathcal{F}_1^{\top} \cap \mathcal{F}_2^{\top} = \{a, c\}$$

$$\mathcal{S}^{\bot} = \mathcal{F}_1^{\bot} \cap \mathcal{F}_2^{\bot} = \emptyset$$

$$\mathcal{S}^{\mathsf{U}} = \mathcal{F}_1^{\mathsf{U}} \cap \mathcal{F}_2^{\mathsf{U}} = \{e, f\}$$

It is easy to see that $\mathcal{S}^{\top}$ contains all positive atoms skeptically entailed from AF and $\mathcal{O}$, $\mathcal{S}^{\bot}$ contains all those negative atoms skeptically entailed, and $\mathcal{S}^{\mathsf{U}}$ contains all skeptically entailed unknown atoms.

Chapter 4

A Connectionist Realization

In this chapter we specify how to obtain a neural network such that, given an abductive framework $AF = \langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$ and an observation $\mathcal{O}$, it computes the sets $\mathcal{S}^\top, \mathcal{S}^\bot$ and $\mathcal{S}^\mathsf{U}$ introduced in Chapter 3.

Our approach is inspired by [14, 15, 22] where a connectionist realization for abduction is provided. Their network computes only credulous consequences under two-valued logic. It was extended in [30] to compute skeptical consequences, still under two-valued logic. In this thesis we extend both by, on one hand using three-valued Łukasiewicz logic and weak completion semantics, and on the other hand by computing skeptical inferences.

The idea is as follows: three-valued logic and weak completion semantics are required to specify an abductive framework AF where the consequences drawn from it correspond to the conclusions drawn by humans [24, 25, 26]. The abductive framework and its corresponding neural network are detailed in Section 4.1. The corresponding neural network Net(ALP) is at the center of our approach, over which abduction is computed in several steps: first we need a way to produce all minimal explanations and somehow sequentially add them to Net(ALP). For each explanation added, the next step is to monitor the output of Net(ALP) to obtain the logical consequences of the explanation. Once the consequences of each explanation are available, we need to compute their intersection to finally obtain what skeptically follows.

For each of these tasks, a separate component is presented. When all components are integrated, we obtain a single neural network that computes the skeptical consequences of AF and $\mathcal{O}$, under weak completion semantics. Figure 4.1 depicts the general structure of the network and its components.

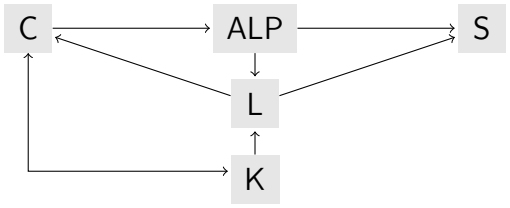


Figure 4.1: *The overall diagram of the components of the network. On the left we have the counter C that outputs all candidate explanations to the abductive logic program ALP. The output from ALP is monitored by component L to detect when an explanation has been found. If so, L instructs S to record the output of ALP, and informs C that the candidate is valid. On the bottom of the diagram is the clock K that synchronizes the system and stops when C has generated all candidates.*

4.1. Abductive Logic Program

Following [15] we start by constructing an abductive logic program from an abductive framework, which can then be transformed into a neural network using the Core method.

Definition 40 (Abductive logic program)**:**
Let $AF = \langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$ be an abductive framework, $\mathcal{O}$ an observation, and let obs and ic be two fresh predicate symbols not occurring in $\mathcal{P}$. We construct the following sets of clauses:

- $\mathcal{O}' = \{obs \leftarrow L_1, \ldots, L_n\}$, *given that* $\mathcal{O} = \{L_1, \ldots, L_n\}$,
- $\mathcal{IC}' = \{ic \leftarrow L_1, \ldots, L_n \mid \mathsf{U} \leftarrow L_1, \ldots, L_n \in \mathcal{IC}\}$

Then, the abductive logic program *ALP corresponding to $AF = \langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$:*

$$ALP = \mathcal{P} \cup \mathcal{IC}' \cup \mathcal{O}'$$

As a remark, it is usual in the literature to refer to an abductive framework and an abductive logic program as the same concept. In this Thesis, the abductive logic program is the logic program obtained from an abductive framework, that includes the observation and the integrity constraints. The reason for constructing ALP is to use the predicates *obs* and *ic* to detect if the observation $\mathcal{O}$ follows from the program $\mathcal{P}$ and an explanation $\mathcal{E}$ and, whether an integrity constraint is violated. We formalize these properties of ALP in the following three propositions:

Given an abductive framework $\mathsf{AF} = \langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$, an observation $\mathcal{O}$, the associated abductive logic program ALP and a formula F, the following holds:

Proposition 6:

$$\mathcal{P} \models_{wcs} F \text{ iff } ALP \models_{wcs} F$$

Proposition 7:

$$\mathcal{P} \models_{wcs} \mathcal{O} \text{ iff } \mathit{ALP} \models_{wcs} \mathit{obs}$$

Proposition 8:
Whenever an integrity constraint in $\mathcal{IC}$ is violated by $\mathcal{P}$, then $\mathit{ALP} \models_{wcs} ic$.

It is straightforward from the construction of ALP that Proposition 6, Proposition 7 and Proposition 8 hold.

Corollary 1:
Given $\mathit{AF} = \langle \mathcal{P}, \mathcal{A}_{\mathcal{P}}, \mathcal{IC}, \models_{wcs} \rangle$ and an observation $\mathcal{O}$, we can obtain the corresponding core $\mathsf{Net}(\mathit{ALP})$ using the Core *method. From Proposition 2, Proposition 6, Proposition 7 and Proposition 8, the associated recurrent network $\mathsf{Net}(\mathit{ALP})^{\circlearrowleft}$ will reach a stable state where:*

- *The least fixed point of Φ_{ALP} can be obtained from the output layer of $\mathsf{Net}(\mathit{ALP})^{\circlearrowleft}$.*
- *The unit obs will be active in the output layer of $\mathsf{Net}(\mathit{ALP})^{\circlearrowleft}$, whenever the observation $\mathcal{O}$ follows from the program $\mathcal{P}$.*
- *The unit ic will be active in the output layer of $\mathsf{Net}(\mathit{ALP})^{\circlearrowleft}$, whenever an integrity constraint in $\mathcal{IC}$ is violated by program $\mathcal{P}$.*

Example 4.1.1:
Recall Example 2.3.1 on page 16 :

$$\mathcal{P}_{\mathit{Train}} = \left\{ \begin{array}{lcl} \mathit{Train} & \leftarrow & \mathit{Train}_A \\ \mathit{Train} & \leftarrow & \mathit{Train}_B \\ \mathit{Take\_train} & \leftarrow & \mathit{Train}_A, \neg \mathit{Full} \\ \mathit{Full} & \leftarrow & \bot \end{array} \right\}$$

Applying the above mentioned transformation to $\mathcal{P}_{\mathit{Train}}$ yields:

$$\mathsf{ALP} = \mathcal{P}_{\mathit{Train}} \cup \left\{ \begin{array}{lcl} \mathit{obs} & \leftarrow & \mathit{Train} \\ \mathit{ic} & \leftarrow & \mathit{Train}_A, \mathit{Train}_B \end{array} \right\}$$

The corresponding recurrent network $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$ is shown in Figure 4.2.

Up to this point, we have shown how to construct a network computing $\Phi^{\mathsf{lfp}}_{\mathsf{ALP}}$ given an abductive logic program ALP. The next step is how to add candidate explanations $\mathcal{C}$ to ALP. In order to determine if $\mathcal{C}$ is indeed an explanation for $\mathcal{O}$, we need to compute $\Phi^{\mathsf{lfp}}(\mathsf{ALP} \cup \mathcal{C})$. We want to avoid constructing a new network for each candidate $\mathcal{C}$ to obtain $\Phi^{\mathsf{lfp}}(\mathsf{ALP} \cup \mathcal{C})$. Instead, we construct the

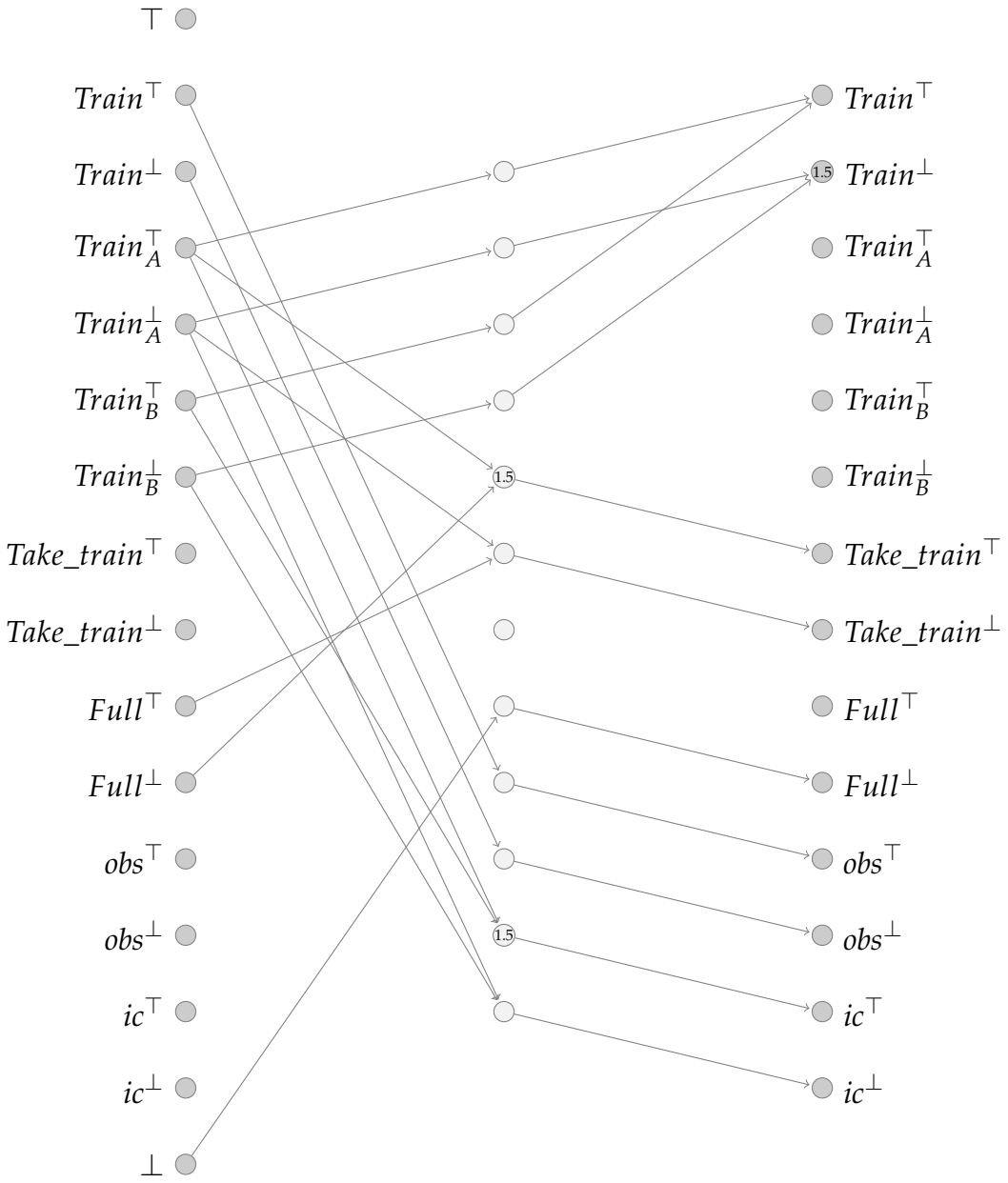


Figure 4.2: *Figure 4.2 shows the core* $\mathsf{Net}(ALP)^{\circlearrowleft}$ *obtained from program ALP, for program* $\mathcal{P}_{Train}$*. Units in the input and output layers represent atoms. Units in the hidden layer represent rules. All units have a threshold value of 0.5, unless stated otherwise. Recurrent connections from the output to the input layer are not shown.*

network for ALP and manipulate its input layer, in such a way that we can test each candidate without modifying Net(ALP).

From the construction of Net(ALP) we know that the active units in the input layer of Net(ALP) represent an input interpretation I for the corresponding Φ_{ALP}. The idea is to use this property to modify the result of each iteration of Φ_{ALP} in a way that it reaches $\Phi^{\mathsf{lfp}}_{\mathsf{ALP}\cup\mathcal{C}}$. First we need to impose a restriction on the candidate explanations:

Definition 41 (Contradictory candidate)**:**
We say that candidate explanation $\mathcal{C}$ is contradictory *if there exists an atom A for which we find that $\{A \leftarrow \bot, A \leftarrow \top\} \in \mathcal{C}$.*

Discriminating contradictory candidate explanations is useful, because we find that iterating $\Phi_{\mathcal{P}}$, and adding a non-contradictory $\mathcal{C}$ to the input and output of each iteration, is equivalent to iterate $\Phi_{\mathcal{P}\cup\mathcal{C}}$.

