

A connectionist approach to abductive problems: employing a learning algorithm

Andrzej Gajda, Adam Kups, Mariusz Urbański
Adam Mickiewicz University
in Poznań

ul. Szamarzewskiego 89AB, 60-568 Poznań, Poland
Email: {Andrzej.Gajda, adamku, Mariusz.Urbanski}@amu.edu.pl

Abstract—This paper presents preliminary results of an application of artificial neural networks and Backpropagation learning algorithm to solve logical abductive problems. To represent logic programs in the form of artificial neural networks CIL²P approach proposed by Garcez et al. [3] is employed. Our abductive procedure makes use of translation of a logic program representing a knowledge base into a neural network, training of the neural network with an example representing an abductive goal and translation of the trained network back to the form of a logic program. An abductive hypothesis is represented as the symmetric difference between the initial logic program and the one obtained after training of the network. The first part of the paper introduces formal description of the tools used to model the abductive process, while the second part illustrates our contribution with results of a few computational experiments and discusses the ways of possible improvements of the proposed procedure.

I. INTRODUCTION

ABDUCTION is a kind of reasoning which allows to fill the gap between a knowledge base Γ and a puzzling phenomenon ϕ , unattainable from Γ (cf. [6, 14]). We follow the algorithmic point of view, according to which an abductive hypothesis H “is legitimately dischargeable to the extent to which it makes it possible to prove (or compute) from a database a formula not provable (or computable) from it as it is currently structured” [2, p. 88]. A short characteristic of abduction interpreted along these lines can be found in [9]. Our goal in this paper is to use definite logic programs to formalise a class of abductive problems and to apply neural networks as a tool for abductive hypotheses generation.

Definite logic programs are characterised in the first order language [11]. However, in our approach we use only grounded definite logic programs [3]. Therefore, the formalisation of the abductive problems is restricted to the classical propositional logic.

The connectionist approach that we adopt makes use of positive partially recurrent one hidden-layer networks. This is sufficient to deal with definite logic programs as indicated in [3].

An abductive problem is stated for definite logic programs and the abductive hypotheses are generated by means of neural

networks. Therefore, we use Connectionist Inductive Learning and Logic Programming System’s ($C\text{-}IL^2P$) translation algorithm proposed by Garcez et al. [3] to translate logic programs into neural networks. The learning process of the neural network is aimed at changing it in such a way that the abductive goal is attained. Subsequently, the neural network is translated back into a logic program. Differences between the initial logic program and the one obtained from the trained neural network may be interpreted as abductive hypotheses generated in the learning process.

There are a few ways to model abduction using $C\text{-}IL^2P$. Two of them were described in [4] and they employ either a connectionist modal logic or an neural-symbolic system for abductive logic program. The approach that we want to present takes advantage of network learning process (using Backpropagation algorithm) and program-to-network and network-to-program translation algorithms. The main advantage of this approach in comparison with the two previous ones lies in its flexibility: abductive hypotheses do not have to be reduced to the form of propositional formulas (in particular conjunctions of atoms). At the present stage of our research we consider only abductive goals represented by atoms, but our approach allows for any form of an abductive goal (atoms or other types of formulas). Abduction here can be seen as a process of training of a network previously obtained from a logical program. The program represents a knowledge base and an abductive goal is represented by a training example. The reverse translation of the trained network allows to obtain abductive hypotheses. Using this approach abductive hypotheses can be represented by new logical formulas that are extending initial knowledge base. However, it is also possible that modifications or even removal of clauses from a knowledge base will represent abductive hypotheses (some similarity can be found in *contraction* and *revision* operations in belief revision theory, e.g. [5]).

In order to better comprehend the stated problem let us consider the following abductive problem. Suppose that we established a general medical facts¹: “If people suffer from a fatigue and a fever, then they have a flu”. “If people suffer from a fever, then they are fatigued”. “If people have a flu, then they

Research reported in this paper were supported by the National Science Centre, Poland (DEC-2013/10/E/HS1/00172).

¹Please, note that that the presented problem is only a simple toy-example, advanced medical decision support systems are for example presented in [13, 15]

are fatigued". Now, we would like to know what should we check if we suspect that someone has a flu. Formally, this problem can be described in the following way: let the knowledge base be a set of formulas $X = \{B \wedge C \rightarrow A, C \rightarrow B, A \rightarrow B\}$ and the abductive goal be a formula A , where A means a person has a flu, B means a person is fatigued and C means that a person has a fever. We shall go back to this example in every section.

II. LOGIC PROGRAMS

Definite logic programs are usually formalised in the first order language. However, for the translation of definite logic program to neural network it is required for the program to be grounded. Therefore, grounded definite logic program consists only of predicates with constants as arguments. This is the reason for using a simpler language than the first order language. We characterize definite logic programs following [3] and similarly to [11].

We use language \mathcal{L} , which consists of the following elements:

- $\{a_1, a_2, \dots\}$ — an infinite, countable set of propositional letters,
- \leftarrow — a primitive connective,
- $,$ — a comma.

Atomic formulas are denoted by propositional letters a_1, a_2, \dots .

Definition 2.1: Let a_i for $0 \leq i \leq n$ be an atomic formula. *Horn clauses* are formulas of the form:

$$h_i = a_{i_0} \leftarrow a_{i_1}, \dots, a_{i_n}.$$

The atomic formula a_{i_0} from the definition of the Horn clause is called the *head* of the Horn clause h_i and will be denoted as $head(h_i)$. Similarly, the set of atomic formulas $\{a_{i_1}, \dots, a_{i_n}\}$ forms the *body* of the Horn clause h_i and will be denoted as $body(h_i)$. It is possible that the body of a Horn clause contains no atoms. Such Horn clauses will be called *facts*.

Atomic formulas and Horn clauses are the only well-formed formulas we use.

Definition 2.2: The *set of well-formed formulas* is defined in the following way:

$$Form = \{f \mid f = a_i \text{ or } f = h_j\}.$$

Definition 2.3: Let h_i for $1 \leq i \leq n$ be a Horn clause. A *definite logic program* is denoted by \mathcal{P} and defined as follows:

$$\mathcal{P} = \{h_1, \dots, h_n\}.$$

Definition 2.4: Let \mathcal{P} be a definite logic program. We define the set of all atoms that occur in \mathcal{P} in the following way:

$$B_{\mathcal{P}} = \{a_i \mid \text{for every } h_k \in \mathcal{P} \text{ and for every } a_j \in body(h_k): a_i = head(h_k) \text{ or } a_i = a_j\}.$$

$B_{\mathcal{P}}$ is called Herbrand base of program \mathcal{P} .

We are now going to define *Least Herbrand model* of a definite logic program \mathcal{P} which in turn will be used in the definition of logical consequence of a definite logic program \mathcal{P} .