Proposition 9:
Consider an abductive framework $AF = \langle \mathcal{P}, \mathcal{A}_{\mathcal{P}}, \mathcal{IC}, \models_{wcs}\rangle$, an observation $\mathcal{O}$ and a non-contreadictory candidate explanation $\mathcal{C}$. Let $\mathsf{mod}\,\Phi^{\mathsf{lfp}}_{\mathcal{P}}$ be the least fixpoint of $\Phi_{\mathcal{P}}$ reached by adding $\mathcal{C}$ to the input and output interpretations of each iteration. Then the following holds:

$$\mathsf{mod}\,\Phi^{\mathsf{lfp}}_{\mathcal{P}} = \Phi^{\mathsf{lfp}}_{\mathcal{P}\cup\mathcal{C}}$$

Proof. We need to show that the iteration of $\Phi_{\mathcal{P}}$ modified by adding the positive/negative facts in $\mathcal{C}$ to the input and output of $\Phi_{\mathcal{P}}$, corresponds to $\Phi^{\mathsf{lfp}}_{\mathcal{P}\cup\mathcal{C}}$. Consider $\mathcal{C}$ contains a positive fact of the form $A \leftarrow \top$. The effect of adding $A \leftarrow \top$ to $\mathcal{P}$ is: all occurrences of A in a clause in $\mathcal{P}$ will be mapped to true and, as a result of the first rule of Φ, atom A will be mapped to true in every subsequent iteration of $\Phi^{\mathsf{lfp}}_{\mathcal{P}}$. It is easy to see that this also holds for the modified iteration. Consider now $\mathcal{C}$ contains a negative fact with atom B as its head. Since B is an abducible, it is not the head of any other clause in $\mathcal{P}$, and thus by the second rule of Φ it will be mapped to false in every subsequent iteration of $\Phi^{\mathsf{lfp}}_{\mathcal{P}}$, which is also the case in the modified iteration. Initially, it would seem sufficient to add $\mathcal{C}$ only to the input of $\Phi_{\mathcal{P}}$, but since adding abducibles to $\mathcal{P}$ implies that each abducible is now a head of a clause, they are assigned a value by $\Phi_{\mathcal{P}}$, whereas if we only add them to the input of $\Phi_{\mathcal{P}}$ they could remain unknown. That is why we add each abducible also to the output of $\Phi_{\mathcal{P}}$. Regarding contradictory candidates, consider $\mathcal{C}$ containing a positive and a negative fact with the same head A. In this case, A will be only mapped to true by Φ, whereas our modified version would map it to both: true and false. This is the reason for restricting to non-contradictory candidates. □

We will see that our approach automatically discards contradictory candidates, because explanations are found in order of minimality and non-minimal candidates are not tested. Since every minimal explanation is non-contradictory, they are skipped. Now we formalize how to test candidate explanations on $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$:

Proposition 10:
Let $AF = \langle \mathcal{P}, \mathcal{A}_{\mathcal{P}}, \mathcal{IC}, \models_{wcs} \rangle$ be an abductive framework, $\mathcal{O}$ an observation, $\mathcal{C}$ a non-contradictory candidate explanation and $\mathsf{Net}(ALP)^{\circlearrowleft}$ the recurrent network associated to ALP. If we keep unit $A^{\top}$ active for each $A \leftarrow \top \in \mathcal{C}$ and keep unit $A^{\bot}$ active for each $A \leftarrow \bot \in \mathcal{C}$, in the input and output layer of $\mathsf{Net}(ALP)^{\circlearrowleft}$, it holds that :
The relaxed network $\mathsf{Net}(ALP)^{\circlearrowleft}$ reaches a stable state that corresponds to $\Phi^{\mathsf{lfp}}_{ALP \cup \mathcal{C}}$.

Proof. From Proposition 9 we have that adding the positive and negative facts of a candidate explanation $\mathcal{C}$ to the input and output interpretations of each iteration of Φ_{ALP}, corresponds to $\Phi^{\mathsf{lfp}}_{\mathsf{ALP} \cup \mathcal{C}}$. From the construction of $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$, we know that active units in the input layer, represent the input interpretation for Φ_{ALP}. Consequently keeping units active in the input layer of $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$ is equivalent to add them to the current input interpretation. The output layer of Φ_{ALP} behaves in the same manner. □

As we have seen, the input and output layers of network $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$ provide a way to add a candidate explanation $\mathcal{C}$ to ALP. The output layer of network $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$ allows us to decide whether $\mathcal{C}$ is a valid explanation, and determine its consequences. This can be done by monitoring the value of each unit in the output layer of $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$.
With the abductive framework encoded and transformed into a neural network, our task now is to manipulate the units in $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$. Since units can only have two values: active/passive, the rest of the components of the network are specified using two-valued logic programs. This seems not only a natural choice, but from our point of view its also simpler.

4.2. The Counter C: Obtaining only Minimal Explanations

In this section we present a component, called the counter, that given an abductive framework $\mathsf{AF} = \langle \mathcal{P}, \mathcal{A}_{\mathcal{P}}, \mathcal{IC}, \models_{wcs} \rangle$ and an observation $\mathcal{O}$, generates all minimal explanations for $\mathcal{O}$. The idea is to use the counter to first activate the units on the input layer of $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$, representing each candidate explanation $\mathcal{C}$ and then, obtain feedback from the output layer of $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$ to determine if the candidate $\mathcal{C}$ is a valid explanation. Accordingly, the counter is composed of two components: the first obtains all candidate explanations in order of minimality, corresponding to the ordered set of candidate explanations $Ord(\mathcal{A}_{\mathcal{P}})$ introduced in Chapter 3,

Definition 39. The second component discards all those who can not be minimal. As a result, only minimal explanations will be tested in $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$.

We begin presenting an algorithm to obtain the ordered set of all candidate explanations $Ord(\mathcal{A}_\mathcal{P})$ given $\mathcal{A}_\mathcal{P}$. Then the algorithm is encoded as a logic program, and the logic program is finally translated into a neural network. As part of this Thesis, a multi-platform application was developped using Java and Prolog to compute $T_\mathcal{P}$ and Φ semantic operators. The example on Table 4.1 was encoded and tested using the afore mentioned software, on which the expected results were confirmed. Details about the tool designed, can be found in the Annex.

As explained in Section 4.1, we are interested in changing/monitoring the state of specific units in $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$. This state can either be active or passive. For this reason the components described in this section are encoded as two-valued logic programs. This implies that when constructing a network from them, we use the two-valued version of the CORE method (page 13). Likewise, when referring to fixpoints and iterations, we refer to the two-valued semantic operator $T_\mathcal{P}$ (page 9).

4.2.1. Generating Ordered Candidates

Recall the set $Ord(\mathcal{A}_\mathcal{P})$. It contains all possible candidate explanations for a given observation $\mathcal{O}$, ordered according to $\preceq$. Here we are interested in obtaining **only and all minimal explanations**, thus we test each of the candidates $\mathcal{C} \in Ord(\mathcal{A}_\mathcal{P})$ for minimality. As in Algorithm 1 we discover explanations in order of minimality, when we find an explanation $\mathcal{E}$ we can discard all other candidates $\mathcal{C} \supset \mathcal{E}$, and continue searching for other minimal explanations.

But since we are given only the abductive framework AF, we need to obtain the set $Ord(\mathcal{A}_\mathcal{P})$ from it. Algorithm 3 computes $Ord(\mathcal{A}_\mathcal{P})$ given the set of abducibles $\mathcal{A}_\mathcal{P}$.

Before we turn to the algorithm, let us introduce some useful notation:

Definition 42 (Bits)**:**
A bit *v (possibly indexed) is a variable which can only have two values: 0 and 1. A bit is said to be* active *if its value is 1, and* passive *if it is 0. A bit* flips *if it changes its value in two consecutive time steps.*

Definition 43 (Binary vector)**:**
A binary vector *(or simply a vector) $\vec{v} = \{v_1, \ldots, v_n\}$ is an ordered set of n bits, where $v_1 < v_2 < \cdots < v_n$. The* right-most bit *is the highest bit in $\vec{v}$. A bit v_i can be* shifted*, meaning that the new position of bit v_i in $\vec{v}$ is $i+1$. And the new position of bit $v_{(i+1)}$ in $\vec{v}$ is i. A bit v_i can be shifted multiple times until it reaches position j, by v_{ij} we represent that the bit originally in position i is currently in position j.*

As an example consider a binary vector:

$$\vec{v} = \{v_1, v_2, v_3, v_4\}$$

where v_{11}, v_{22}, v_{33} and v_{44} represent that each bit is in its original position. If we shift v_1 once, the vector becomes:

$$\vec{v} = \{v_2, v_1, v_3, v_4\}$$

where bits v_1 and v_2 exchange positions, represented by v_{12} and v_{21}. Shifting v_1 once again we have:

$$\vec{v} = \{v_2, v_3, v_1, v_4\}$$

where bits v_1 and v_3 exchange now positions, represented by v_{13} and v_{32}. Shifting v_1 once more yields:

$$\vec{v} = \{v_2, v_3, v_4, v_1\}$$

represented by v_{21}, v_{32}, v_{43} and v_{14}.

We are interested in binary vectors, because a candidate explanation $\mathcal{C}$ can be represented as a binary vector of length $n : \vec{v} = \{v_1, \ldots, v_n\}$ where each bit represents one abducible in $\mathcal{A}_\mathcal{P}$, and $n = |\mathcal{A}_\mathcal{P}|$. An active bit v_i indicates that the corresponding i-th abducible in $\mathcal{A}_\mathcal{P}$ belongs to $\mathcal{C}$. The number of active bits, given a vector $\vec{v}$ is denoted by k. We can refer to $Ord(\mathcal{A}_\mathcal{P})$ as a set of binary vectors of length n, each one representing one candidate explanation $\mathcal{C}$. As an example consider:

Example 4.2.1:
Given the set of abducibles:

$$\mathcal{A}_\mathcal{P} = \{p \leftarrow \top, p \leftarrow \bot, q \leftarrow \top, q \leftarrow \bot\}$$

an ordered set of all candidate explanations is:

$$\begin{aligned} Ord(\mathcal{A}_\mathcal{P}) = \{&\emptyset, \{p \leftarrow \top\}, \{p \leftarrow \bot\}, \{q \leftarrow \top\}, \{q \leftarrow \top\}, \{p \leftarrow \top, q \leftarrow \top\}, \\ &\ldots, \{p \leftarrow \top, p \leftarrow \bot, q \leftarrow \top, q \leftarrow \bot\}\} \end{aligned}$$

their binary representation is:

$k = 2, \vec{v} = \{1,0,1,0\}$ for $C = \{p \leftarrow \top, q \leftarrow \top\}$, or
$k = 2, \vec{v} = \{1,0,0,1\}$ for $C = \{p \leftarrow \top, q \leftarrow \bot\}$, or
$k = 4, \vec{v} = \{1,1,1,1\}$ for $C = \{p \leftarrow \top, p \leftarrow \bot, q \leftarrow \top, q \leftarrow \bot\}$.

Using this notation, we can obtain all candidates $|C| = 1$ by generating all vectors of length n, where only one bit is active ($k = 1$), all candidates $|C| = 2$ by generating all vectors of length n with two active bits ($k = 2$), and so on.

In order to manipulate bits and their positions, we need to differentiate between the original position of a bit, and the current one. Function *pos* tells us the current position of a given bit, as well as the opposite: given a position in a vector, it will tell us the bit occupying that position:

Definition 44 (Function *pos*)**:**
Given a binary vector $\vec{v} = \{v_1, \ldots, v_n\}$ *with* $n = |\vec{v}|$*:*

- $pos(v_i) = j$ *iff the current position of bit* v_i *in* $\vec{v}$ *is* j*.*
- $pos^{-1}(j) = v_i$ *iff the position* j *is currently occupied by bit* v_i*.*

If $j > |\vec{v}|$*, then* $pos^{-1}(j) = 0$*.*

Now we are ready to introduce the algorithm:

Input: $\mathcal{A}_\mathcal{P}$
Output: $Ord(\mathcal{A}_\mathcal{P})$

$n = |\mathcal{A}_\mathcal{P}|$
$\vec{v} = \{v_1, \ldots, v_n\}$ where $v_i = 0$ for $1 \leq i \leq n$
$\vec{pos} = \{pos^{-1}(1), \ldots, pos^{-1}(n)\}$
$Ord(\mathcal{A}_\mathcal{P}) = \emptyset$
$i = 0$
for $k = 1 \ldots n$ **do**
    $v_k = 1$ *activate k-th bit*
    $restart = false$
    for $j = 1 \ldots n$ **do**
        $pos(v_j) = j$ *set all bits to their original positions*
    end
    add $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$ *add vector with current positions to* $Ord(\mathcal{A}_\mathcal{P})$
    $i = k$ *get current highest bit position*
    while *(!restart)* **do**
        if $pos(v_i) = 0$ **then**
            $restart = true$
        else
            if $(v_{pos(v_i)+1} = 0)$ *and* $(pos(v_i)! = n)$ **then**
                $pos(v_i) = pos(v_i) + 1$
                $shift\_val = pos(v_i) - i$
                for $i < m < n$ **do**
                    $pos(v_m) = m + shift\_val$ *reset position of higher bits*
                end
                add $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$
            else
                $v_i = v_{i-1}$
            end
        end
    end
end

Algorithm 3: The computation of the set $Ord(\mathcal{A}_\mathcal{P})$.