Definition 2.5: A mapping $v : Form \mapsto \{true, false\}$ is a *valuation* defined as follows:

- for every atomic formula a_i : either $v(a_i) = true$ or $v(a_i) = false$,
- for every Horn clause h_i : $v(h_i) = true$ iff $v(head(h_i)) = true$ or for at least one $a_j \in body(h_i)$: $v(a_j) = false$.

Definition 2.6: Let \mathcal{P} be a definite logic program, $B_{\mathcal{P}}$ a Herbrand base of \mathcal{P} and v a valuation. An (Herbrand) *interpretation of a program* \mathcal{P} w.r.t. v is a set $I_{\mathcal{P}}$ of all atoms in $B_{\mathcal{P}}$ that are mapped by v to *true*:

$$I_{\mathcal{P}}(v) = \{a_i \mid a_i \in B_{\mathcal{P}} \text{ and } v(a_i) = true\}.$$

Atoms that do not belong to the interpretation $I_{\mathcal{P}}$ are mapped to *false*.

Definition 2.7: Let \mathcal{P} be definite logic program, h_i be a Horn clause that belongs to \mathcal{P} and $I_{\mathcal{P}}$ a Herbrand interpretation of \mathcal{P} w.r.t. valuation v . *Model of* \mathcal{P} is defined as follows:

$$m_{\mathcal{P}} =_{df} I_{\mathcal{P}}(v) \text{ such that for every } h_i \in \mathcal{P}: v(h_i) = true.$$

It follows from the definition 2.7 that models of definite logic program \mathcal{P} are of the form of sets of atoms. Therefore, we can establish a hierarchy over those models and use it to define the smallest model of \mathcal{P} .

Definition 2.8: Let S be a set. Function $c : S \rightarrow \mathbb{N}$ returns the number of elements in the set S .

Definition 2.9: Let \mathcal{P} be a definite logic program and $m_{\mathcal{P}}$ a model of \mathcal{P} . By $m_{\mathcal{P}}^{min}$ we denote a *minimal model of* \mathcal{P} and define it in the following way:

$$m_{\mathcal{P}}^{min} =_{df} m_{\mathcal{P}} \text{ such that for every } m'_{\mathcal{P}}: c(m_{\mathcal{P}}) \leq c(m'_{\mathcal{P}}).$$

Definition 2.10: Let \mathcal{P} be a definite logic program and $m_{\mathcal{P}}$ a model of \mathcal{P} . By $M_{\mathcal{P}}$ we denote the *least Herbrand model of* \mathcal{P} and defined it in the following way:

$$M_{\mathcal{P}} =_{df} m_{\mathcal{P}} \text{ such that for every } m'_{\mathcal{P}} \neq m_{\mathcal{P}}: c(m_{\mathcal{P}}) < c(m'_{\mathcal{P}}).$$

Now we are going to define Immediate Consequence Operator denoted as $T_{\mathcal{P}}$. It „provides the link between the declarative and procedural semantics of \mathcal{P} ” [11, p. 37]. Related definitions concerning lattices are standard, therefore they are included in the Appendix.

Definition 2.11: Let \mathcal{P} be a definite logic program and $I_{\mathcal{P}}$ an interpretation. The mapping $T_{\mathcal{P}} : 2^{B_{\mathcal{P}}} \mapsto 2^{B_{\mathcal{P}}}$ is defined as follows:

$$T_{\mathcal{P}}(I_{\mathcal{P}}) =_{df} \{head(h_i) \mid h_i \in \mathcal{P} \text{ and for all } a_j \in body(h_i): a_j \in I_{\mathcal{P}}\}.$$

Proposition 2.1: Let \mathcal{P} be a definite logic program. Then the mapping $T_{\mathcal{P}}$ is continuous.

Proposition 2.2: Let \mathcal{P} be a definite logic program and $I_{\mathcal{P}}$ be an interpretation. Then $I_{\mathcal{P}}$ is a model for \mathcal{P} iff $T_{\mathcal{P}}(I_{\mathcal{P}}) \subseteq I_{\mathcal{P}}$.

Theorem 2.3: Let \mathcal{P} be a definite logic program. Then $M_{\mathcal{P}} = lfp(T_{\mathcal{P}}) = T_{\mathcal{P}} \uparrow \omega$.

Proofs of the propositions 2.1, 2.2 and theorem 2.3 are described in [11, p. 37–38].

Definition 2.12 (Logical consequence of \mathcal{P}): Let \mathcal{P} be a definite logic program and $M_{\mathcal{P}}$ be a least Herbrand model of \mathcal{P} . Atom a_i is a logical consequence of \mathcal{P} iff it belongs to $M_{\mathcal{P}}$:

$$\mathcal{P} \models a_i \text{ iff } a_i \in M_{\mathcal{P}}.$$

Definition 2.13: Let \mathcal{P} be a definite logic program and $M_{\mathcal{P}}$ the least Herbrand model of \mathcal{P} . a_i is an atom and is called the *abductive goal* for \mathcal{P} (denoted by $\mathcal{G}_{\mathcal{P}}$) if it fulfils the following criterion:

$$\mathcal{G}_{\mathcal{P}} =_{df} a_i \text{ such that } a_i \notin M_{\mathcal{P}}.$$

Coming back to the abductive problem given in the Introduction, observe that the knowledge base can be expressed as a logic program $\mathcal{P} = \{A \leftarrow B, C; B \leftarrow C; B \leftarrow A\}$ and the least Herbrand model $M_{\mathcal{P}} = \emptyset$ (because $T_{\mathcal{P}} \uparrow (\emptyset) = \emptyset$) does not contain fact A , which is the abductive goal. This means that A is not entailed by \mathcal{P} .

III. NEURAL NETWORKS

Definition 3.1: We use the following language \mathcal{L}^N to describe neural networks:

- $\{i_1, \dots, i_n, \dots\}$ — an infinite, countable set of symbols for input neurons,
- $\{h_1, \dots, h_n, \dots\}$ — an infinite, countable set of symbols for hidden layer neurons,
- $\{o_1, \dots, o_n, \dots\}$ — an infinite, countable set of symbols for output neurons,
- $\{n_1, \dots, n_n, \dots\}$ — an infinite, countable set of symbols for meta variables representing any neuron,
- $\{n_1, \dots, n_n, \dots\}$ — an infinite, countable set of symbols for *labels* of neurons which are not associated with an atom from \mathcal{L} ,
- *label* — an identification symbol for a neuron,
- t — a label denoting *truth neuron*,
- tn — a label denoting hidden layer neuron reserved for truth neuron,
- $g(x), h(x)$ — neuron activation functions,
- $x \in \mathbb{R}$ — a weighted sum of the input signals for the neuron,
- $A_{min} \in \mathbb{R}$ — a value computed by algorithm [3, p. 48–50],
- $A_{min}^f \in \mathbb{R}$ — A_{min} enlarge factor,
- $\theta_i \in \mathbb{R}$ — threshold of a neuron,
- $\theta_a \in \mathbb{R}$ — threshold of additional neurons,
- $\beta \in \mathbb{R}$ — steepness of neuron activation function (used only for bipolar semi-linear activation functions),
- $W \in \mathbb{R}$ — a weight computed by the algorithm [3, p. 48–50],
- $W^f \in \mathbb{R}$ — W enlarge factor,
- $r \in \mathbb{R}$ — a variable,
- $l \in \mathbb{N}$ — the number of additional hidden layer neurons per each neuron in the output layer.

Definition 3.2: *Input neurons* are tuples of the form:

$$i_i =_{df} \langle label, g(x), A_{min}^m \rangle$$

where:

- *label* — a symbol of the represented atom form \mathcal{P} ,
- $g(x) = x$,
- $A_{min} \in \mathbb{R}$ — if $g(x) \geq A_{min}$ then *label* is mapped to *true*,
- the output signal of the neuron is equal to $g(x)$.

Definition 3.3: *Truth neuron* is an input neuron with the following properties:

- $label = t$,
- $g(x) = 1$.

Definition 3.4: *Hidden layer neurons* are tuples of the form:

$$h_i =_{df} \langle label, h(x), \theta_h, A_{min}^m \rangle$$

where:

- *label* — n_i associated with clause h_i from \mathcal{P} ,
- $h(x) = \frac{2}{1+e^{-\beta(x-\theta_h)}} - 1$,
- $\theta_h \in \mathbb{R}$,
- $A_{min} \in \mathbb{R}$ — if $h(x) \geq A_{min}$ then *label* is mapped to *true*,
- the output signal of the neuron is equal to $h(x)$.

Definition 3.5: *Output neurons* are tuples of the form:

$$o_i =_{df} \langle label, h(x), \theta_o, A_{min}^m \rangle$$

where:

- *label* — a symbol of the represented atom form \mathcal{P} ,
- $h(x) = \frac{2}{1+e^{-\beta(x-\theta_o)}} - 1$,
- $\theta_o \in \mathbb{R}$,
- $A_{min} \in \mathbb{R}$ — if $g(x) \geq A_{min}$ then *label* is mapped to *true*,
- the output signal of the neuron is equal to $h(x)$.

Definition 3.6: Let i_1, \dots, i_n be input neurons. By \mathcal{N}_I we denote the set of all input neurons:

$$\mathcal{N}_I = \{i_1, \dots, i_n\}.$$

Definition 3.7: Let h_1, \dots, h_n be input neurons. By \mathcal{N}_H we denote the set of all hidden layer neurons:

$$\mathcal{N}_H = \{h_1, \dots, h_n\}.$$

Definition 3.8: Let o_1, \dots, o_n be input neurons. By \mathcal{N}_O we denote the set of all output neurons:

$$\mathcal{N}_O = \{o_1, \dots, o_n\}.$$

Definition 3.9: Let $\mathcal{N}_I, \mathcal{N}_H$ and \mathcal{N}_O be the set of all input, hidden and output neurons respectively. By \mathcal{N} we denote the set of all neurons:

$$\mathcal{N} = \mathcal{N}_I \cup \mathcal{N}_H \cup \mathcal{N}_O.$$

Definition 3.10: Let i_i, i_k and h_j, h_l were input and hidden layer neurons respectively. By $\mathcal{C}_{i \rightarrow h}$ we denote the set of the connections from input to hidden layer neurons. The connection runs from the first neuron in the tuple to the second:

$$\mathcal{C}_{i \rightarrow h} = \{\langle i_i, h_j \rangle, \dots, \langle i_k, h_l \rangle\}.$$

Definition 3.11: Let h_i, h_k and o_j, o_l were hidden layer and output neurons respectively. By $\mathcal{C}_{h \rightarrow o}$ we denote the set

of the connections from hidden layer to output neurons. The connection runs from the first neuron in the tuple to the second:

$$\mathcal{C}_{h \rightarrow o} = \{\langle h_i, o_j \rangle, \dots, \langle h_k, o_l \rangle\}.$$

Definition 3.12: Let o_i and i_j were output and input neurons respectively. By \mathcal{C}_r we denote the set of the recursive connections from output to input neurons. The connection runs from the first neuron in the tuple to the second:

$$\mathcal{C}_r = \{\langle o_i, i_j \rangle \mid o_i(\text{label}) = i_j(\text{label})\}.$$

Definition 3.13: Let n_i, n_k, n_j, n_l be neurons. By \mathcal{C}_a we denote the set of the additional connections. The connection runs from the first neuron in the tuple to the second:

$$\mathcal{C}_a = \{\langle n_i, n_j \rangle, \dots, \langle n_k, n_l \rangle\}.$$

Definition 3.14: \mathcal{C} is the set of all connections in the network:

$$\mathcal{C} = \mathcal{C}_{i \rightarrow h} \cup \mathcal{C}_{h \rightarrow o} \cup \mathcal{C}_r \cup \mathcal{C}_a.$$

Definition 3.15: Let n_i, n_j be neurons. The function $w : \mathcal{C} \rightarrow \mathbb{R}$ establishes the weight of the connection between two connected neurons:

- if $\langle n_i, n_j \rangle \in \mathcal{C}_r$ then $w(\langle n_i, n_j \rangle) = 1$,
- if $\langle n_i, n_j \rangle \in \mathcal{C}_a$ then $w(\langle n_i, n_j \rangle) \in [-r, 0) \cup (0, r]$,
- otherwise $w(\langle n_i, n_j \rangle) = W^m$.

Definition 3.16: Let n_i, n_j be neurons. The set of all weights is denoted by \mathcal{W} . It consists of the tuple, where on the first and second place are connected neurons and on the third the weight of the connection:

$$\mathcal{W} = \{\langle n_i, n_j, w(n_i, n_j) \rangle \mid \langle n_i, n_j \rangle \in \mathcal{C}\}.$$

Definition 3.17: Neural network denoted by \mathfrak{N} is a tuple:

$$\mathfrak{N} =_{df} \langle \mathcal{N}, \mathcal{C}, \mathcal{W} \rangle.$$

Definition 3.18: Let \mathcal{P} be a definite logic program and $\mathcal{G}_{\mathcal{P}}$ an abductive goal. By $T_{\mathcal{P} \rightarrow \mathfrak{N}}$ we denote the translation from \mathcal{P} to \mathfrak{N} w.r.t. the set of predetermined factors $\{l, A_{min}^f, W^f, \beta, \theta_a, r\}$:

$$T_{\mathcal{P} \rightarrow \mathfrak{N}}(\langle \mathcal{P}, \{l, A_{min}^f, W^f, \beta, \theta_a, r\} \rangle) =_{df} \langle \mathcal{P}, \mathfrak{N} \rangle,$$

\mathfrak{N} is obtained from \mathcal{P} in the following way:

- 1) Calculate the following values by means of the algorithm [3, p. 48–50]:
 - A_{min} ,
 - W .
- 2) Calculate the following values:
 - $A_{min}^m = A_{min} + A_{min}^f$,
 - $W^m = W + W^f$.
- 3) For every atom $a_i \in B_{\mathcal{P}}$ an input neuron i_i is added to the set of input neurons \mathcal{N}_I . Properties of each input neuron are the following:
 - $label = a_i$.
- 4) For every clause $h_i \in \mathcal{P}$, if $body(h_i) \neq \emptyset$ then a hidden layer neuron h_i is added to the set of hidden layer

neurons \mathcal{N}_H . Properties of each hidden layer neuron are the following:

- $label = n_i$,
 - θ_o is computed as in the algorithm [3, p. 48–50].
- 5) For every atom in $a_i \in B_{\mathcal{P}}$ an output o_i neuron is added to the set of output neurons \mathcal{N}_O . Properties of each output neuron are the following:
 - $label = a_i$,
 - if $a_i \in B_{\mathcal{P}}^h$ then θ_o is computed as in the algorithm [3, p. 48–50], otherwise $\theta_o = \theta_a$.
 - 6) For $\mathcal{G}_{\mathcal{P}}$, $name = 'g_{\mathcal{G}_{\mathcal{P}}}'$ (letter 'g' is added to the propositional letter assigned to atom $\mathcal{G}_{\mathcal{P}}$) and:
 - if $\mathcal{G}_{\mathcal{P}} \notin B_{\mathcal{P}}$ then:
 - an input neuron is added to the set of input neurons \mathcal{N}_I where: $label = name$,
 - an output neuron is added to the set of output neurons \mathcal{N}_O where: $label = name, \theta_o = \theta_a$,
 - otherwise:
 - $label$ fields in neurons associated with $\mathcal{G}_{\mathcal{P}}$ in input and output set of neurons are changed to $name$.
 - 7) Truth neuron t is added to the set of input neurons \mathcal{N}_I .
 - 8) A hidden layer neuron is added to the set of hidden layer neurons \mathcal{N}_H with the following properties:
 - $label = tn$,
 - $\theta_h = 0$.
 - 9) For each neuron in the set of output neurons \mathcal{N}_O add l additional neurons to the hidden layer with the properties (the overall number of the additional neurons is: $k = l \cdot c(\mathcal{N}_O)$):
 - $label = an_i$, where $i \in [1, k]$,
 - $\theta_h = 0$.
 - 10) Generate the set of all neurons \mathcal{N} .
 - 11) For every $h_i \in \mathcal{P}$:
 - for every $a_j \in body(h_i)$ add a tuple $\langle i_j, h_i \rangle$ to the set $\mathcal{C}_{i \rightarrow h}$, where $i_j(\text{label}) = a_j$,
 - for $a_k = head(h_i)$:
 - if $body(h_i) = \emptyset$ then add tuple $\langle h_t, o_k \rangle$ to the set $\mathcal{C}_{h \rightarrow o}$, where $h_t(\text{label}) = tn$ and $o_k(\text{label}) = a_k$,
 - otherwise add tuple $\langle h_i, o_k \rangle$ to the set $\mathcal{C}_{h \rightarrow o}$, where $o_k(\text{label}) = a_k$.
 - 12) For every $o_i \in \mathcal{N}_O$ add tuples $\langle h_j, o_i \rangle$ to the set \mathcal{C}_a , where $h_j(\text{label}) = an_k$. Every output neuron o_i should be connected with l additional hidden neurons that are assigned to it².
 - 13) For every $o_i \in \mathcal{N}_O$: add tuples $\langle h_t, o_i \rangle$ to the set \mathcal{C}_a if :
 - $h_t(\text{label}) = tn$ and
 - $\langle h_t, o_i \rangle \notin \mathcal{C}_{h \rightarrow o}$.

²For example: we have 2 output neurons and we set the number of additional neurons per output neuron $l = 2$. In this case we establish connections between the first two additional hidden layer neurons with the first output neuron, and the other two additional hidden layer neurons with the second output neuron.

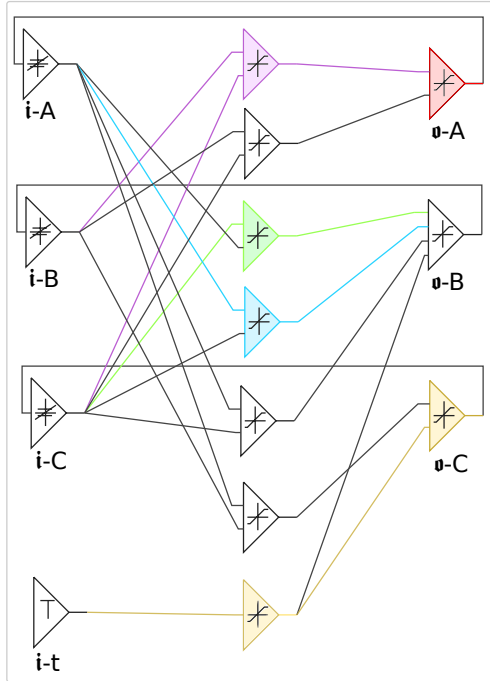


Fig. 1. A diagram of a neural network generated for the program in Example 1. The input neurons are labelled $i-X$, where X is a letter referring to an atom in the body of a clause, and output neurons are labelled $o-Y$, where Y refers to an atom in the head of a clause. The rest of the neurons are hidden layer units which represent possible clauses. Red color represents an abductive goal (fact A), purple color represents formula $A \leftarrow B, C$, green color represents formula $B \leftarrow C$, blue color represents formula $B \leftarrow A$. Yellow color represents the abductive hypothesis obtained for the problem (a fact C).

- 14) For every $i_i \in \mathcal{N}_I$, for every $h_j \in \mathcal{N}_H$: add tuple $\langle i_i, h_j \rangle$ to the set \mathcal{C}_a if:
 - $h_j(\text{label}) \neq tn$ or
 - $\langle h_j, o_z \rangle \notin \mathcal{C}_{h \rightarrow o}$, where $i_i(\text{label}) = o_z(\text{label})$.
- 15) Generate the set of recursive connections \mathcal{C}_r .
- 16) Generate the set of all connections \mathcal{C} .
- 17) Generate the set of all weights \mathcal{W} .
- 18) Generate the neural network \mathfrak{N} .

The result of the application of the above procedure to the logic program obtained from the exemplary problem is depicted in Fig. 1.

Definition 3.19: Let \mathfrak{N} be a neural network. By $T_{\mathfrak{N} \rightarrow \mathcal{P}}$ we denote the translation from \mathfrak{N} to \mathcal{P} :

$$T_{\mathfrak{N} \rightarrow \mathcal{P}}(\mathfrak{N}) =_{df} \langle \mathfrak{N}, \mathcal{P} \rangle,$$

where \mathcal{P} is obtained from \mathfrak{N} by using the pedagogical approach described in [3, Chapter 5].