Algorithm 3 uses two binary vectors to obtain $Ord(\mathcal{A}_\mathcal{P})$: $\vec{v} = \{v_1, \ldots, v_n\}$ contains the original positions of the bits to be combined, and vector $\vec{pos} = \{pos_1, \ldots, pos_n\}$ holds the current positions (possibly shifted) of those bits. For each vector $\vec{v}$ Algorithm 3 will obtain all possible combinations of its bits. Each combination correspond to a $\vec{pos}$ vector.

In Algorithm 3 the **for** cycle (6) increases by 1 the number k of active bits in $\vec{v}$. For each $\vec{v}$ the **while** loop (14) obtains all different combinations $\vec{pos}$ that can be made with those bits. This **while** loop shifts bit v_i one position each time it is executed, and adds the current combination $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$. When the right-most bit v_i reaches the n-th position, it can no longer be shifted (18). Consequently the algorithm tries to shift the previous right-most active bit (26), until there are no more bits to be shifted (15). At this point, the *restart* variable is set to true (16), ending the **while** cycle. Then, the **for** cycle (6) increases k (the number of active bits in $\vec{v}$) and the process restarts.
To ensure that all combinations are generated, when a bit v_i is shifted, all higher bits must be repositioned relative to v_i. Consequently, whenever a bit is shifted Algorithm 3 obtains the number of positions shifted (20). This value ($shift\_val$) is added to the original position of each bit higher than v_i (22). This is the reason why we keep a vector $\vec{v}$ with the original positions. In the following, we will refer to the **for** cycle in line 6 of Algorithm 3 as simply the **for** cycle. Algorithm 3 can be further clarified with an example:

k	*restart*	$\vec{v}$	$\vec{pos}$	$Shift(v_i)$
		$\downarrow$		
1	*false*	1 0 0 0	v_1 0 0 0	add $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$
			0 v_1 0 0	$Shift(v_1) : pos(v_1) = 2, Reset(1)$, add $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$
			0 0 v_1 0	$Shift(v_1) : pos(v_1) = 3, Reset(1)$, add $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$
			0 0 0 v_1	$Shift(v_1) : pos(v_1) = 4, Reset(1)$, add $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$
	true			$Shift(v_1) : Shift(v_0) : restart = true$
		$\downarrow$		
2	*false*	1 1 0 0	v_1 v_2 0 0	add $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$
			v_1 0 v_2 0	$Shift(v_2) : pos(v_2) = 3, Reset(2)$, add $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$
			v_1 0 0 v_2	$Shift(v_2) : pos(v_2) = 4, Reset(2)$, add $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$
			0 v_1 v_2 0	$Shift(v_2) : Shift(v_1) : pos(v_1) = 2,$ $Reset(1) : pos(v_2) = 3$, add $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$
			0 v_1 0 v_2	$Shift(v_2) : pos(v_2) = 4, Reset(2)$, add $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$
			0 0 v_1 v_2	$Shift(v_2) : Shift(v_1) : pos(v_1) = 3,$ $Reset(1) : pos(v_2) = 4$, add $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$
	true			$Shift(v_2) : Shift(v_1)) : Shift(v_0) : restart = true$
		$\downarrow$		
3	*false*	1 1 1 0	v_1 v_2 v_3 0	add $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$
			v_1 v_2 0 v_3	$Shift(v_3) : pos(v_3) = 4, Reset(3)$, add $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$
			v_1 0 v_2 v_3	$Shift(v_3) : Shift(v_2) : pos(v_2) = 3,$ $Reset(2)$, add $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$
			0 v_1 v_2 v_3	$Shift(v_3) : Shift(v_2) : Shift(v_1) : pos(v_1) = 2,$ $Reset(1)$, add $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$
	true			$Shift(v_3) : Shift(v_2) : Shift(v_1) :$ $Shift(v_0) : restart = true$
		$\downarrow$		
4	*false*	1 1 1 1	v_1 v_2 v_3 v_4	add $\vec{pos}$ to $Ord(\mathcal{A}_\mathcal{P})$
	true			$Shift(v_4) : Shift(v_3) Shift(v_2) : Shift(v_1) :$ $Shift(v_0) : restart = true$

Table 4.1: *Computation of $Ord(\mathcal{A}_\mathcal{P})$ by Algorithm 3, with $n = |\mathcal{A}_\mathcal{P}| = 4$*

Table 4.1 shows how Algorithm works. The first four columns show the values of k, *reset* and the vectors $\vec{v}$ and $\vec{pos}$. Each row in the first column corresponds to one cycle of the **for** loop, where the value of k is incremented, and the initial vector $\vec{pos}$ is added to $Ord(\mathcal{A}_\mathcal{P})$. For each such **for** cycle we have several **while** loops each one shifting the highest bit once, until $restart = true$. The fifth column, shows the values yielded by each cycle of the **while** loop in line 6 of Algorithm 3. The arrow $\downarrow$ on vector $\vec{v}$ shows the current bit being shifted.

Encoding Algorithm 3 in a Logic Program

To obtain a neural network corresponding to Algorithm 3, we first need to express it as a logic program. To encode Algorithm 3 we construct a logic program divided in two components:

1. C_1 corresponds to the **while** loop in Algorithm 3.
2. C_2 corresponds to the **for** loop in Algorithm 3, which increases k.

Component C_1 uses the following predicate symbols:

- v_{ij} represents the i-th bit at position j.
- s_i states that the bit to be shifted is v_i.
- δ tells C_1 to make a shift.
- try_i indicates that we are trying to shift bit v_i.
- $reset_i$ indicates that *reset* signal has been fired by bit v_i.
- *restart* indicates C_1 has produced all combinations of vector with k active bits (and C_2 is asked to increase k).
- k_i indicates bit v_i is active in $\vec{v}$.

$$\mathsf{C}_1 = \bigcup_{i,j=1}^{n} \left\{ \begin{array}{lll} (1)\ v_{i(j+1)} & \leftarrow & v_{ij}, s_i, \delta \\ (2)\ try_{i-1} & \leftarrow & v_{in}, s_i, \delta \\ (3)\ try_{i-1} & \leftarrow & v_{ij}, v_{(i+1)(j+1)}, try_i \\ (4)\ reset_i & \leftarrow & v_{ij}, \neg v_{(i+1)(j+1)}, try_i \\ (5)\ v_{i(j+1)} & \leftarrow & v_{ij}, \neg v_{(i+1)(j+1)}, try_i \\ (6)\ v_{m(j+m-i)} & \leftarrow & reset_i, v_{ij}, k_m (\text{ for m>i}) \\ (7)\ v_{ij} & \leftarrow & v_{ij}, \neg v_{i(j+1)}, \neg reset_m, \neg restart\ (0 \leq \text{m} < \text{i}) \\ (8)\ restart & \leftarrow & try_0 \end{array} \right\}$$

In C_1, an atom v_{ij} is *true*, only when bit v_i is active in position j. Atom s_i tells us which bit needs to be shifted, as a result when signal δ is active, (1) will shift bit v_{ij} to position $j+1$. In the case bit v_i is already in the last position n, (2) will try to shift the previous active bit. (3) will recursively try to shift the previous bit if the current one can't be shifted. On the other hand, if the bit v_{ij} can be shifted to position $j+1$, meaning that bit $v_{(i+1)(j+1)}$ is passive, (4) will fire the corresponding $reset_i$ signal. (6) will reposition all bits higher than v_i. (7) serves as a memory, stating that each bit holds its value if:

1. It has not been shifted to the next position,
2. there is no *reset* signal from a lower bit, and
3. there is no *restart* signal.

Finally, if there are no more bits to shift, try_0 will be active and (8) fires the *restart* signal, asking C_2 to update $\vec{v}$, by incrementing k.
Component C_2 uses the additional predicate symbols:

- ψ tells C_2 to obtain the next vector $\vec{v}$.
- *sync* is an external signal, which tells the counter C to generate the next vector $\vec{pos}$, by either activating δ signal (**while** loop) or ψ signal (**for** loop).
- *done* is used to detect when all vectors $\vec{pos}$ have been generated.

$$\mathsf{C}_2 = \bigcup_{i,j=1}^{n} \left\{ \begin{array}{lll} (9) & k_1 & \leftarrow \\ (10) & k_i & \leftarrow k_i \\ (11) & k_{i+1} & \leftarrow k_i, restart \\ (12) & done & \leftarrow b_n, restart \\ (13) & \psi & \leftarrow \neg\psi, sync \\ (14) & \delta & \leftarrow \psi, sync \\ (15) & v_{ii} & \leftarrow \neg\psi, k_i, sync \\ (16) & \psi & \leftarrow \psi, \neg reset \\ (17) & s_{i+1} & \leftarrow s_i, reset \\ (18) & s_i & \leftarrow s_i, \neg s_{i+1} \ldots, \neg s_n \\ (19) & s_1 & \leftarrow \neg k_2, \neg\psi, sync \\ (20) & pos_j & \leftarrow v_{ij} \\ (21) & done & \leftarrow done \end{array} \right\}$$

In C_2, clauses (9)(10) and (11) increase the value of k, yielding the next $\vec{v}$. (9) initializes $k=1$, (10) ensures each bit in $\vec{v}$ remains active, and (11) increments k. When all bits in $\vec{v}$ are active (12) forces *done* to be *true*, indicating that we have generated all possible combinations, and we can stop the computations. The *sync*

signal is used to obtain the next combination $\vec{pos}$. If signal ψ is not active, a loop of the **for** cycle must be executed. Whereas if we are already in the **for** loop, one cycle of the **while** loop must be executed. (15) initializes the position of each bit in $\vec{v}$. (16) indicates that we remain in the **for** cycle (ψ) until the **while** loop finishes, setting *reset* to *true*. (17) updates the right-most bit (s_i) in each **for** cycle. (18) ensures that there is only one right-most bit at each time, and (19) initializes the bit in position 1 as the first right-most bit, as soon as we enter the **for** cycle. (21) ensures that signal *done* stays active after all combinations were generated. This signal tells us when the entire process has ended.

The $T_{\mathcal{P}}$ operator associated to program $\mathsf{C} = \mathsf{C}_1 \cup \mathsf{C}_2$ reaches multiple fixpoints, each one induced by the signal *sync*. At each fixpoint, one of the elements of $Ord(\mathcal{A}_{\mathcal{P}})$ is computed, represented by $\vec{pos}$.

Proposition 11:
Let $\mathsf{Net}(C_1 \cup C_2)^{\circlearrowleft}$ *be the recurrent two-valued core computing the* $T_{\mathcal{P}}$ *operator associated to program* $C_1 \cup C_2$*. Then the relaxed network* $\mathsf{Net}(C_1 \cup C_2)^{\circlearrowleft}$ *reaches multiple stable states, each one induced by activating the unit sync in its input layer, for two consecutive timesteps. On each such stable state, one distinct element of* $Ord(\mathcal{A}_{\mathcal{P}})$ *can be obtained, represented by* $\vec{pos}$*.*

Proof. As initially $\mathsf{Net}(\mathsf{C}_1 \cup \mathsf{C}_2)^{\circlearrowleft}$ is relaxed, its input layer represents the empty interpretation $I = \emptyset$. We prove that the associated $T_{\mathcal{P}}$ reaches multiple fixpoints starting from I:
Let $T_{\mathcal{P}}$ be the operator associated to program $\mathsf{C}_1 \cup \mathsf{C}_2$. We have:

1. $T_{\mathcal{P}}(\emptyset) = b$ is the first fixpoint of $T_{\mathcal{P}}$ (since in clause (9) of C_2 we have the positive fact $b \leftarrow \top$).

2. Activating the unit *sync* two consecutive timesteps, is equivalent to add *sync* to the current input interpretation I: $T_{\mathcal{P}}(I \cup \{sync\}) = J$. The resulting interpretation J has the property that $sync \notin J$, since there is no clause in $\mathsf{C}_1 \cup \mathsf{C}_2$ with *sync* as its head. As $(I \cup \{sync\}) \neq J$, $I \cup \{sync\}$ is not a fixpoint.

3. When we force $T_{\mathcal{P}}$ to leave the current fixpoint I, by activating *sync*, it will converge to a new fixpoint J: For any further I' resulting from iterating $T_{\mathcal{P}}(I \cup \{sync\})$, we have that $sync \notin I'$. This means that for all clauses $A \leftarrow body \in \mathsf{C}_1 \cup \mathsf{C}_2$ with $sync \in body$ we find $I'(body) = \bot$. That is, the value of no atom can be affected by these clauses (13,14,15,19) anymore. If we remove them from $\mathsf{C}_1 \cup \mathsf{C}_2$ the iteration process is not affected. The resulting program obtained from removing these causes, has no negative cycles.

4. Let I, J, J' be interpretations and $\mathcal{P}$ a program, such that $T_{\mathcal{P}}(I) = J$ and $T_{\mathcal{P}}(J) = J'$. Then, if we add $A \leftarrow \top$ for each $A \in J$ to $\mathcal{P}$ we have $T_{\mathcal{P}}(\emptyset) = J'$. That is, adding all atoms in the last iteration as facts to the program, is equivalent to compute $T_{\mathcal{P}}(\emptyset)$ in the modified program.