Using the naive version of pedagogical approach we simply map input neurons into output neurons in the following way: for each output neuron we check all combinations of input values for all input neurons relevant for this output neuron. In the case of the activation of the considered output neuron we associate the atom represented by this neuron with the following clause h_i : $head(h_i)$ is the considered atom and the

body consists of the atoms represented by input neurons which were set to 1.0 and the negated atoms represented by input neurons that were set to -1.0 . For example for output neuron with $label = C$ with only one relevant input neuron with $label = A$, if it is the case that for the considered input neuron $g(x) = 1.0$ and for the output $h(x) > A_{min}$ then we obtain the clause $C \leftarrow A$; analogically, if it is the case that for the considered input neuron $g(x) = -1.0$ and for the output $h(x) > A_{min}$ then we obtain the clause $C \leftarrow nA$ (where nA means negated A). The set of clauses extracted from a neural network is minimized by means of the the Quine-McCluskey algorithm [7, 12].

The results of learning and translation of the running example are presented in Section VI-A.

IV. ABDUCTION

Traditionally abduction can be seen as a process of searching for explanations of a problem during which *additional* information is obtained (see [1, p. 74] on abductive explanatory characterization styles). Formally speaking, whenever we have some knowledge base K and abductive goal A , the abduction leads to generation of some additional formulas h_1, \dots, h_n , that are not present in K , but which together with formulas in K enable derivation of A . In the approach presented in this paper, this does not necessarily has to be the case. It is possible that after a training of the network which represents logical program/knowledge base with an example that represents an abductive goal, translation of the trained network back to the form of logical clauses will change the initial program in other ways than just extending it. It may happen that some formulas will be modified (e.g. by removing or adding some atoms in the body of some clauses) or they can be even removed from the initial program (it is probably a matter of discussion whether it is a desirable phenomenon or not). In our approach each such modification can be seen as an abductive hypothesis, hence the term *hypothesis* gain somewhat dynamic character. This approach is more flexible than the traditional one and allows more interesting conceptual applications for the abduction, in particular accomodating substantial revisions of the initial knowledge base (see abductive schematics in [2, p. 47]). The general scheme of the proposed procedure is depicted in Fig. 2.

The abductive procedure begins in the left upper corner of the scheme presented in Fig. 2. The knowledge base is represented by the definite logic program \mathcal{P} and there is a fact ϕ which cannot be derived from the knowledge base. The fact ϕ is of the form of an atom a_i and the abductive problem is represented by the fact that $a_i \notin M_{\mathcal{P}}$.

The first step of the abductive procedure is the translation of the \mathcal{P} to a neural network \mathfrak{N} . The first step of the translation is obtained by means of the algorithm developed by Garcez in [3, p. 49]. The difference is that we add all atoms from the $B_{\mathcal{P}}$ along with the a_i to the input (\mathcal{N}_I) and output (\mathcal{N}_O) layer of the \mathfrak{N} (steps 3 and 5 in $T_{\mathcal{P} \rightarrow \mathfrak{N}}$). In case of absence of the *facts* in \mathcal{P} , we also add a truth neuron t (which gives always 1 on the output) to \mathcal{N}_I (step 7 in $T_{\mathcal{P} \rightarrow \mathfrak{N}}$). The hidden layer

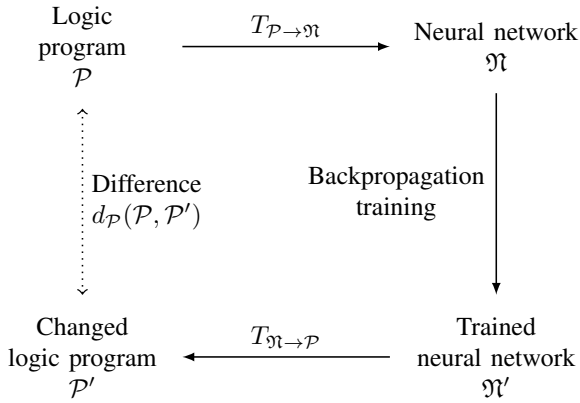


Fig. 2. A scheme of the abductive procedure

(\mathcal{N}_H) is modified in the following way: we add m neuron, which is reserved only for t neuron from \mathcal{N}_I , and a number of additional atoms per clause and a_i (steps 8 and 9 in $T_{\mathcal{P} \rightarrow \mathcal{N}}$).

The set of all connections from input to hidden layer ($\mathcal{C}_{i \rightarrow h}$) is enriched by the additional connections running from every atom in \mathcal{N}_I to \mathcal{N}_H with the exception of t and m neurons. There is a unique connection between t and m . Additional connections do not double the connections obtained by the previous step. We also forbid additional connections from input to the hidden layer neurons that are connected with an output neuron with the same label as the neuron from input layer. In other words, additional connections cannot allow to formulate formulas of the form $A \leftarrow A$. There are also additional connections added to the set of connections from hidden to output layer ($\mathcal{C}_{h \rightarrow o}$). Those connections run from additional neurons assigned to the particular clause from \mathcal{P} to the neuron which represents the head of the concerned clause. The neuron m from the hidden layer connects with every neuron from output layer, except for the neuron which represents the fact ϕ denoted by a_i . The function w gives every additional connection a random value from the range $[-r, r]$ (with the exception of 0).

After the whole neural network \mathcal{N} is completed, the training begins with the use of the standard backpropagation algorithm. The training set consists of only one element: a table with all input atoms set to -1 and all output atoms set to 1 . Error calculation is performed after \mathcal{N} achieves the stable state.

Trained neural network \mathcal{N}' is then translated back to logic program \mathcal{P}' . We define the difference between \mathcal{P} and \mathcal{P}' as $d_{\mathcal{P}}(\mathcal{P}, \mathcal{P}')$.

Definition 4.1: Let \mathcal{P} and \mathcal{P}' were a definite logic programs. The difference between \mathcal{P} and \mathcal{P}' is denoted by $d_{\mathcal{P}}(\mathcal{P}, \mathcal{P}')$:

$$d_{\mathcal{P}}(\mathcal{P}, \mathcal{P}') = (\mathcal{P}' \setminus (\mathcal{P} \cap \mathcal{P}')) \cup (\mathcal{P} \setminus (\mathcal{P} \cap \mathcal{P}')).$$

The change of the logic program defined as a symmetric difference is interpreted as a set of abductive hypotheses.