5. From the above considerations when $T_{\mathcal{P}}$ leaves a fixpoint I induced by *sync* is equivalent to iterate $T_{\mathcal{P}}(\emptyset)$ on the program modified by adding the elements in I as facts, and removing the clauses depending on *sync*. Since the modified program contains no negative cycles, it is ensured the existence of a least fixpoint.

We conclude that activating the unit *sync* in the input layer of the stable network $\mathsf{Net}(\mathsf{C}_1 \cup \mathsf{C}_2)^{\circlearrowleft}$ for two consecutive timesteps, forces it to leave its current state and reach a new stable state in finite time. On each such stable state, the units pos_i in the output of $\mathsf{Net}(\mathsf{C}_1 \cup \mathsf{C}_2)^{\circlearrowleft}$ represent a candidate explanation $\mathcal{C}$. Given that $\mathsf{C}_1 \cup \mathsf{C}_2$ is an encoding of Algorithm 3, all of them are distinct and generated in order of minimality. □

Lemma 4:
Let $\mathsf{Net}(C_1 \cup C_2)^{\circlearrowleft}$ *be the recurrent two-valued core computing the* $T_{\mathcal{P}}$ *operator associated to program* $C_1 \cup C_2$*, after all candidate explanations* $\mathcal{C} \in \mathcal{A}_{\mathcal{P}}$ *have been generated. Unit done in the output layer of* $\mathsf{Net}(C_1 \cup C_2)^{\circlearrowleft}$ *remains active.*

Proof. Let $n = |\mathcal{A}_{\mathcal{P}}|$. From Lemma 6, if the last candidate is represented by the output of $\mathsf{Net}(\mathsf{C}_1 \cup \mathsf{C}_2)^{\circlearrowleft}$ all units pos_i and all units $v_i i$ with $1 \leq i \leq n$ are active (representing $\mathcal{C}$). This means that all bits in $\vec{pos}$ are active, and thus no shift can be done. From rules (2,3,8) in C_1, if no further shift can be done unit *reset* will be active. The combination pos_n and *reset*, both active is detected by clause (12) in C_2, thus activating unit *done*. Once *done* is active, it will indefinitely remain active because of clause (21) in C_2. □

4.2.2. Minimality

Given $\mathsf{C}_1 \cup \mathsf{C}_2$ we can obtain all candidate explanations in order of minimality. Now the task is to somehow filter these candidates, in such a way that all those that can not represent a minimal explanation are discarded. The logic program M has the tasks to filter the output of $\mathsf{Net}(\mathsf{C}_1 \cup \mathsf{C}_2)^{\circlearrowleft}$ such that only minimal candidates are tested in $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$ and to record each minimal explanation that has been found.
First, we need to determine when a minimal explanation $\mathcal{E}$ is contained in the current candidate $\mathcal{C}$. If this condition holds, $\mathcal{C}$ can not be a minimal explanation, and can be discarded. We assume that $\mathcal{E}$ is contained in $\mathcal{C}$, unless there is an

element in $\mathcal{E}$ not present in $\mathcal{C}$. The following definition allows us o detect this combination:

Definition 45:
Given the set of abducibles $\mathcal{A}_\mathcal{P}$, a candidate $\mathcal{C}$ and a minimal explanation $\mathcal{E}$, we define for each abducible $A_i \in \mathcal{A}_\mathcal{P}$:

1. $q_i = \bot$ *iff* $A_i \in \mathcal{E}$ *and* $A_i \notin \mathcal{C}$
2. $q_i = \top$ *otherwise.*

We refer to the set of all q_i depending on $\mathcal{C}$ and $\mathcal{E}$ as $\mathcal{Q}_{\mathcal{CE}} = \{q_1, \ldots, q_n\}$.

Lemma 5:
Given an explanation $\mathcal{E}$ a candidate $\mathcal{C}$ where $\mathcal{E} \subseteq \mathcal{C}$ it holds that: $q_i = \top$ for all $q_i \in \mathcal{Q}_{\mathcal{EC}}$.

Proof. Straightforward. □

Intuitively, in case all elements of $\mathcal{E}$ are also in $\mathcal{C}$, all $q_i \in \mathcal{Q}_{\mathcal{EC}}$ will be true, indicating that $\mathcal{C}$ is a superset of $\mathcal{E}$ and thus $\mathcal{C}$ can not be minimal . The process is encoded by program M_1, where we label $A_i \in \mathcal{E}$ as $A_i^\mathcal{E}$ in order to distinguish it from $A_i \in \mathcal{C}$.
For each $\mathcal{E} \in \mathcal{M}_\mathcal{E}$ and each $A \in \mathcal{A}_\mathcal{P}$:

$$\mathsf{M}_1 = \left\{ \begin{array}{l} (1)\ q_i \leftarrow \neg A_i^\mathcal{E} \\ (2)\ q_i \leftarrow \neg A_i, A_i^\mathcal{E} \end{array} \right\} \cup \left\{ (3)\ non\_min \leftarrow q_1, \ldots, q_n \right\}$$

Lemma 6:
Given an explanation $\mathcal{E}$ and a candidate $\mathcal{C}$ where $\mathcal{E} \subseteq \mathcal{C}$. The corresponding program M_1 has a least fixpoint where: $T_\mathcal{P}^{\mathsf{lfp}}(non\_min) = \top$.

Proof. Assume $\mathcal{E} \subset \mathcal{C}$ but $T_\mathcal{P}^{\mathsf{lfp}}(non\_min) = \bot$. Only if for some $T_\mathcal{P}^{\mathsf{lfp}}(q_i) = \bot$. Then clauses (1) and (2) in M are not satisfied, only if for the corresponding A_i we find both: $A_i \in \mathcal{E}$ and $A_i \notin \mathcal{C}$. Since $A_i \in \mathcal{E}$ and $\mathcal{E} \subset \mathcal{C}$ it follows that $A_i \in \mathcal{C}$. But then $q_i = \top$ (contradiction).

□

The program M_1 requires to build a set of clauses for each minimal explanation, but how can we know how many are they in advance? We provide an upper bound for the number of minimal explanations that an abductive framework can have, depending on the number of abducibles.

Lemma 7:
Given a set of abducibles $\mathcal{A}_\mathcal{P}$, let $n = |\mathcal{A}_\mathcal{P}|$. Then the maximal number of possible minimal explanations is given by $\max = \frac{(n!)}{\frac{n}{2}!\frac{n}{2}!}$*, which is the maximum number of subsets of the same size ($\frac{n}{2}$) out of n elements.*

Proof. Let $n = |\mathcal{A}_\mathcal{P}|$, and let $\mathcal{A}^k_\mathcal{P} = \{\mathcal{E} \subseteq \mathcal{A}_\mathcal{P} \mid |\mathcal{E}| = k\}$, $dep(\mathcal{E}) = \{\mathcal{E}' | \mathcal{E}' \supset \mathcal{E}\}$ and $k = m$. We prove that if an explanation is not in $\mathcal{A}^k_\mathcal{P}$, the number m of possible minimal explanations never increases, thus the case where more explanations can exist is when all of them are of size k.
Assume $\mathcal{E}^x$ is a minimal explanation, but $\mathcal{E}^x \notin \mathcal{A}^k_\mathcal{P}$, then :
Case 1: $\mathcal{E}^x \in \mathcal{A}^x_\mathcal{P}$ for some $x < k$
Let B be the set of all possible minimal explanations and $\mathcal{E}^x \in B$. Then no element of $dep(\mathcal{E}^x)$ can be minimal, whereas if $\mathcal{E}^x$ is not a minimal explanation, the set $B \setminus \mathcal{E}^x$ can at least be extended by $dep(\mathcal{E}^x) \in \mathcal{A}^{x+1}_\mathcal{P}$. For any $x < k$ we have $|dep(\mathcal{E}^x) \in \mathcal{A}^{x+1}_\mathcal{P}| \geq 1$, thus no set B can be bigger than the set where all possible explanations are of size k.
Case 2: $\mathcal{E}^x \in \mathcal{A}^x_\mathcal{P}$ for some $x > k$
Let $dep(\mathcal{E}^x)^- = \{\mathcal{E} | \mathcal{E} \subset \mathcal{E}^x\}$ and B be the set of all possible minimal explanations, if there exists a minimal explanation $\mathcal{E}^x \in \mathcal{A}^{(x)}_\mathcal{P}$, then $dep(\mathcal{E}^x)^- \notin B$. But the set $B \setminus \mathcal{E}^x$ can at least be extended by $dep(\mathcal{E}^x)^- \in \mathcal{A}^{x-1}_\mathcal{P}$. For any $x > k$ we have that $|\mathcal{E}^x \in \mathcal{A}^{(x+1)}_\mathcal{P}| \geq 1$, thus B can never be larger than the set where all possible explanations are of size k. □

Given the upper bound max, the following program will keep track of how many explanations have been discovered, and in which order. This program uses the external unit *sol* from component L in Section 4 to discover minimal explanations (Lemma 8). Here a new predicate symbol e_i is introduced, stating that the i-th explanation $\mathcal{E} \in \mathcal{M}_\mathcal{E}$ has been found:

$$\mathsf{M}_2 = \left\{ \begin{array}{lcl} \mathsf{e}_1 & \leftarrow & sol, \neg \mathsf{e}_1 \\ \mathsf{e}_2 & \leftarrow & sol, \neg \mathsf{e}_2, \mathsf{e}_1 \\ \mathsf{e}_3 & \leftarrow & sol, \neg \mathsf{e}_3, \mathsf{e}_2 \\ \ldots & & \\ \mathsf{e}_{\max} & \leftarrow & sol, \neg \mathsf{e}_{\max}, \mathsf{e}_{\max-1} \end{array} \right\} \cup \left\{ \; \mathsf{e}_i \leftarrow \mathsf{e}_i \; \right\}$$

When the i-th explanation has been found, the correspondent unit e_i will be active, and because *sol* is active for only two time steps (component 4.3 page 48), it ensures that only one of the clauses above will succeed for each minimal explanation found.
As soon as $\mathcal{E}_i$ has been found, its elements must be recorded. The following program uses the symbol $A_i^{\mathsf{e}_j}$ to denote that an abducible A_i belongs to $\mathcal{E}_j$. Where each A_i represents the corresponding unit in the output of the counter C. For each $A_i \in \mathcal{A}_\mathcal{P}$ and each $\mathcal{E}_j$ with $1 \leq j \leq \max$, we have:

$$\mathsf{M}_3 = \left\{ \begin{array}{l} (1)\; A_i^{\mathsf{e}_j} \leftarrow A_i, \mathsf{e}_j, \neg block\_\mathsf{e}_j \\ (2)\; A_i^{\mathsf{e}_j} \leftarrow A_i^{\mathsf{e}_j} \end{array} \right\} \cup \left\{ (3)\; block\_\mathsf{e}_i \leftarrow \mathsf{e}_i \; \right\}$$

When a minimal explanation is detected by e_j, we know from Proposition 11 and Lemma 8 that the elements of $\mathcal{E}_j$ are represented by all active units A_i corresponding to an abducible, in the output of C. Additionally, e_j tells us which explanation we have found. Clause (1) uses this combination to set the corresponding $A_i^{\mathsf{e}_j}$. The additional condition $\neg block\_\mathsf{e}_j$ combined with (3) ensures that we record the values only when the $\mathcal{E}_j$ explanation was found (because of M_2 each e_j remains active). Clause (2) serves as a memory for each $A_i^{\mathsf{e}_j} \in \mathsf{e}_j$.

As a result the program $\mathsf{M}_1 \cup \mathsf{M}_2 \cup \mathsf{M}_3$ detects when a minimal explanation has been found, and records it. Connecting its core $\mathsf{Net}(\mathsf{M}_1 \cup \mathsf{M}_2 \cup \mathsf{M}_3)$ to the corresponding units in $\mathsf{Net}(\mathsf{C}_1 \cup \mathsf{C}_2)^{\circlearrowleft}$ we obtain the complete counter:

$$\mathsf{Net}(\mathsf{C}) = \mathsf{Net}(\mathsf{C}_1 \cup \mathsf{C}_2)^{\circlearrowleft} \cup \mathsf{Net}(\mathsf{M}_1 \cup \mathsf{M}_2 \cup \mathsf{M}_3)$$

Corollary 2:
Given an abductive framework $AF = \langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$ and an observation $\mathcal{O}$. The associated network $\mathsf{Net}(C) \cup \mathsf{Net}(ALP)^{\circlearrowleft} \cup \mathsf{Net}(L) \cup \mathsf{Net}(K) \cup \mathsf{Net}(S)$ has the following properties:

1. *Whenever the output of $\mathsf{Net}(C)$ represents a non minimal explanation, it is detected by unit non_min. If unit non_min is active then unit good can not be active in $\mathsf{Net}(L)$. The current candidate is discarded and $\mathsf{Net}(C)$ generates the next candidate.*

2. *Whenever unit sol is active in $\mathsf{Net}(L)$, the network has discovered a minimal explanation $\mathcal{E}$ for $\mathcal{O}$, represented by all active units corresponding to an abducible in the output of $\mathsf{Net}(C)$.*

Proof. 1) Follows from the construction of the network and Lemma 6 which states that unit *non_min* detects candidates that can not be minimal explanations.
2) From Lemma 2 we know that $\mathsf{Net}(\mathsf{C})^{\circlearrowleft}$ computes the associated $T_\mathcal{P}$. From Proposition 11 we know that for each fixpoint of $T_\mathcal{P}$, the output of $\mathsf{Net}(\mathsf{C})^{\circlearrowleft}$ represents a candidate explanation $\mathcal{C}$. From Lemma 8 we know that if unit *sol* is active, then $\mathcal{C}$ is indeed an explanation for $\mathcal{O}$, moreover from point 1) on this corollary, we know that any non-minimal candidate is discarded. We conclude that each explanation detected by *sol* is minimal. □

4.2.3. Properties of C

Finally, Proposition 12 and Proposition 13 summarizes the most relevant properties of the network obtained from counter C. These properties follow from the construction of $\mathsf{Net}(\mathsf{C})$, from Lemmata 11, 4 and Corollary 2.