Definition 4.2: Let \mathcal{P} be a definite logic program and \mathcal{N} a neural network obtained from \mathcal{P} by the translation $T_{\mathcal{P} \rightarrow \mathcal{N}}$. Let us further assume that \mathcal{N}' is obtained from \mathcal{N}

by backpropagation training described above. After translation of the \mathcal{N}' to \mathcal{P}' by translation $T_{\mathcal{N}' \rightarrow \mathcal{P}'}$ the set of abductive hypotheses $H_{\mathcal{P}}$ can be defined in the following way:

$$H_{\mathcal{P}} = d_{\mathcal{P}}(\mathcal{P}, \mathcal{P}').$$

V. IMPLEMENTATION

To implement the ideas given above, we have decided to use Framsticks software [10] — a versatile tool which among its many merits, gives the possibility to perform computational experiments concerning artificial neural networks. Framsticks platform is equipped with an advanced scripting language that easily enables any kind of experiment. This, together with an advanced network simulator, makes it a suitable tool for the research presented in our article. Apart from that, the software was already used in computational experiments concerning logical abduction [8, 9].

The whole implementation can be seen as a general framework consisting of scripts representing operations described earlier in the text. Hence, the whole abduction experiment can be represented as the knowledge flow between different scripts which is schematically depicted in Fig. 2.

The first important part implemented is the algorithm of translation of the default logic programs into one-hidden layer neural networks described in Sect. IV.

Next, Backpropagation algorithm with momentum has been programmed as it was not available in Framsticks software that is mainly focused on evolutionary optimisation. Finally, the algorithm translating (trained) neural networks into logic programs, described in Sect. IV has been also implemented. The whole learning procedure, however, has been modified (in comparison to the standard application of Backpropagation algorithm) and adapted to the needs of the presented research. The training set for any abductive problem consists of only one training example which in its input part contains only -1.0 values. This represents the situation where the operator $T_{\mathcal{P}}$ starts from the bottom of the lattice of all possible interpretations of \mathcal{P} . The values in the output part are now selected arbitrarily (apart from the value related to the neuron representing an abductive goal, which is always set to 1.0), but as mentioned later in Sect. VII solving this issue is one of the immediate future tasks. The scheme of learning is also modified as each change of the weights is based on the error calculation performed after the network achieves stable state (after several cycles of signal propagation). The network-to-program translation is implemented using the brute force approach (sometimes called *pedagogical*, which is of exponential complexity with respect to the number of input neurons), that is inefficient, especially for more complicated problems. However, as complexity-reducing approaches may be associated with lack of completeness and soundness of the translation [3, Chapter 5] and a general view on the abductive process is needed at the moment, we have decided to temporarily pay the price of the computational cost. The future plans include reduction of complexity of the used methods. Yet another issue related to the network-to-program

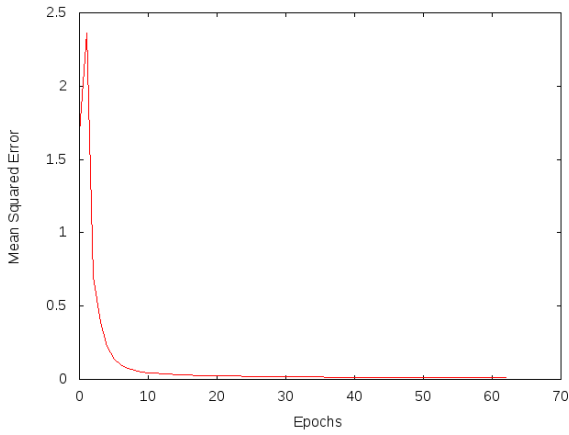


Fig. 3. A learning error chart for the Example 1.

translation is that it demands several stages of processing of the obtained clauses. In the present approach the first stage of translation may result in obtaining redundant information (e.g. two formulas of the form $A \leftarrow B, \neg C$ and $A \leftarrow B, C$ can be obtained, which should be reduced to $A \leftarrow B$).

VI. RESULTS

This section presents preliminary results for two toy-example problems. The first example is presented in more detailed way to demonstrate how the procedure works and it concerns the problem provided in the introduction.

A. Example 1

The knowledge base for the Example 1 is $\mathcal{P} = \{A \leftarrow B, C; B \leftarrow C; B \leftarrow A\}$ and the abductive goal is a fact A . According to the procedure described in Sect. IV, the network obtained after translation of \mathcal{P} is extended to contain additional hidden-layer nodes and connections to enable generation of new concepts in the knowledge base. To perform learning we prepared a training example in which every input neuron sends a signal of value -1.0 and every output value should be equal 1.0 .

The learning procedure is a modified version of a Back-propagation algorithm, as mentioned earlier and in Sect. IV to let the network compute the fix point of the program.

The learning error is presented in Fig. 3. As it can be seen the error was minimized rather quickly (after a bit more than 60 epochs), which is not surprising as the training set contained only one example.

The translation of the network back to the form of a logical program resulted in the following set of formulas $\mathcal{P}_t = \{A \leftarrow B, C; B \leftarrow nC, A, t; B \leftarrow C, nA, t; B \leftarrow C, A, t; C \leftarrow nA, nB, t; C \leftarrow nA, B, t; C \leftarrow A, nB, t; C \leftarrow A, B, t\}$, where t stand for truth and n is a negation (which at this stage may be interpreted as negation as a failure — however it is removed during reduction of the obtained logic program). Further reduction of the obtained set of formulas resulted in the set $\mathcal{P}_{tr} = \{A \leftarrow B, C; B \leftarrow C; B \leftarrow A; C\}$. This means

that the initial knowledge base was extended by a fact C , which is an abductive hypothesis for the problem.

B. Example 2

Example 2 demonstrates an effect of increase of the number of hidden neurons (which represent the potential concepts that can be learned by a network, see Sec. IV) and the role of the initial state of the network (represented by randomly initiated weights of the additional connections). In order to preliminarily examine these effects we performed the abductive procedure several times with other parameters fixed. The knowledge base is $\mathcal{P} = \{A \leftarrow B; A \leftarrow C; D \leftarrow E; C \leftarrow E; B \leftarrow C\}$ and an abductive goal is a fact A . The training example in the output part contained only 1.0 values. The most frequent program obtained from a network containing *one* additional hidden neuron per each output neuron is $\mathcal{P}_{t1} = \{A \leftarrow B; A \leftarrow C; D \leftarrow E; C \leftarrow E; B \leftarrow C; E\}$ — which means that the abductive hypotheses is E . The most frequent definite logic³ program obtained from the trained networks with 17 additional hidden-layer neurons per output neuron is $\mathcal{P}_{t2} = \{A \leftarrow B; A \leftarrow C; D; C; B \leftarrow C; E\}$. This means that the abductive hypothesis is: E and change of formulas $D \leftarrow E$ and $C \leftarrow E$ into respectively D and C . This example illustrates two phenomena. Firstly, the knowledge base can be modified during the learning process: for example clause $D \leftarrow E$ has been strengthened into a fact D . Secondly, the number of hidden-layer neurons can influence the obtained solutions for abductive problems. These two observations in turn mean, that a size of hidden layer represents a “reasoning potential” of a network and may possibly be used later to control the quality of obtained hypotheses. In the presented exemplary result, a greater number of the hidden neurons resulted in the lowered quality of the obtained hypothesis: this is because, D does not help with the derivation of A and the hypothesis contains redundancy as A can be derived from both E and C . Moreover, C can be derived from E . This lowers the internal minimality of the hypothesis, understood as non-derivability of one of the subformula of the hypothesis from another. Note, however, that the increased number of neurons is not completely unpromising, as the best solution (obtained most frequently for networks with the lower number of hidden neurons) also appeared in the set of possible solutions, but much less often.