Proposition 12:
Let $\mathsf{Net}(C)$ *be the network specified in Section 4.2:*

- *Each unit in the output layer of* $\mathsf{Net}(C)$*, correspond to one abducible in* $\mathcal{A}_\mathcal{P}$*.*
- *The relaxed network* $\mathsf{Net}(C)$*, reaches multiple stable states.*
- *On each such stable state, one distinct minimal explanation* $\mathcal{E} \in \mathcal{M}_\mathcal{E}$ *can be obtained, represented by the active units in its output layer.*
- *If the network* $\mathsf{Net}(C)$ *is already in a stable state. The next stable state can be reached by activating unit sync in its input layer, for two consecutive timesteps.*
- *After all possible combinations of abducibles have been generated, unit done in the output layer of* $\mathsf{Net}(C)$ *will remain active.*

Proposition 13:
Let $\mathsf{Net}(C) \cup \mathsf{Net}(ALP)^{\circlearrowleft}$ *be the neural network obtained from connecting the output of component* $\mathsf{Net}(C)$ *to the corresponding units in* $\mathsf{Net}(ALP)^{\circlearrowleft}$*. The relaxed network* $\mathsf{Net}(C) \cup \mathsf{Net}(ALP)^{\circlearrowleft}$*, reaches multiple stable states. On each such stable state it holds:*

1. *A minimal explanation* $\mathcal{E}$ *for* $\mathcal{O}$ *is represented by the output layer of* $\mathsf{Net}(C)$*.*
2. *For each literal* L *represented by an active unit in the output of* $\mathsf{Net}(ALP)^{\circlearrowleft}$*, we have* $\mathcal{P} \cup \mathcal{E} \models_{wcs} L$*.*

Proof. (1) From Proposition 12 we know that $\mathsf{Net(C)}$ reaches multiple stable states. On each stable state its output layer represents a minimal explanation $\mathcal{E}$ for $\mathcal{O}$. If we connect the output of $\mathsf{Net(C)}$ to $\mathsf{Net(ALP)}^{\circlearrowleft}$, by Proposition 10 $\mathsf{Net(ALP)}^{\circlearrowleft}$ will compute Φ^{lfp}, and $\mathsf{Net(C)} \cup \mathsf{Net(ALP)}^{\circlearrowleft}$ will reach a stable state, given that $\mathsf{Net(C)}$ waits for network $\mathsf{Net(ALP)}^{\circlearrowleft}$ to stabilize.
(2) Follows from (1) and corollary 1. □

4.3. The Control L

The different components of the network need to know when the current output of the counter C represent a valid explanation $\mathcal{E}$ for $\mathcal{O}$. Either to produce the next candidate, or to analyze its logical consequences. The control logic L consists of:

$$\mathsf{L} = \left\{ \begin{array}{lll} (1) & good & \leftarrow obs, \neg ic, \neg non\_min \\ (2) & sol & \leftarrow sync, good \end{array} \right\}$$

In clause (1) L verifies if the integrity constraints are satisfied $\neg ic$ and whether observation $\mathcal{O}$ follows from the program along with the current candidate $\mathcal{C}$. If

both conditions hold, then *good* will be true in $T_{\mathcal{P}}^{\mathsf{lfp}}$. Meaning that the current candidate $\mathcal{C}$ is indeed an explanation for $\mathcal{O}$. The additional condition *non_min* is used to discard any non minimal explanations.

Throughout this thesis, we use the properties of the fixpoints of Φ and $T_{\mathcal{P}}$. Consequently, when analyzing the the output of $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$ or $\mathsf{Net}(C)$, we need to be sure that they have reached the corresponding fixpoint. Unit *sync* (detailed in the next section) will only be active when all components of the network have had enough time to reach a fixpoint. Accordingly, clause (2) will verify if the current candidate $\mathcal{C}$ is an explanation for $\mathcal{O}$ only when signal *sync* is active. Under these considerations, we have the following properties:

Lemma 8:
Let $\mathsf{Net}(L)$ *be the network obtained by applying the* CORE *method to* L*. Given network* $\mathsf{Net}(C) \cup \mathsf{Net}(ALP)^{\circlearrowleft}$*, we can obtain network* $\mathsf{Net}(C) \cup \mathsf{Net}(ALP)^{\circlearrowleft} \cup \mathsf{Net}(L)$ *by connecting the units in the output layer of* $\mathsf{Net}(ALP)^{\circlearrowleft}$ *to the corresponding units in* $\mathsf{Net}(L)$*. Whenever unit sol is active in* $\mathsf{Net}(C) \cup \mathsf{Net}(ALP)^{\circlearrowleft} \cup \mathsf{Net}(L)$ *it holds that:*

1. *The output of* $\mathsf{Net}(C)$ *represents a minimal explanation* $\mathcal{E}$ *for* $\mathcal{O}$*.*
2. *The output of* $\mathsf{Net}(ALP)^{\circlearrowleft}$ *represents* $\Phi_{\mathcal{P}\cup\mathcal{E}}^{\mathsf{lfp}}$*.*

Proof. Assume unit *sync* is active only when $\mathsf{Net}(\mathsf{C}) \cup \mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$ has reached a stable state. From Proposition 13 we have that the output of $\mathsf{Net}(\mathsf{C})$ represents a minimal explanation $\mathcal{E}$, and the output of $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$ represents $\Phi_{\mathcal{P}\cup\mathcal{E}}^{\mathsf{lfp}}$. If $\mathcal{E}$ explains $\mathcal{O}$, $\Phi_{\mathcal{P}\cup\mathcal{E}}^{\mathsf{lfp}}(obs) = \top$ and $\Phi_{\mathcal{P}\cup\mathcal{E}}^{\mathsf{lfp}}(ic) \neq \top$, that is, unit $obs^{\top}$ is active and unit $ic^{\top}$ passive in the output layer of $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$. Additionally from corollary 2 unit *non_min* is passive in $\mathsf{Net}(\mathsf{C})$, consequently unit *good* is active in $\mathsf{Net}(\mathsf{L})$. Both *sync* and *good* are active, so we conclude that unit *sol* is active only when the output of $\mathsf{Net}(\mathsf{C}) \cup \mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$ represents $\Phi_{\mathcal{P}\cup\mathcal{E}}^{\mathsf{lfp}}$ for some $\mathcal{E}$. □

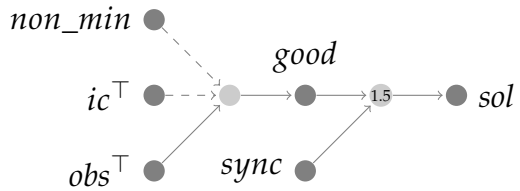


Figure 4.3: *Figure 4.3 shows the core* $\mathsf{Net}(L)$ *obtained from program L. Dark gray units represent atoms. Light gray units represent rules. All units have a threshold value of 0.5, unless stated otherwise, . Dashed lines represent negative connections. Layers are unfolded, and isolated units are not shown.*

4.4. The Clock K

In our approach, we rely on the properties of the stable states of each component of the network. Each stable state needs some time to be reached. Since the output of $\mathsf{Net}(\mathsf{C})$ becomes the input of $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$, and likewise the output of $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$, becomes the input of $\mathsf{Net}(\mathsf{L})$ (and $\mathsf{Net}(\mathsf{S})$ introduced next) we need to take into account the time each component needs to reach a stable state. This ensures that the information transmitted in between components is correct.

Component K will activate the *sync* signal, after enough time has passed, for the whole network to reach a stable state.

$$\mathsf{K} = \bigcup_{i=1}^{D} \{k_i \leftarrow k_{i-1}\} \cup \begin{Bmatrix} k_0 & \leftarrow \neg done, \neg k_D \\ sync & \leftarrow k_0, \neg k_1 \end{Bmatrix}$$

Each cycle of the clock activates the *sync* signal for two consecutive time steps. It is used by L to decide if the current solution is valid (*sol*), and to induce the counter to output the next possible explanation, through signal *sync*. This causes C to output the next combination of abducibles. The clock has $D + 1$ atoms k_i. The value[1] of $D = round(\frac{1}{2}(Path + N + 5))$ is chosen to give the rest of the network enough time to stabilize, where given $\langle \mathcal{P}, \mathcal{A}_{\mathcal{P}}, \mathcal{IC}, \models_{wcs} \rangle$, $Path$ is the longest directed path without repeated atoms in the dependency graph of $\mathcal{P}$, and $N = |\mathcal{A}_{\mathcal{P}}|$. In K each atom k_i depends on its predecessor. The cycle of the clock is $2 \times D$ timesteps given by the combination $k_0, \neg k_1$, which is reached once per cycle. This combination is detected by *sync*, and indicates that network $\mathsf{Net}(\mathsf{C}) \cup \mathsf{Net}(\mathsf{ALP})^{\circlearrowleft} \cup \mathsf{Net}(\mathsf{L})$ has reached a stable state.

The clock has a negative cycle in clause $k_0 \leftarrow \neg done, \neg k_n$, thus runs indefinitely (as intended). Only if *done* is constantly true, k_0 can not change its value anymore, and thus the clock stops. We will see that the unit associated to *done* is activated after all candidate explanations are generated (Proposition 12) and prevents k_0 to change its value. Stopping the clock, also implies that signal *sync* will no longer be activated. And thus the current stable state of $\mathsf{Net}(\mathsf{C}) \cup \mathsf{Net}(\mathsf{ALP})^{\circlearrowleft} \cup \mathsf{Net}(\mathsf{L})$, which depends on *sync*, is the final stable state.

By $\mathsf{Net}(\mathsf{C}) \cup \mathsf{Net}(\mathsf{ALP})^{\circlearrowleft} \cup \mathsf{Net}(\mathsf{L}) \cup \mathsf{Net}(\mathsf{K})$ we denote the network obtained by connecting the units in $\mathsf{Net}(\mathsf{K})$ to the corresponding units in $\mathsf{Net}(\mathsf{C}) \cup \mathsf{Net}(\mathsf{ALP})^{\circlearrowleft} \cup \mathsf{Net}(\mathsf{L})$.

[1] A proof showing that the value of D is correct is given in the annex.

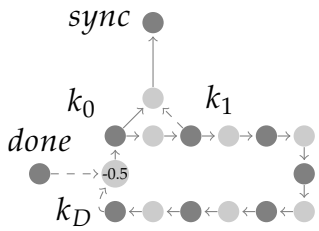


Figure 4.4: *Figure 4.4 shows the core* $\mathsf{Net}(K)$ *obtained from program K. Dark units represent atoms. Light units represent rules. All units have a threshold value of 0.5, unless stated otherwise. Dashed lines represent negative connections. Isolated units are not shown.*

4.5. The Solution S

Given $\mathsf{Net}(\mathsf{C}) \cup \mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$, we now know that the we can obtain the entailed literals for each minimal explanation, from its output layer. But, how can we distinguish those literals entailed by **all and only** minimal explanations?
The solution to the problem of computing skeptical inferences in abduction, is given by the sets $\mathcal{S}^{\top}$, $\mathcal{S}^{\bot}$ and $\mathcal{S}^{\mathsf{U}}$. Taking $\mathcal{S}^{\top}$, it is built by intersecting all corresponding sets $\mathcal{F}_k^{\top} = \{A \mid \mathcal{P} \cup \mathcal{E}_k \models_{wcs} A\}$. Where each $\mathcal{F}_k^{\top}$ is associated to a minimal explanation $\mathcal{E}_k \in \mathcal{M}_{\mathcal{E}}$. Our first task is to obtain each $\mathcal{F}^{\top}$.
We know that when unit *sol* is active in $\mathsf{Net}(\mathsf{C}) \cup \mathsf{Net}(\mathsf{ALP})^{\circlearrowleft} \cup \mathsf{Net}(\mathsf{L})$:

1. The output of $\mathsf{Net}(\mathsf{C})$ is a minimal explanation $\mathcal{E}$ for $\mathcal{O}$.
2. The output of $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$ represents $\Phi^{\mathsf{lfp}}_{\mathcal{P}\cup\mathcal{E}}$.

This means that for each atom A represented by an active unit $A^{\top}$ in the output of $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$, it holds that $\mathcal{P} \cup \mathcal{E} \models_{wcs} A$. To detect this, we introduce a copy A^{+} for each atom $A \in \mathsf{atoms}(\mathcal{P})$, stating that atom $A \in \mathcal{F}_k^{\top}$, for some $\mathcal{E}_k \in \mathcal{M}_{\mathcal{E}}$.

$$\mathsf{S}_1 = \left\{ \begin{array}{ll} (1) & A_i^{+} \leftarrow A_i^{\top}, sol \\ (2) & A_i^{+} \leftarrow A_i^{+} \end{array} \right\}$$

In program S_1, because of (1) A^{+} will be *true* only if $\mathcal{P} \cup \mathcal{E} \models_{wcs} A$. Additionally, unit *sol* is only temporarily active (because it depends on unit *sync* in $\mathsf{Net}(\mathsf{L})$). Clause (2) is necessary to remember A^{+} was active. A similar logic program is provided for the cases where A is mapped to $\bot$ or U in $\Phi^{\mathsf{lfp}}_{\mathcal{P}\cup\mathcal{E}}$:

$$\mathsf{S}_2 = \left\{ \begin{array}{ll} (3) & A_i^{-} \leftarrow A_i^{\bot}, sol \\ (4) & A_i^{-} \leftarrow A_i^{-} \\ (5) & A_i^{\mathsf{U}} \leftarrow \neg A_i^{\top}, \neg A_i^{\bot}, sol \\ (6) & A_i^{\mathsf{U}} \leftarrow A_i^{\mathsf{U}} \end{array} \right\}$$

So far the program $\mathsf{S}_1 \cup \mathsf{S}_2$ allows us to obtain all sets $\mathcal{F}^\top$, besides $\mathcal{F}^\bot$ and $\mathcal{F}^\mathsf{U}$. How can we obtain $\mathcal{S}^\top$ from these sets? The idea is that given any two sets $\mathcal{F}^\top, \mathcal{F}^{\top\prime}$ and an atom A, if $A \in \mathcal{F}^\top$ but $A \notin \mathcal{F}^{\top\prime}$, then $A \notin \mathcal{S}^\top$. That is, A has changed its value depending on the explanation. We refer to such A as *bad* atoms, and represent them as A^*. The following clauses detect the bad atoms:

$$\mathsf{S}_3 = \left\{ \begin{array}{ll} (7) & A_i^* \leftarrow A_i^+, A_i^- \\ (8) & A_i^* \leftarrow A_i^+, A_i^\mathsf{U} \\ (9) & A_i^* \leftarrow A_i^-, A_i^\mathsf{U} \end{array} \right\}$$

Finally we have:

$$\mathsf{S} = \mathsf{S}_1 \cup \mathsf{S}_2 \cup \mathsf{S}_3$$

By $\mathsf{Net}(\mathsf{C}) \cup \mathsf{Net}(\mathsf{ALP})^{\circlearrowleft} \cup \mathsf{Net}(\mathsf{L}) \cup \mathsf{Net}(\mathsf{K}) \cup \mathsf{Net}(\mathsf{S})$ we denote the network obtained by connecting the units in $\mathsf{Net}(\mathsf{S})$ to the corresponding units in $\mathsf{Net}(\mathsf{C}) \cup \mathsf{Net}(\mathsf{ALP})^{\circlearrowleft} \cup \mathsf{Net}(\mathsf{L}) \cup \mathsf{Net}(\mathsf{K})$.

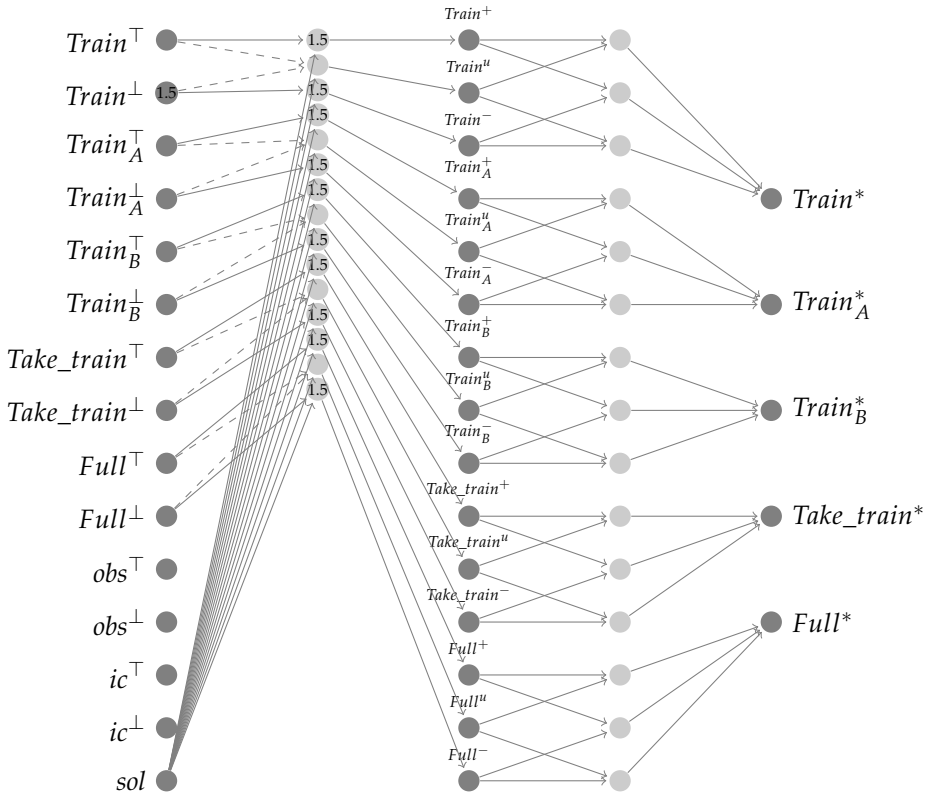


Figure 4.5: *Figure 4.5 shows the core* $\mathsf{Net}(S)$ *obtained from program* S*. All units have a threshold of 0.5, unless stated otherwise. Dark units represent atoms and light units represent rules and are active if all of its inputs are 1. Dashed lines represent negative connections. Layers are unfolded.*

4.6. The Resulting Network

The sets $\mathcal{S}^{\top}$, $\mathcal{S}^{\bot}$ and $\mathcal{S}^{\mathsf{U}}$ can now be easily obtained from $\mathsf{Net(C)} \cup \mathsf{Net(ALP)}^{\circlearrowleft} \cup \mathsf{Net(L)} \cup \mathsf{Net(K)} \cup \mathsf{Net(S)}$ after it has reached a stable state. Each set is composed by the corresponding atoms that are not bad:

Theorem 1:
Given an abductive framework $AF = \langle \mathcal{P}, \mathcal{A}_{\mathcal{P}}, \mathcal{IC}, \models_{wcs} \rangle$ and an observation $\mathcal{O}$. The associated network $\mathsf{Net}(C) \cup \mathsf{Net}(ALP)^{\circlearrowleft} \cup \mathsf{Net}(L) \cup \mathsf{Net}(K) \cup \mathsf{Net}(S)$ reaches a stable state from which we can obtain:

- *The set $\mathcal{S}^{\top} = \{A | A \in \mathsf{atoms}(\mathcal{P})$, unit A^{+} is active and unit A^{*} passive $\}$*
- *The set $\mathcal{S}^{\bot} = \{A | A \in \mathsf{atoms}(\mathcal{P})$, unit A^{-} is active and unit A^{*} passive $\}$*
- *The set $\mathcal{S}^{\mathsf{U}} = \{A | A \in \mathsf{atoms}(\mathcal{P})$, unit A^{U} is active and unit A^{*} passive$\}$*

Figure 4.6 shows a full diagram of the network, for the abductive framework $\mathsf{AF} = \langle \mathcal{P}_{Train}, \mathcal{A}_{\mathcal{P}}, \mathcal{IC}, \models_{wcs} \rangle$. For the sub-network corresponding to $\mathsf{Net(C)}$ in the diagram, only the units that interact with the rest of the network are specified.

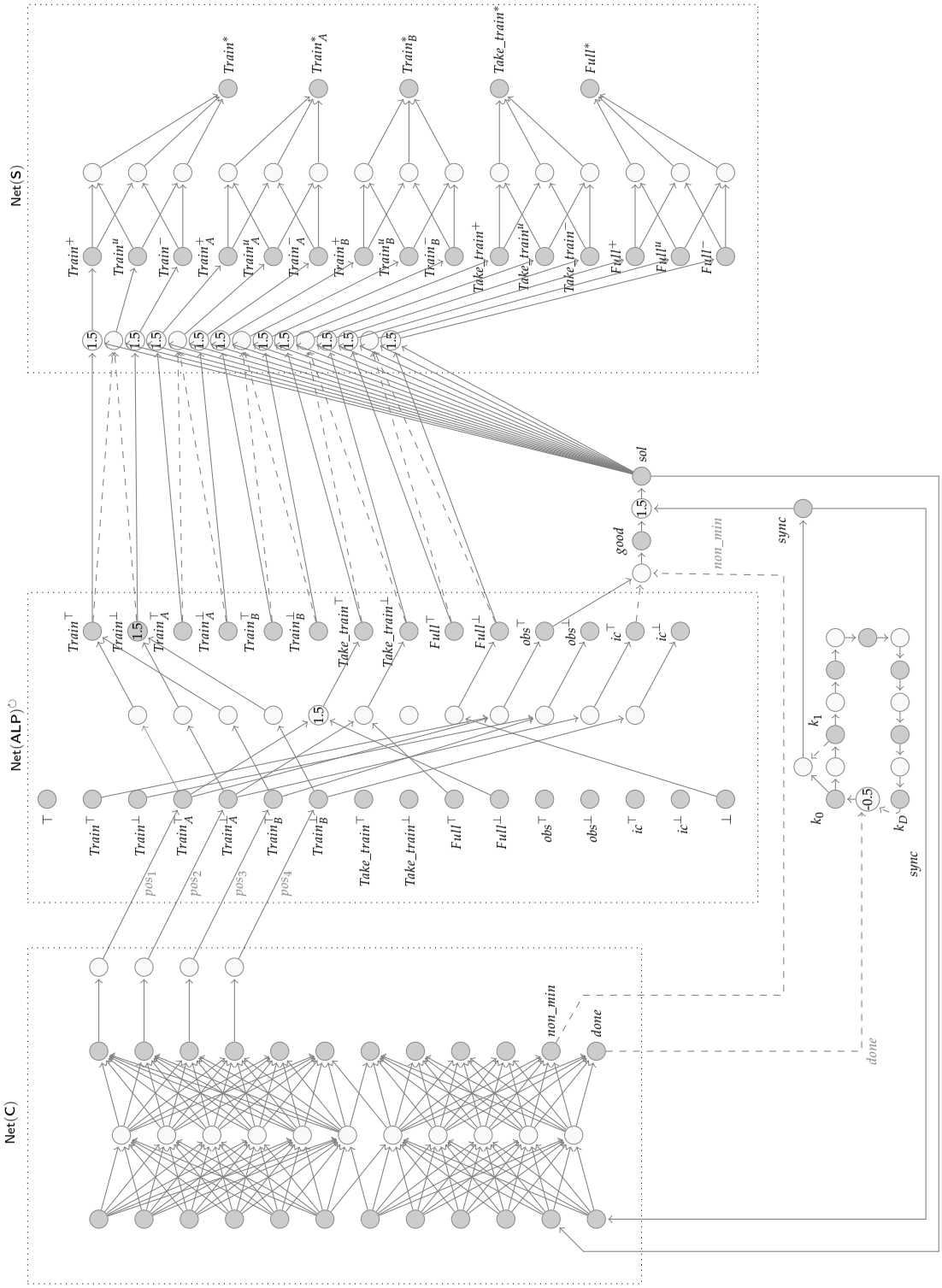


Figure 4.6: *The network* $\mathsf{Net}(C) \cup \mathsf{Net}(ALP)^{\circlearrowleft} \cup \mathsf{Net}(L) \cup \mathsf{Net}(K) \cup \mathsf{Net}(S)$. *Dark units represent atoms. Light units represent rules. All units have a threshold value of 0.5, unless stated otherwise. Dashed lines represent negative connections. Recurrent connections are not shown. Networks* $\mathsf{Net}(C)$, $\mathsf{Net}(ALP)^{\circlearrowleft}$ *and* $\mathsf{Net}(S)$ *are surrounded by dotted rectangles. Some layers are unfolded.*

Chapter 5

Applications in Human Reasoning

In this chapter we show how our approach can be used for modeling human reasoning tasks under a fully connectionist realization.

5.1. The Suppression Task

In [5] Byrne presented the selection task, where its shown that students with no prvious experience in formal logic did suppress previously drawn conclusions when additional information became available. As an example consider the following premises:

- A: If she has an essay to finish then she will study late in the library.
- B: If she has a textbook to read then she will study late in the library.

If given premise A we observe that *she is studying in the library*, most humans conclude *she has an essay to finish*. But interestingly, if we additionally provide premise B, most of the people suppress the previously drowned conclusion [5] and no longer infer that *she will study late in the library*.