Interestingly, further analyses revealed one more interesting issue related to the increased number of hidden neurons and connections. Apparently, in the larger networks the abductive processes became more sensitive to the initial state of the connections (with randomly initialized weights). The repetitions of the abductive experiments resulted in establishing different abductive hypotheses and this diversity was bigger than in the case of the network with lower number of neurons. The knowledge bases obtained from the subsequently trained networks with the larger number of neurons were represented i.a. by the

³In this work we purposely left out of analysis other kinds of logic programs, however the most frequent program obtained with these set of parameters was actually a general logic program.

following sets of formulas: $\{A \leftarrow B; A \leftarrow C; D \leftarrow E; C \leftarrow E; B; E\}$, $\{A \leftarrow B; A \leftarrow C; D \rightarrow E; C; B; E\}$. These few results preliminarily demonstrate that random initialization of the network which is capable of acquiring new concepts (due to the additional hidden neurons and connections), may influence the degree of modification of the initial knowledge and the quality of the abductive hypotheses.

VII. SUMMARY AND FUTURE WORK

In this paper we presented an attempt at synthesizing formal description of abductive problems with connectionist methodology based on work of Garcez et al. [3]. Contrary to the received approaches to the problem of logical abduction, we decided to employ Backpropagation algorithm to search for abductive hypotheses. The results of such experiments show that training of an artificial neural network can indeed be a method of filling a gap between a knowledge base and an abductive goal.

The neural networks obtained according to the presented procedure can be further used to solve real life problems as classifiers (just like traditional artificial neural networks are usually used) – examples of such applications of *C-IL²P* are presented in [3, Chapter 4]. Apart from that, the trained networks can be used as massively parallel deduction systems to solve logical problems. Yet another advantage of the presented approach, which at the same time distinguishes it from often discussed, pure logical accounts, is that it offers broad and flexible definition of the abductive hypothesis. The obtained hypotheses can either be additional formulas extending the initial knowledge base or some modifications done to the knowledge base. Finally, such methodology can be used for modelling abduction as a cognitive process, by combining connectionist structures, resembling in the limited sense the human brain, with rigours of the formal logic. The flexibility of the concept of the abductive hypothesis is likely to increase the accuracy of such modelling.

One of the interesting observations obtained at this preliminary stage of the currently presented research is the joint influence of the size of the trained network and its initial state on the obtained abductive hypotheses. While such observations may seem a little vague, more advanced computational experiments and quantitative analyses are currently under way. Among them, interesting research task is examination of the influence of the different parameters of the networks (i.e. biases of the additional neurons, initial weights of the additional connections, etc.) and parameters of training process (e.g. a form of the training example, value of learning rate or momentum) on the resulting knowledge. Research on the influence of interaction of these parameters with characteristics of the considered problems (e.g. a number of clauses in the knowledge base, a number of propositional variables, some more sophisticated structural traits) on the resulting knowledge (understood as a logic program obtained from the trained neural network) is also possible. This kind of research demands development of some measures of similarity between different knowledge bases, which is an interesting issue in

itself and gives possibility of performing quantitative multi-dimensional analyses of the results of abductive process.

In the nearest future a number of improvements to the existing implementations are also in order. Among them, the most pending ones are: a decrease of computational complexity of the network-to-program implementation and intelligent automatic translation of an abductive goal into the form of a training example. This encompasses formulation of the training example in such a way that knowledge base is properly represented.

The further plans concern extension of our methodology to include more types of logic programs in order to introduce a negation, default or classical, into the researched abductive problems. The introduction of a negation will require even more sophisticated approach to generation of training examples containing representations of abductive goals. Additionally, as it is straightforward for our methodology, we would like to introduce the abductive goals in the form of formulas more complex than single atoms.

To solve the issue concerning generation of training examples a computational approach may be employed. The computational experiments may concern analysis of the impact of the desirable output values on the obtained modifications of the knowledge base. The potential results may shed light on how to construct an initial training example to properly reflect an abductive goal alongside desired concepts from the knowledge base.

Finally, a well-established abduction research requires employment of some quality-control mechanisms to obtain efficient abductive hypotheses. This goal is partially achieved in the presented approach heuristically, by removal of some unwanted connections, careful generation of the training examples and (possibly) by imposing restrictions on the size of the trained network. However, employment of some more sophisticated quality criteria is still a pending issue. An interesting approach concerning application of multi-criteria dominance relation [8, 9] to evaluate already generated abductive hypotheses gives the possibility to perform a comparative study with the methodology presented in this paper.

APPENDIX

Definition 7.1: A binary relation R on a set is called a *partial order* when it is reflexive, transitive and antisymmetric.

We will denote a relation which is a partial order by \preceq . A set S with a partial order \preceq is called a *partially ordered set*.

Definition 7.2: Let S be a set with a partial order \preceq and $x \in S$. We define the following:

- x is a *minimum* of S if for all elements $y \in S$: $x \preceq y$.
- x is a *maximum* of S if for all elements $y \in S$: $y \preceq x$.

Minimum and maximum of a set S are unique, if they exist, and will be denoted as $\inf(S)$ and $\sup(S)$ respectively.

Definition 7.3: Let S be a set with a partial order \preceq and $R \subseteq S$. We define the following:

- An element $x \in S$ is an *upper bound* of R if for all elements $y \in R$: $y \preceq x$.

- An element $x \in S$ is an *lower bound* of R if for all elements $y \in R$: $x \preceq y$.
- An element $x \in S$ is *least upper bound* of R if x is an upper bound of R and $x \preceq z$ for all upper bounds z of R .
- An element $x \in S$ is *greatest lower bound* of R if x is an lower bound of R and $z \preceq x$ for all lower bounds z of R .

The least upper bound and the greatest lower bound of a set R are unique, if they exist, and will be denoted as $\text{lub}(R)$ and $\text{glb}(R)$ respectively.

Definition 7.4: Let S be a partially ordered set and $R \subseteq S$. We call S a *complete lattice* if $\text{lub}(R)$ and $\text{glb}(R)$ exist for every $R \subseteq S$.

Definition 7.5: Let S be a complete lattice and $R \subseteq S$. We call R *directed* if every finite subset of R has an upper bound in R .