Taking [25] as a reference, the first scenario can be encoded as:

$$\mathcal{P}_A = \left\{ \begin{array}{lll} (1) & l & \leftarrow e, \neg ab_1 \\ (2) & ab_1 & \leftarrow \bot \end{array} \right\}$$

Where l indicates she will study late in the library, and e indicates she has an essay to write. Predicate ab_1 states we know something abnormal relevant to clause (1). In our example nothing abnormal is known and consequently $ab_1 \leftarrow \bot$ is added to $\mathcal{P}_A$. Given the observation $\mathcal{O} = \{l\}$, we find that the unique minimal explanation is $\mathcal{E} = \{e \leftarrow \top\}$. The corresponding sets of skeptical inferences are:

- $\mathcal{S}^\top = \{e, l\}$

- $\mathcal{S}^{\bot} = \{ab_1\}$
- $\mathcal{S}^{\mathsf{U}} = \emptyset$

And thus, we can skeptically conclude e and l. Now, consider a modification of $\mathcal{P}_A$ by adding to it premise B:

$$\mathcal{P}_{AB} = \left\{ \begin{array}{lll} (1) & l & \leftarrow e, \neg ab_1 \\ (2) & l & \leftarrow t, \neg ab_2 \\ (3) & ab_1 & \leftarrow \bot \\ (4) & ab_2 & \leftarrow \bot \end{array} \right\}$$

In program $\mathcal{P}_{AB}$, predicate t states *she has a textbook to read* and ab_2 is the auxiliary predicate for expressing abnormality for clause 2. Given again observation $\mathcal{O} = \{l\}$, we now find two minimal explanations for $\mathcal{O}$: $\mathcal{E}_1 = \{e \leftarrow \top\}$ and $\mathcal{E}_2 = \{t \leftarrow \top\}$. The corresponding sets of skeptical inferences now are:

- $\mathcal{S}^{\top} = \emptyset$
- $\mathcal{S}^{\bot} = \{l\}$
- $\mathcal{S}^{\mathsf{U}} = \emptyset$

As a result, if we reason skeptically under weak completion semantics, we can no longer conclude that *she has an essay to write*, which was a valid conclusion on the absence of premise B. This corresponds to the results in [5] and shows that weak completion semantics is adequate for modeling this task.

5.2. A Connectionist Realization

To obtain a connectionist network for the suppression task, we first construct the abductive framework $AF_{AB} = \langle \mathcal{P}_{AB}, \mathcal{A}_{\mathcal{P}}, \mathcal{IC}, \models_{wcs} \rangle$ where:

$$\mathcal{P}_{AB} = \left\{ \begin{array}{ll} l & \leftarrow e, \neg ab_1 \\ l & \leftarrow t, \neg ab_2 \\ ab_1 & \leftarrow \bot \\ ab_2 & \leftarrow \bot \end{array} \right\}$$
$$\mathcal{A}_{\mathcal{P}} = \{e \leftarrow \top, e \leftarrow \bot, l \leftarrow \top, l \leftarrow \bot\}$$
$$\mathcal{IC} = \emptyset$$
$$\mathcal{O} = \{l\}$$

From the abductive framework AF_{AB} we obtain the corresponding abductive logic program:

$$\mathsf{ALP} = \left\{ \begin{array}{ll} l & \leftarrow e, \neg ab_1 \\ l & \leftarrow t, \neg ab_2 \\ ab_1 & \leftarrow \bot \\ ab_2 & \leftarrow \bot \\ obs & \leftarrow l \end{array} \right\}$$

Applying the extended CORE method to ALP we obtain the associated three-layered recurrent neural network $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$:

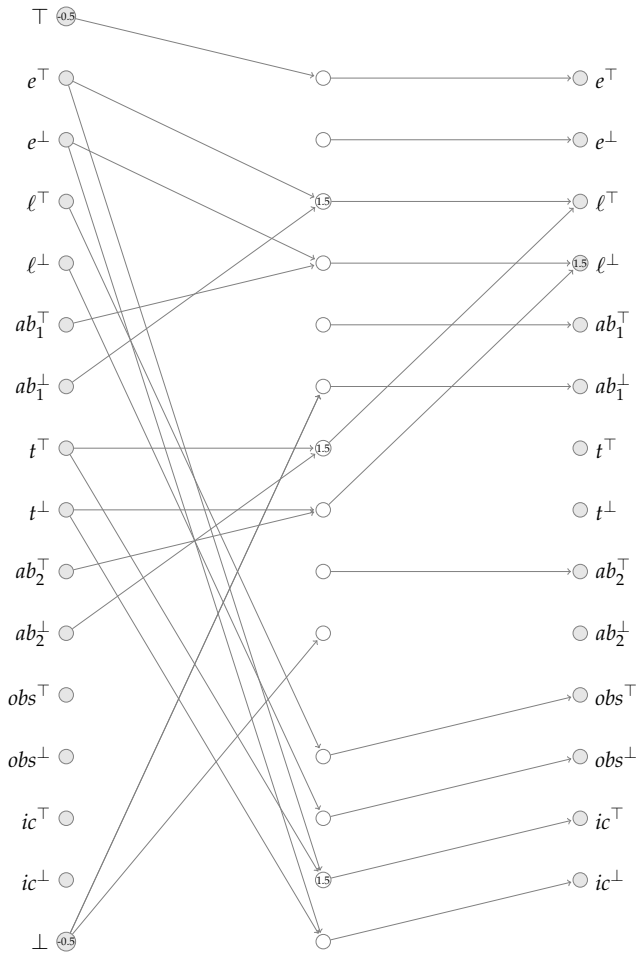


Figure 5.1: *Diagram of the network* $\mathsf{Net}(ALP)^{\circlearrowleft}$ *for program ALP. Units in the input and output layer represent atoms, units in the hidden layer represent rules. Recurrent connections from the input to the output layer are not shown. All units have a threshold* $\Omega = 0.5$*, unless stated otherwise.*

Finally, we surround $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$ by all the components necessary to compute skeptical abduction specified in Chapter 4 and make the corresponding connections between units. The stable state of the resulting network is shown in figure 5.2, from where we can obtain what skeptically follows from $\mathcal{P}_{AB}$ and the observation $\mathcal{O} = \{l\}$. If we construct the sets $\mathcal{S}^{\top}, \mathcal{S}^{\perp}$ and $\mathcal{S}^{\mathsf{U}}$ from the resulting network,

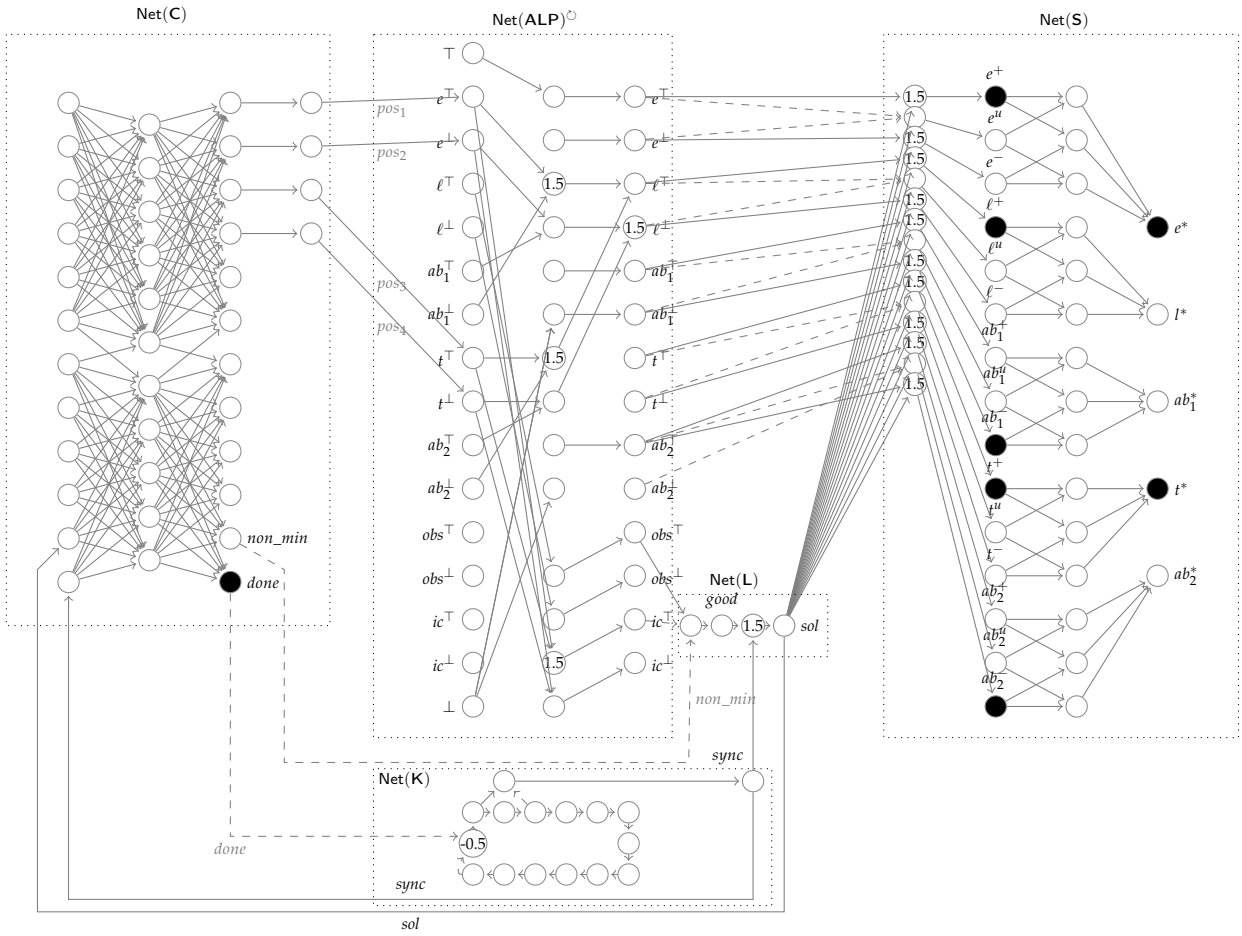


Figure 5.2: *Stable state of the complete network for program $\mathcal{P}_{AB}$ and observation $\mathcal{O} = \{l\}$. Black units are active and light units are passive. All units have a threshold $\Omega = 0.5$, unless stated otherwise. Recurrent connections are not shown.*

by inspecting the units in the sub-network $\mathsf{Net}(\mathsf{S})$, we obtain :

- $\mathcal{S}^{\top} = \{l\}$
- $\mathcal{S}^{\perp} = \{ab_1, ab_2\}$
- $\mathcal{S}^{\mathsf{U}} = \emptyset$

From which we can not conclude *e* (*she has an essay to write*) which corresponds to [5] and shows the adequacy of the approach. The other scenarios of the suppression task can be modeled similarly. The interested reader can refer to [5, 16, 25, 26] for further details and examples.

Chapter 6

Conclusions and Further Work

We have provided a connectionist realization, solving an open problem in [17, 25, 30]. Showing that all scenarios presented in Byrne [5] can be adequately modeled using weak completion semantics and the CORE method.

While studying the problem and its characteristics, a revised definition for skeptical abduction was given. Where we show that restricting to minimal explanations is not necessary, although they suffice to determine skeptical inferences of an abductive framework.

The consequences of considering only minimal explanations are explored, and we find that more inferences can be made than considering all explanations. We extended the definition of skeptical abduction to refer to the newly obtained inferences. Moreover we show that this allows us to detect irrelevant information in the original program, with respect to the observation.

In the process of looking for explanations for a given observation, one of the main difficulties of skeptical abduction is the exponential number of possible explanations. We have seen that instead of considering all of them, we can focus only on those minimal. But there might still be an exponential number of minimal explanations. In our approach, some of the abducibles might be found as irrelevant. This provides a way to further impose restrictions in the explanations, and a way to revise the knowledge provided in the original program. Both these tasks remain as further work.

The translation of the different components into neural networks, provided a seamless way to integrate two and three-valued components, preserving their properties. To the best of our knowledge, this has not been done before. This integration shows some of the possibilities of providing a neuro-symbolic approach. While two and three-valued logic have different semantics, neural networks are composed of very similar units. Once a translation from logic to neural networks is achieved, the components can still interact with each other without affecting

their semantics. This provides an excellent example on how hybrid approaches can be more flexible than only symbolic approaches. Moreover, if we abstract from the logic programs and focus on the resulting neural network, it could be trained and some of the connections and units might be dropped (isolated units, connections with eight 0, etc). It would be interesting to see how much of the original architecture is preserved, if we allow the network to be optimized by machine learning techniques.

As we have seen, detecting minimality is not a trivial task. In the counter we provided a specification that produces all explanations, in a specific order, and detect the minimal ones. Although efficiency is not our main goal, both components can be further optimized. In the current setting, the counter tests the current explanation and then detects if it is non-minimal. We could discard the non-minimal candidate before being tested, or even not generate them at all. Additionally, the counter has been implemented and tested, but prooving completness of Algorithm 1 remains an open task.

We must also mention how cycles in the original program must be handled. As discussed in [14, 15, 30] cycles can introduce a memory in the recurrent network $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$ corresponding to the abductive logic program, since a unit will remain active once activated. For each new candidate generated, $\mathsf{Net}(\mathsf{ALP})^{\circlearrowleft}$ must be relaxed. This can be done by adding a $\neg sync$ to each rule in ALP, thus relaxing the network on each cycle of the clock.