Definition 7.6: Let S be a complete lattice, x and y be elements of S , and $T: S \rightarrow S$ be a mapping. The following holds:

- T is *monotonic* if $T(x) \preceq T(y)$, where $x \preceq y$.
- T is *continuous* if for every directed subset R of S : $T(\text{lub}(R)) = \text{lub}(T(R))$.

The collection of all Herbrand interpretations of a definite logic program P , which is 2^{B_P} , forms a complete lattice under the partial order of set inclusion. The top and the bottom element of 2^{B_P} is B_P and \emptyset respectively. There can be described a continuous and monotonic mapping from 2^{B_P} to 2^{B_P} which will serve in finding the least Herbrand model of P . The procedure is based on fixpoints of mappings on the set 2^{B_P} .

Definition 7.7: Let S be a complete lattice and $T: S \rightarrow S$ be a mapping. An element $x \in S$ is the *least fixpoint* of T if x is a fixpoint of T (i.e. $T(x) = x$) and for all fixpoints y of T : $x \preceq y$. An element $x \in S$ is the *greatest fixpoint* of T if x is a fixpoint of T and for all fixpoints y of T : $y \preceq x$.

The least fixpoint of T and the greatest fixpoint of T will be denoted as $\text{lfp}(T)$ and $\text{gfp}(T)$ respectively.

Proposition 7.1: Let S be a complete lattice and $T: S \rightarrow S$ be monotonic. T has $\text{lfp}(T)$ and $\text{gfp}(T)$.

The proof of the theorem 7.1 is described in [11, p. 28].

Now we want to define *ordinal powers of T* . The definition is based on properties of ordinal numbers described in [11, p. 28–29] or [3, p. 26].

Definition 7.8: Let S be a complete lattice and $T: S \rightarrow S$ be monotonic. Then we define:

$$\begin{aligned} T \uparrow 0 &= \text{inf}(S); \\ T \uparrow \alpha &= T(T \uparrow (\alpha - 1)), \text{ if } \alpha \text{ is a successor ordinal}; \\ T \uparrow \alpha &= \text{lub}(T \uparrow \beta \mid \beta \prec \alpha), \text{ if } \alpha \text{ is a limit ordinal}; \\ T \downarrow 0 &= \text{sup}(S); \\ T \downarrow \alpha &= T(T \downarrow (\alpha - 1)), \text{ if } \alpha \text{ is a successor ordinal}; \\ T \downarrow \alpha &= \text{glb}(T \downarrow \beta \mid \beta \prec \alpha), \text{ if } \alpha \text{ is a limit ordinal}. \end{aligned}$$

Proposition 7.2: Let S be a complete lattice and $T: S \rightarrow S$ be continuous. Then $\text{lfp}(T) = T \uparrow \omega$.

The ω in theorem 7.2 denotes the first infinite ordinal. Proof of the theorem 7.2 is described in details in [11, p. 30].

REFERENCES

- [1] Atocha Aliseda. *Abductive Reasoning. Logical Investigations into Discovery and Explanation*. Springer, Dordrecht, 2006. doi: 10.1007/1-4020-3907-7.
- [2] Dov M. Gabbay and John Woods. *The Reach of Abduction. Insight and Trial*. Elsevier, 2005. doi: 10.1016/S1874-5075(05)80034-8.
- [3] Artur S. d’Avila Garcez, Krysia Broda, and Dov M. Gabbay. *Neural-symbolic learning systems: foundations and applications*. Springer Science & Business Media, 2002. doi: 10.1007/978-1-4471-0211-3.
- [4] Artur S d’Avila Garcez, Dov M Gabbay, Oliver Ray, and John Woods. Abductive reasoning in neural-symbolic systems. *Topoi*, 26(1):37–49, 2007. doi: 10.1007/s11245-006-9005-5.
- [5] Peter Gärdenfors. *Belief Revision*. Tracts in Theoretical Computer Science 29. Cambridge University Press, 2003. doi: 10.1017/CBO9780511526664.
- [6] Jaakko Hintikka. Abduction — inference, conjecture, or an answer to a question? In *Socratic Epistemology. Explorations of Knowledge-Seeking by Questioning*, pages 38–60. Cambridge University Press, 2007. doi: 10.1017/CBO9780511619298.003.
- [7] Tarun Kumar Jain, Dharmender Singh Kushwaha, and Arun Kumar Misra. Optimization of the quine-mccluskey method for the minimization of the boolean expressions. In *Fourth International Conference on Autonomic and Autonomous Systems (ICAS’08)*, pages 165–168. IEEE, 2008. doi: 10.1109/ICAS.2008.11.
- [8] M. Komosinski, A. Kups, and M. Urbański. Multi-criteria evaluation of abductive hypotheses: towards efficient optimization in proof theory. In *Proceedings of the 18th International Conference on Soft Computing*, pages 320–325, Brno, Czech Republic, 2012.
- [9] M. Komosinski, A. Kups, D. Leszczyńska-Jasion, and M. Urbański. Identifying efficient abductive hypotheses using multi-criteria dominance relation. *ACM Transactions on Computational Logic*, 15(4), 2014. doi: 10.1145/2629669.
- [10] Maciej Komosinski and Szymon Ulatowski. Framsticks web site, 2016. <http://www.framsticks.com>.
- [11] John Wylie Lloyd. *Foundations of logic programming*. 1993. doi: 10.1007/978-3-642-83189-8.
- [12] Edward J McCluskey. Minimization of boolean functions. *Bell system technical Journal*, 35(6):1417–1444, 1956. doi: 10.1002/j.1538-7305.1956.tb03835.x.
- [13] Noel Pérez, Miguel Angel Guevara, Augusto Silva, Isabel Ramos, and Joana Loureiro. Improving the performance of machine learning classifiers for breast cancer diagnosis based on feature selection. In M. Paprzycki M. Ganzha, L. Maciaszek, editor, *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, volume 2 of *Annals of Computer Science and Information Systems*, pages 209–217. IEEE, 2014. doi: 10.15439/2014F249. URL <http://dx.doi.org/>

- 10.15439/2014F249.
- [14] P. Thagard. Abductive inference: From philosophical analysis to neural mechanisms. In A. Feeney and E. Heit, editors, *Inductive reasoning: Cognitive, mathematical, and neuroscientific approaches*, pages 226–247. Cambridge University Press, Cambridge, 2007. doi: 10.1017/cbo9780511619304.010.
- [15] Agnieszka Wosiak and Danuta Zakrzewska. On integrating clustering and statistical analysis for supporting cardiovascular disease diagnosis. In M. Ganzha, L. Maciaszek, and M. Paprzycki, editors, *Proceedings of the 2015 Federated Conference on Computer Science and Information Systems*, volume 5 of *Annals of Computer Science and Information Systems*, pages 303–310. IEEE, 2015. doi: 10.15439/2015F151. URL <http://dx.doi.org/10.15439/2015F151>.