Finally, we do not claim that the presented approach precisely models how we humans reason. Although it can be used to adequately model the conclusions drawn by humans not only on Byrne's tasks, but is additionally applicable to syllogisms [29], spatial reasoning [28] and conditionals [27]. We consider that weak completion semantics and neuro-symbolic integration, provide a solid framework, on which we can build theories on how we reason. Validating, improving or discarding them remains a further work.

Chapter 7

Appendix

7.1. Semantic Operators Software

The software developed for computing $T_{\mathcal{P}}$ and Φ operators, used to test the examples in this Thesis was developped at the Technische Universität Dresden in the period 2015/2016. In the following the specifications and usage are provided.

SvL Software

Quick guide

The SvL software allows to compute the two-valued Tp operator or the three-valued SvL operator, given a logic program. Additionally to computing the above semantic operators fix-point, the software is capable of performing abduction, although this feature is still under development.

The software is developed in Java and delivered as a .jar file, along with some prolog routines in separate files. This makes it possible to use it under any platform: Windows, Linux or Mac.

Requirements:

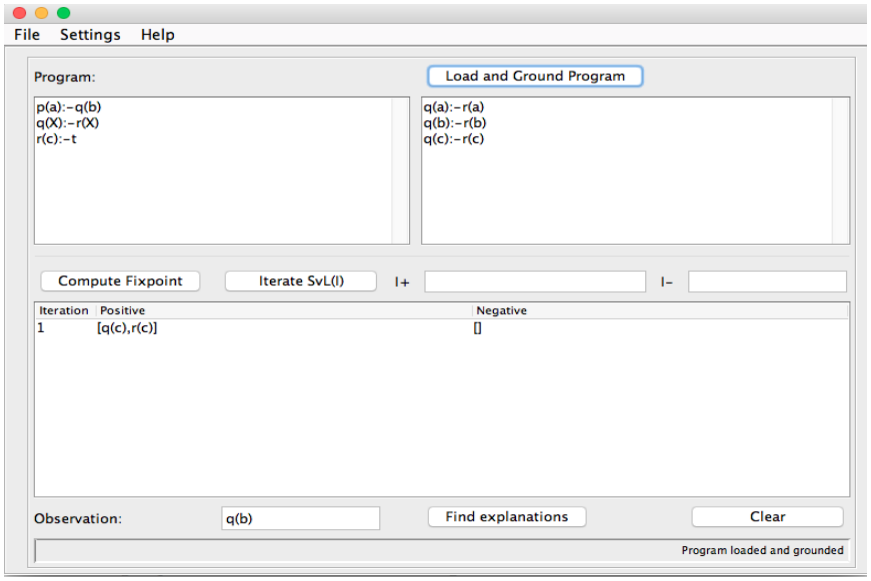
1. Java Runtime 6+ installed
2. Prolog installed

How to run it

1. Make sure all files provided are located in the same directory (SvL.jar and the *.pl files)
2. Execute SvL.jar from the command line: `java -jar SvL.jar`

   (In some cases double clicking SvL.jar will suffice)

This will prompt the application interface. As soon as the application is started, it will test if it can connect to Prolog. If it fails, settings must be changed in **Menu>Settings>Prolog Version**.

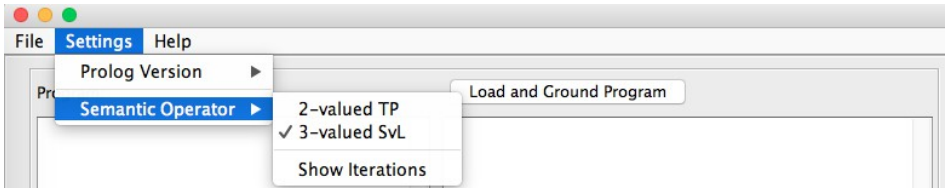


The following is a description of the main functions of the program:

1. **Load and Ground Program:** The input program is loaded into the software's engine and the clauses with variables are grounded. Grounded clause are shown in the top-right area.
2. **Compute Fixpoint:** Depending on the selected operator (Menu>Settings>Semantic Operator) the fixpoint is computed and presented as a two or three-valued interpretation, in the lower area.
3. **Iterate SvL(I):** The program allows to provide an input interpretation for iterating the operator. Pressing the **Iterate SvL(I)** will compute one iteration of the selected operator, and the ouput will be provided in both: the lower area, and as the new input for the next itration.

4. **Find Explanations:** Allows to compute abductive explanations for the observation given in the **Observation** text field. An observation consists of a single literal.
5. **Clear:** Erases the computed iterations, and the input for the initial interpretation.

Menu



The top menu has three sections:

1. **File:** Allows to save the current program, and to load previously saved ones. A program file can contain comments. These are lines starting with the character "**%**".
2. **Settings:** Allows to change the Prolog version to be used, as well as choosing the semantic operator to be computed. It also allows to show all iterations until getting to the fixpoint.
3. **Help:** Provides a link to this guide, and the credits of the software.

Syntax

The **syntax** of the logic program has the following restrictions: s an example, the following is a logic program with 3 rules:

1. Each rule is of the form:

   `head:-body1,body2,body3`

   where `head` is an atom and `body1`, `body2`, `body3` are literals.
2. Each Atom is of the form: `predicate(arg1,...,arg3).`
3. In principle each predicate can be of any arity, although an excessive number of arguments may cause memory fails.
4. Only one rule per line.
5. Literals in the body are separated by a coma ",".
6. The program must not contain white spaces.
7. Variables always begin with a capital letter.
8. The constants `t` and `f` are reserved to denote true and false, respectively. And should not be used as predicate symbols nor arguments. They are used to represent negative and positive facts.
9. Negation is represented as a predicate of the form: `n(Literal).`
10. Avoid using numbers as arguments.

Example

As an example, the following is a logic program with 3 rules:

- A positive fact : `p(a,b):-t`
- A negative fact: `q(a,a):-f`
- A non-ground rule: `r(X,Y):-n(q(X,Y)),p(X,X)`

Where the non-ground clause is replaced by four ground clauses in the ground program, and whose least SvL fix-point is `I=<{p(a,b)},{q(a,a)}>` .

Abduction

Abduction (finding explanations given an observation) is still on development and can take excessive time to compute if the number of abducibles is big, leading to a memory overflow error.

To obtain explanations for a given observation, with respect to a logic program:

1. First, obtain the SvL fix-point as described above.
2. Provide the observation on the corresponding text box, in the same format as described above. (As a ground literal).
3. Use the (Find explanations) button to look for explanations for the observation. All candidate explanations are shown, and when a valid one is found, it is highlighted and the corresponding least fix-point is given as evidence.

Getting back to the example, if we would have that our observation is `r(b,b)`, and we want to have explanations for it, the result would be:

Meaning there exists 27 explanations for `r(b,b)`, i.e. all of those that contain `p(b,b)` and `n(q(b,b))`.

Installing Prolog

The following links provide information on how to install prolog in the different platforms:

1. Linux: http://www.swi-prolog.org/build/Debian.html
2. MAC: http://www.swi-prolog.org/build/macos.html
3. Wndows: http://www.swi-prolog.org/build/windows.html

Luis Palacios Medinacelli

palacios.medinacelli@gmail.com

Bibliography

[1] J. Łukasiewicz. O logice trójwartościowej. *Ruch Filozoficzny*, 5:169–171, 1920. English: On Three-Valued Logic. In: *Jan Łukasiewicz Selected Works*. (L. Borkowski, ed.), North Holland, 87-88, 1990.

[2] K. Apt and M. van Emden. Negation as failure. In Gallaire, H., and Minker, J., eds., Logic and Data Bases, volume 1. New York, NY: Plenum Press. 293–322.

[3] K. Apt and M. van Emden. Contributions to the theory of logic programming. *J. ACM*, 29:841–862, 1982.

[4] J. W. Lloyd. *Foundations of Logic Programming*. Springer, 1984.

[5] Byrne, R. *Suppressing valid inferences with conditionals*. Cognition, 31, 61-83, 1989.

[6] A.C. Kakas and P. Mancarella. Generalized Stable Models: a Semantics for Abduction. In *Proc. 9th European Conference on Artificial Intelligence*, 385–391. Pitman, 1990.

[7] A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive Logic Programming. *J. Logic and Computation*, 2(6):719–770, 1993.

[8] M. Fitting. Metric methods – three examples and a theorem. *J. of Logic Programming*, 21(3):113–127, 1994.

[9] S. Hölldobler and Y. Kalinke. Towards a new massively parallel computational model for logic programming. In *Proc. ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, 68–77. ECCAI, 1994.

[10] Y. Kalinke. Ein massiv paralleles Berechnungsmodell für normale logische Programme. Master's thesis, TU Dresden, Fakultät Informatik, 1994. (in German).

[11] S. Hölldobler, Y. Kalinke, and H.-P. Störr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11:45–59, 1999.

[12] A. Seda and M. Lane. Some aspects of the integration of connectionist and logic-based systems. In *Proc. Third International Conference on Information*, 297–300, International Information Institute, Tokyo, 2004.

[13] S. Bader and S. Hölldobler. The Core method: Connectionist model generation. In S. Kollias, A. Stafylopatis, W. Duch, and E. Ojaet, eds., *Proc. 16th International Conference on Artificial Neural Networks (ICANN)*, vol. 4132 of *LNCS*, 1–13. Springer, 2006.

[14] A. d'Avila Garcez, D. Gabbay, O. Ray, and J. Woods. Abductive reasoning in neural-symbolic learning systems. *TOPOI*, 26:37–49, 2007.

[15] O. Ray and A. d'Avila Garcez. Towards the integration of abduction and induction in artificial neural networks. In *Proc. ECAI'06 Workshop on Neural-Symbolic Learning and Reasoning*, 41–46, 2006.

[16] K. Stenning and M. van Lambalgen. *Human Reasoning and Cognitive Science*. MIT Press, 2008.

[17] Kencana Ramli Logic Programs and Three-Valued Consequence Operators. Master's Thesis, Technische Universität Dresden, 2009.

[18] S. Hölldobler and Kencana Ramli Logics and Networks for Human Reasoning. In: Artificial Neural Networks, Alippi et.al. eds., LNCS 5769, 85-94: 2009.

[19] S. Hölldober. *Logik und Logikprogrammierung: Grundlagen*. Synchron Publishers GmbH, Heidelberg, 2009.

[20] S. Hölldobler and Kencana Ramli. Contraction properties of a semantic operator for human reasoning. In L. Li and K. K. Yen, eds., *Proc. Fifth International Conference on Information*, 228–231. International Information Institute, Tokyo, 2009.

[21] S. Hölldobler and Kencana Ramli. Logic programs under three-valued Łukasiewicz semantics. In P. M. Hill and D. S. Warren, eds., *Logic Programming*, vol. 5649 of *LNCS*, 464–478. Springer, 2009.

[22] O. Ray and A. d'Avila Garcez. A Neural Network Approach for First-Order Abductive Inference. In *Proc. IJCAI09 Workshop on Neural-Symbolic Learning and Reasoning*, 2009.

[23] Bernhard Sendhoff, Edgar Körner, Olaf Sporns,Helge Ritter,Kenji Doya Lecture Notes in Computer Science: Creating Brain-Like Intelligence, Springer, 2009.

[24] S. Hölldobler, T. Philipp, and C. Wernhard. An abductive model for human reasoning. In *Proc. 10th International Symposium on Logical Formalizations of Commonsense Reasoning*, 2011.

[25] E.-A. Dietz, S. Hölldobler, and M. Ragni. A computational logic approach to the suppression task. In N. Miyake, D. Peebles, and R. P. Cooper, eds., *Proc. 34th Annual Conference of the Cognitive Science Society*, 1500–1505. Cognitive Science Society, 2012.

[26] E.-A. Dietz, S. Hölldobler, and M. Ragni. A computational logic approach to the abstract and the social case of the selection task. In *Proc. 11th International Symposium on Logical Formalizations of Commonsense Reasoning*, 2013.

[27] E.-A. Dietz, S. Hölldobler, and L. M. Pereira. On indicative conditionals. In S. Hölldobler and Y. Liang, eds., *Proc. First International Workshop on Semantic Technologies*, vol. 1339 of *CEUR Workshop Proc.*, 19–30, 2015. `http://ceur-ws.org/Vol-1339/`.

[28] E.-A. Dietz, S. Hölldobler, and R. Höps. A Computational Logic Approach to Human Spatial Reasoning. 2015.

[29] E.-A. Dietz. A Computational Logic Approach to Syllogisms in Human Reasoning. In Ulrich Furbach, Claudia Schon, eds., Proceedings of the Workshop on Bridging the Gap between Human and Automated Reasoning. A workshop of the 25th International Conference on Automated Deduction (CADE-25), Berlin, Germany, August 1, 2015., 17-31, 2015.

[30] L. Palacios Medinacelli, S. Hölldobler and E.-A. Dietz A connectionist network for skeptical abduction. Proceedings of the 10th International Workshop on Neural-Symbolic Learning and Reasoning NeSy'15, IJCAI 2015.